bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Modeling heterogeneous populations using Boolean networks

Brian C. Ross[1,2], Mayla Boguslav[1,2], Holly Weeks[3], James Costello[1,2*]

[1]Computational Bioscience Program,

[2]Department of Pharmacology,

[3]Department of Biostatistics and Informatics,

University of Colorado Anschutz Medical Campus, Aurora, CO, USA

Abstract

Boolean networks are commonly used to model biological pathways and processes, in part because analyses can often find all possible long-term outcomes. Here we describe a Boolean network analysis that captures both the long-term outcomes of a heterogeneous population, as well as the transient behavior leading up to those outcomes. In contrast to other approaches, our method gives an explicit simulation through time using the composition of the mixed population without having to track each subpopulation individually, thus allowing us to simulate heterogeneous populations. This technique accurately models the dynamics of large populations of deterministic, probabilistic or continuous-time Boolean networks that use either synchronous or asynchronous updating. Our method works by treating the network dynamics as a linear system in a variable space that includes products of the Boolean state variables. We show that these product-basis analyses can help find very rare subpopulations or behaviors that sampling-based analyses would likely miss. Such rare events are critical in processes such as the initiation and progression of cancer, and the development of treatment resistance.

Mathematical models are an important part of testing and extrapolating our knowledge of biological systems [1], but they can be difficult to fully analyze. A straightforward way to study a model is to simulate individual instances of that model using a random sampling technique such as Monte Carlo [2]. These simulations can be run very efficiently, allowing the use of complex models extending even to whole-cell simulations [3]. A major drawback to random sampling is that simulations have difficulty capturing rare events such as those that initiate biological processes leading to novel and potentially disease-related cellular phenotypes [4]. For example, tumor initiation, progression, and the survival of select cells following drug treatment all require rare alterations to arise and clonally expand to eventually dominate the population in the long run [5–7]. While one can bias Monte Carlo to oversample certain outcomes by artificially raising or lowering global parameters such as a mutation rate, and then post-correct for the biased sampling (a strategy known as importance sampling [8]), this does not help find outcomes that are rare just because they require a very particular starting state or set of mutations.

An alternative approach to random sampling techniques is to study a model analytically in order to learn about *all* possible behaviors or outcomes even if they are rare. Analytic results are difficult to obtain with complex models, but significant advances have been made in analyzing Boolean networks [9–16], which are very simple models built entirely from ON/OFF variables. In particular, the focus has been on extracting the possible long-term outcomes, or *attractors*, of Boolean models. An attractor may be a stable state (steady state) or else a repeating sequence of states (limit cycle). Attractors have been found using network-reduction algorithms that find simple networks encoding the long-term behavior of more complex networks [9, 11, 17], methods that solve steady states as zeros of a polynomial equation [18], SAT methods [13, 14, 19], and binary decision diagrams [15, 16, 20]. See the introduction of Ref. [10] for a review of these techniques.

In between Monte Carlo simulations of individuals and attractor analyses of populations, there remains an unaddressed challenge: methods dealing directly with populations do not explicitly track their dynamics in the way that a simulation does. Therefore the power of exact analyses has not been applied to the early-time 'transient' behavior of populations, or used to connect initial states of different subpopulations to the attractors they fall into.

*E-mail: james.costello@ucdenver.edu

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Here we present an analytical method for simulating a heterogeneous population of Boolean networks, without having to simulate each network individually. The simulations are exact, so they capture every subpopulation of the model and every event that occurs, no matter how rare. The feasibility of building these simulations depends on the size and topology of the network (for complex or highly-recurrent networks it can be an exponential problem), though we can simplify the difficult cases by ignoring the first few time steps of the simulation. In the limit where we focus only on the longest-term behavior, the output describes the attractors of the network.

Our method applies to deterministic [1], probabilistic [21] and continuous-time [22] Boolean networks, and finds all attractors with equal computational effort (some attractor-finding techniques have more difficulty with limit cycles than steady states). In this paper we only analyze synchronous Boolean network models (i.e. networks whose variables all update together in discrete time steps). However, we argue that an asynchronous network can be accurately modeled as a synchronous probabilistic network that is typically easier to analyze than the original network would be if it were synchronous. One major benefit to our approach is its simplicity, as it follows only two rules: 1) work in a linear basis whose variables are products of the Boolean state variables, and 2) ignore quickly-decaying modes if we are looking at late-time behavior. Based on (1) we refer to our analysis as a *product-basis* method.

Results

To demonstrate our method, we applied it to the T-cell activation network described in Ref. [23] (see Figure 10 and Table 2 of that paper). This is a deterministic, 40-node network with fifty-four edges containing multiple feedback loops, and whose attractors include both steady states and limit cycles. To use our method, we first provided a target set of variables to follow in time, which the product-basis algorithm used to generate a set of time-evolution equations involving those variables (along with other variables that were added automatically to close the system of equations). We chose to track three variables: the ligand-binding state of the T-cell receptor $\text{TCR}^{\text{BOUND}}$, the phosphorylation state TCR^{P}, and the co-occurrence of binding and phosphorylation. The starting population we considered was a uniform mixture of all possible 2^{40}($\approx$ 1 trillion) initial states of the Boolean variables. We generated the product-basis time-evolution equations, and used them to track the population-level average of each of these three variables for 50 time steps (Figure 1A). It should be noted that the co-occurrence variable $\text{TCR}^{\text{BOUND}}$ AND TCR^{P} is not simply the product $\text{TCR}^{\text{BOUND}} \times \text{TCR}^{\text{P}}$, because $\text{TCR}^{\text{BOUND}}$ and TCR^{P} could be correlated or anti-correlated to some degree.

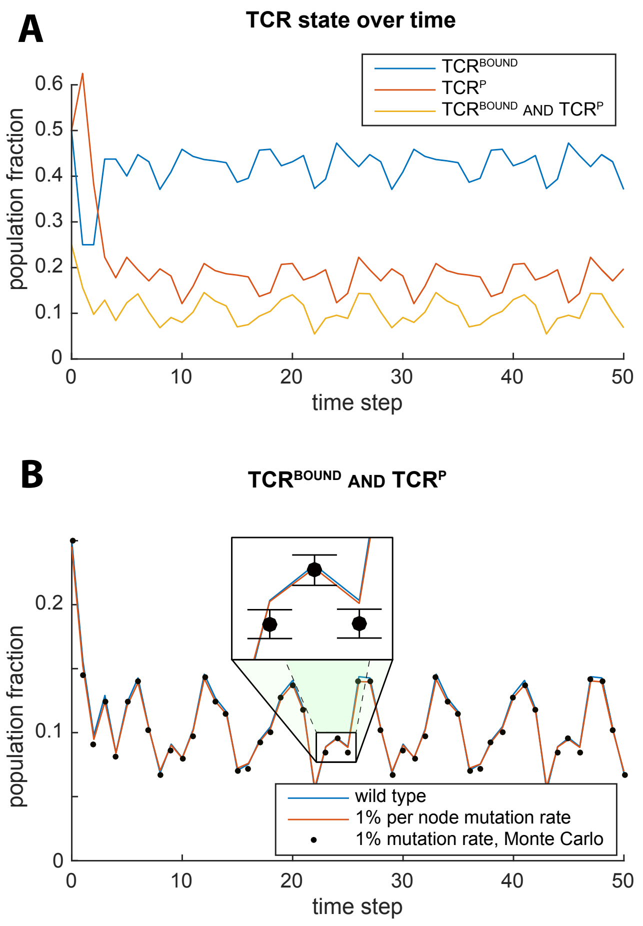

Figure 1. A) Time-evolution of the mean TCR state in a heterogeneous population, based on the model of Ref. [23]. The population begins at $t = 0$ as a uniform mixture of all possible starting states. **B)** The effect of a 1% knock-out mutation rate per gene on the time evolution of the co-occurrence $\text{TCR}^{\text{BOUND}}$ AND TCR^{P} in the population. The slight difference between a 0% and 1% mutation rate per gene is too small to reliably resolve by Monte Carlo, despite the fact that a third of the population has at least one mutation.

Next we demonstrated the ability of the product-basis method to analyze mutations in the network by including the full set of possible gene knock-outs in the T-cell activation network. We did this by adding a set of 'wild-type' variables to the network, one for each original variable in the system, and included the wild-type variables in the update rules using an AND operation. For example, an update rule reading $A \leftarrow B$ OR C becomes $A \leftarrow (B$ OR $C)$ AND A^{WT}. We also adjusted the initial populations so that the non-wild-type genes always be-

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

gan OFF. The initial population contained each possible combination of knock-out mutations at a 1% mutation rate per variable *and* each possible combination of starting states compatible with each knockout set, spanning on the order of a trillion trillion different subpopulations, and then followed the time course of the TCR$^{\text{BOUND}}$ AND TCR$^{\text{P}}$ variable (Figure 1B). This result demonstrates several important aspects of our method. First, we are able to simulate highly heterogeneous populations. Second, we can exactly model subpopulations that were present at very low levels (see the error bars in Figure 1B). Third, our method is able to analyze genetically heterogeneous populations instead of populations that are only heterogeneous in their initial state. Finally, we note that our exact result tracked rare subpopulations over time more precisely than Monte Carlo simulations can. For example, the contribution of each triple-mutant was factored in even though a given triple-mutant was present in only 0.0001% of the population. While one might artificially raise the Monte Carlo mutation rate to oversample the mutations [8], this has the disadvantage of overweighting the effect of multiple mutants even though realistic evolutionary paths take one or very few mutational steps at a time [24]. In contrast, our exact result is dominated by the evolutionarily-accessible subpopulations that are closest to wild-type.

The code used to generate these results is named "tCellActivationEx.m", and is available for download at `https://github.com/heltilda/ProBaBool`. The equation-generating process for Figure 1A took ~ 0.4 seconds using our code (written in MATLAB R2015b 8.6.0.267246, running on a 2.6GHZ Intel core i7 Mac with OS 10.9.5). We checked this exact result against $n_{runs} = 10^4$ Monte Carlo runs, which took longer (~ 290 seconds). Note that Monte Carlo error is proportional to $1/\sqrt{n_{runs}}$.

Methods

Simulations of mixed populations

The principle behind our method is to write the time evolution of each variable in our network using a *linear equation*. Doing so guarantees that the dynamical equations we derive for a single cell also fully describe the dynamics of a mixed population of cells, owing to the superposition property of linear equations.

The key to writing linear equations is to introduce a new variable to represent each nonlinear term in a naïve update rule, acknowledging that we will have to solve for the dynamics of this new variable as well. In our case, each nonlinear term is a product of Boolean variables, so the update rule for its respective introduced variable will be a product of the constituent Boolean update rules. We demonstrate this procedure using Example 1.

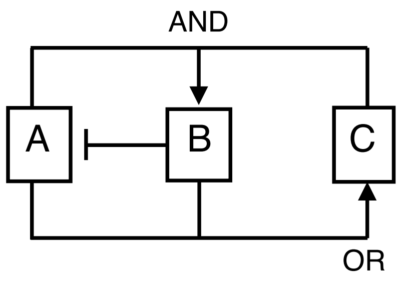

Figure 2. The 3-Boolean network used in Example 1. Arrows indicate how each variable updates based on the values of its inputs at the previous time step. For example if either A or B is ON at time t then C will be ON at time $t+1$; otherwise C will be OFF.

Example 1: a 3-variable network

Suppose we want to track the time evolution of variable A in the network shown in Figure 2. Since this network evolves by discrete time steps, we write $x_A(t+1) = f_A(x_B)$ where f_A performs the NOT operation. A linear equation implementing the NOT gate is:

$$f_A(x_B) = 1 - x_B. \tag{1.1}$$

Evidently, in order to follow x_A over all time we must also track the state of its input variable B over time. B implements an AND gate which is a *nonlinear* operation: $f_B = x_A \cdot x_C$. To make the equation linear, we introduce $x_{AC} = x_A \cdot x_C$, which is 1 if and only if both A and C are ON, and write f_B in terms of this new variable.

$$f_B = x_{AC} \tag{1.2}$$

We still need to calculate f_{AC} for our new variable x_{AC}, which is simply the product $f_A \cdot f_C$. (Proof: $f_{AC} = x_{AC}(t+1) = x_A(t+1) \cdot x_C(t+1) = f_A \cdot f_C$.) f_C implements an OR gate whose linear equation involves yet another product variable x_{AB}.

$$\begin{aligned} f_{AC} &= f_A \cdot f_C \\ &= (1 - x_B) \cdot (x_A + x_B - x_{AB}) \\ &= x_A - x_{AB} \end{aligned} \tag{1.3}$$

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

The formula for f_{AC} made use of the fact that any Boolean value squared equals itself: for example $x_B \cdot x_B = x_B$, and $x_B \cdot x_{AB} = x_B \cdot x_A \cdot x_B = x_{AB}$.

The process of replacing product terms with new variables, and then solving for the time evolution of those new variables, continues until the equations form a closed system: each variable's time evolution is in terms of other variables in our system.

$$\begin{aligned} f_{AB} &= (1 - x_B) \cdot (x_{AC}) \\ &= x_{AC} - x_{ABC} \qquad (1.4) \\ f_{ABC} &= (1 - x_B) \cdot (x_{AC}) \cdot (x_A + x_B - x_{AB}) \\ &= x_{AC} - x_{ABC} \qquad (1.5) \end{aligned}$$

This gives us a closed linear system. To avoid the constant term we can rewrite Eq. (1.1) as $f_A = x_\emptyset - x_B$, where $x_\emptyset = 1$ updates according to:

$$f_\emptyset = x_\emptyset. \qquad (1.6)$$

Equations 1.1-1.6 together with an initial state in $(x_A, x_B, x_{AC}, x_{AB}, x_{ABC})$ describe the time evolution of these quantities in a single Boolean network as a sequence of 0s and 1s in each variable. The final step is to reinterpret these equations as describing the dynamics of a mixed population of networks, by assigning to each variable the *fraction* of that population having that variable set to 1. So whereas for a single network the value of x_A should always be either 0 or 1, for a mixture of networks in which 40% of the population has gene A set ON we would set $x_A = 0.4$. Owing to the superposition property of linear systems, Equations 1.1-1.6 that were derived in the context of a single network also exactly model any mixed ensemble of these networks.

Any state or mixture of states can be written as a linear combination of product-basis variables $\{x\}$, because these variables form a complete basis spanning the state space (see Appendix 1 for a proof; also Ref. [18] proves a similar result for a slightly different Boolean algebra). Since each time-evolution function f is a sum over all states causing a '1' in the output variable when written in the state basis, it follows that each f is also a linear combination of our x variables. Therefore our procedure for modeling a mixed population always works in principle, even if some networks require too many equations for this method to be practical.

We can extend mixed-population modeling to probabilistic [21] and continuous-time [22] Boolean networks. Probabilistic Boolean networks require no changes in the algorithm; the only difference is that the polynomial coefficients in our equations may not be integers. For example, if the NOT gate in Fig. 2 is leaky with A turning ON with a probability of 0.9/0.2 if B was OFF/ON at the last time step, then the transition rule for gene A becomes $f_A = 0.9 - 0.7x_B$. As before, a linear equation can always be written because a) a linear equation can still be written in the state space basis, and b) our x variables are just a different basis covering the state space. Probabilistic networks give one way to incorporate rate information into our model; another way is to work in continuous time using differential equations: $f_A = dx_A/dt$. The differential form does require one change in our method: the rate of change of a higher-order variable is found by using the product rule of derivatives. Whereas under a discrete update $f_{ABC...}$ is the product $f_A \cdot f_B \cdot f_C \cdot \ldots$, for the differential case we compute:

$$\begin{aligned} f_{ABC...} &= \frac{d}{dt}(x_A x_B x_C \ldots) \\ &= x_A x_B \ldots f_C + x_A x_C \ldots f_B + \ldots . \qquad (1.7) \end{aligned}$$

Also, under discrete updates the trivial function is $f_\emptyset = 1$ but with differential updates it is $f_\emptyset = 0$.

Long-term behaviors

A mixed-population simulation may or may not be practical, depending on whether the system of linear equations closes with a manageable number of variables n. In the worst case, a significant fraction of the entire x-variable space is involved. By counting subscripts we know that there are $n = 2^N$ product variables associated with an N-Boolean network, which is expected because our x-variables are simply a change of basis from the state space (see Appendix 1). Therefore the problem has potentially exponential complexity.

One way to make progress even when a closed system of equations is unmanageable is to focus on the attractors (steady states or limit cycles). The attractors are governed by a linear space whose size is determined by the number of attractor states, which for biological networks is usually much smaller than the full equation space. Mathematically, this means that our linear equations form a very degenerate system: if there are only n^* steady states then there are only n^* non-zero eigenvalues and n^* linearly independent equations. So for a 50-node network with a single steady state attractor we might have $n = 2^{50} \approx 10^{15}$ in the worst case, but $n^* = 1$, which is a vastly smaller linear system. To find only the structure of the final-state space we select n^* linearly independent variables, substitute them for

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

the other variables in the time-evolution functions, and do an eigenvalue analysis on the much-smaller $n^* \times n^*$ system. A continuation of Example 1 gives a simple demonstration of this procedure.

Example 1, continued

Eqs. 1.1-1.6 in Example 1 contain a single linear dependency: $f_{AB} = f_{ABC}$. Therefore after the first time step x_{AB} will equal x_{ABC}: we write $x_{AB} \underset{t\geq 2}{=} x_{ABC}$. We use this fact to eliminate x_{AB}, giving a new set of steady state equations:

$$\begin{aligned} f_A &= 1 - x_B \\ f_B &= x_{AC} \\ f_{AC} &\underset{t\geq 2}{=} x_A - x_{ABC} \qquad (1.4') \\ f_{ABC} &= x_{AC} - x_{ABC} \\ f_\emptyset &= 1. \end{aligned}$$

Our new set of equations has the same non-zero eigenspace as the original set (1.1-1.6), except Eq. 1.4′ is only valid from the second time step onwards. However, the equations lack the null eigenspace because we removed the only linear dependency. States lying in the null eigenspace by definition decay and therefore correspond to transients in the time evolution, whereas eigenvectors whose eigenvalues have magnitude 1 do not decay and are part of the final attractor states. The 5 eigenvalues all have phases that are multiples of $2\pi/5$, indicating that the sole attractor is a limit cycle with a period of 5 time steps. The states are: (100) $\rightarrow$ (101) $\rightarrow$ (111) $\rightarrow$ (011) $\rightarrow$ (001) $\rightarrow \ldots$ at which point the sequence repeats.

There would be no time savings if we only eliminated a variable after we had already computed its time-evolution function. Fortunately, each linear dependency constrains not only the dependent variables, but also any variable that is *factorized* by a dependent variable (i.e. has all of the dependent variable's subscripts). Thus a dependency involving a low-order variable with few indices can exclude a significant fraction of the variable space. We find these constraints concurrently with the process of adding equations, and thereby avoid having to evaluate a significant fraction of our variable space.

Constraints are traditionally enforced by substitution; however, there are two problems with substituting constraint expressions in our case. First, there is no guarantee that when applying two constraints the second will not undo the work of the first, for example by reintroducing an index that the first eliminated. That is, there may be no self-consistent solution that uses all constraints. The second problem is that two dependencies might constrain overlapping indices on the same variable. For example, in the network of Figure 3, variable x_{ABC} is subject to constraints from both x_{AB} and x_{BC}, and substituting either constraint would eliminate the subscript '2' that the other constraint requires. We avoid both these problems by *multiplying* constraints rather than substituting them, using the fact that we can freely duplicate (or remove duplicates of) constrained indices, because a Boolean raised to any positive power equals itself. This process forces the removal of all variables containing certain indices and lacking others, of which there is always at least one.

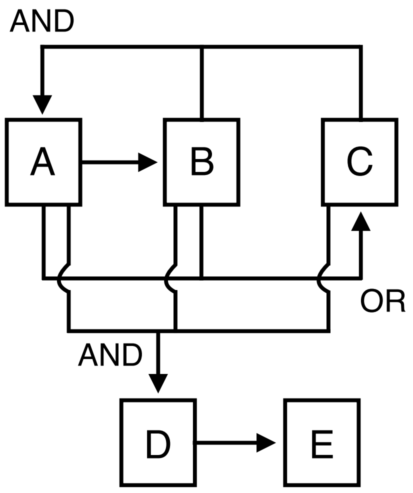

Figure 3. The network used in Example 2

Example 2

Suppose we want to find the long-time behavior of Boolean variables A, C and E in the network of Figure 3. After two iterations of solving for time-evolution functions we have:

$$\begin{aligned} f_A &= x_{BC} && (2.1) \\ f_C &= x_A + x_B - x_{AB} && (2.2) \\ f_E &= x_D && (2.3) \end{aligned}$$

$$\begin{aligned} f_B &= x_A && (2.4) \\ f_D &= x_{ABC} && (2.5) \\ f_{AB} &= x_{ABC} && (2.6) \\ f_{BC} &= x_A && (2.7) \end{aligned}$$

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

At this point there are two linear dependencies: $f_{AB} = f_D$ and $f_{BC} = f_B$, implying that

$$x_{AB} \underset{t\geq 2}{=} x_D \tag{C1}$$
$$x_D \underset{t\geq 2}{=} x_{AB} \tag{C2}$$
$$x_B \underset{t\geq 2}{=} x_{BC}. \tag{C3}$$

Since we are only interested in the long-time behavior, we will use these constraints to simplify our equations. For example if we were to retain x_{ABC} it would be affected by the relationships involving x_{AB}, x_B and x_{BC}, and it would not be possible to enforce all of these by substitution because there is only one B-index on x_{ABC}. But our method enforces constraints by *multiplying* them:

$$\begin{aligned} x_{ABC} &= x_{AB} \cdot x_B \cdot x_{BC} \cdot x_{ABC} \\ &\underset{t\geq 2}{=} x_D \cdot x_{BC} \cdot x_B \cdot x_{ABC} \\ &= x_{ABCD}. \end{aligned}$$

More generally, the first constraint attaches an AB index to every variable containing a D, and a D index to each variable with AB indices, and the second constraint adds a C index to every variable with a B index.

Constraining our system and eliminating disused variables gives us

$$\begin{aligned} f_A &= x_{BC} \\ f_C &\underset{t\geq 2}{=} x_A + x_{BC} - x_{ABCD} && (2.2') \\ f_E &\underset{t\geq 2}{=} x_{ABCD} && (2.3') \\ f_{BC} &= x_A. \end{aligned}$$

Our new equations require us to solve for another variable (while applying the constraints):

$$f_{ABCD} \underset{t\geq 2}{=} x_{ABCD}. \tag{2.8}$$

The system is now closed (5 equations involving 5 variables), so if our goal is to produce a simulation (valid from time step 2 onwards) then we are done. However, if our objective is to find the attractors then we must remove another dependency that was unmasked by the last equation: $f_C \underset{t\geq 2}{=} f_{BC} + f_A - f_E$, which implies three useful constraints.

$$x_A \underset{t\geq 3}{=} x_C + x_E - x_{BC} \tag{C4}$$
$$x_C \underset{t\geq 3}{=} x_A - x_E + x_{BC} \tag{C5}$$
$$x_E \underset{t\geq 3}{=} x_A - x_C + x_{BC} \tag{C6}$$

Each constraint reduces the size of the variable space: for example, the first eliminates all variables containing index A with no C, E or BC. We did not solve for x_{BC} because doing so would not eliminate any variables, because the x_C term does not attach any new indices.

After applying the new constraints we obtain:

$$\begin{aligned} f_{BC} &\underset{t\geq 3}{=} x_{AC} + x_{AE} - x_{ABCD} && (2.7') \\ f_{ABCD} &\underset{t\geq 2}{=} x_{ABCD} \\ f_{AC} &= x_{BC} && (2.9) \\ f_{AE} &\underset{t\geq 2}{=} x_{ABCD} && (2.10) \\ f_{CE} &\underset{t\geq 2}{=} x_{ABCD} && (2.11) \\ f_{BCE} &\underset{t\geq 2}{=} x_{ABCD} && (2.12) \end{aligned}$$

This produces new dependencies: $f_{AE} \underset{t\geq 2}{=} f_{ABCD}$ and $f_{CE} \underset{t\geq 2}{=} f_{ABCD}$, implying that

$$x_{AE} \underset{t\geq 3}{=} x_{ABCD} \tag{C7}$$
$$x_{ABCD} \underset{t\geq 3}{=} x_{AE} \tag{C8}$$
$$x_{CE} \underset{t\geq 3}{=} x_{ABCD}. \tag{C9}$$

With these constraints our example ends with the following equations:

$$\begin{aligned} f_{BC} &\underset{t\geq 3}{=} x_{AC} && (2.7'') \\ f_{ABCDE} &\underset{t\geq 3}{=} x_{ABCDE} && (2.13) \\ f_{AC} &= x_{BC}. \end{aligned}$$

The attractor is always reached at or before time step 3. The constraints map our original variables to linear combinations of variables in the final system: for example x_A is mapped by constraint C4 to $x_{AC} + x_{AE} - x_{ABC}$, then mapped using constraints C1, C7 and C8 to x_{AC}, whose dynamics are given by the final time-evolution equations. The eigenvalues of this final system are $(-1, 1, 1)$, implying a steady state along with a period-2 cycle.

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Each dependency produces at least one constraint that permanently eliminates at least one of the dependent variables, and usually many other variables as well. *Proof*: if the dependency contains only one term then that variable is zero, eliminating it and all variables it factorizes. If there is more than one term in the dependency then each lowest-index variable (which may be $x_\emptyset = 1$) accumulates at least one index from any other variable in the dependency. Therefore each lowest-index variable is always eliminated. *Corollary*: the calculation eventually terminates because there are a finite number of variables, and each new linear dependency removes at least one further variable.

Applying the equation-reduction method to each dependency in our linear system $F = \{f_i\}$ is guaranteed to eliminate the entire null space of F, simply because variables will be removed from the system until there are no more dependencies. On the other hand, our equation-reduction method does not affect the non-null eigenspace involving the variables of interest, because the constraints map those variables to variables in the final equations: $X_{final} = C \cdot X_{interest}$. Therefore the long-time behavior is accurately modeled by $F_{interest} = C^T F_{final} C$, which therefore contains all the persistent eigenmodes. All eigenvalues of F have modulus either 0 or 1 in a deterministic network, owing to the fact that a uniform population initialized with binary values for all variables stays binary for all time (this is obvious in the state-space basis, which by Appendix 1 has the same eigenspectrum as our product-basis).

Probabilistic and asynchronous Boolean networks

Our method supports modeling large populations of probabilistic Boolean networks (PBNs) [21, 25], in which several state transitions are possible at each time step, and the various transitions may have different probabilities. In the limit where the population of PBNs becomes infinite, each possible state transition occurs in a fraction of the population proportional to its likelihood in an individual. From the standpoint of our method, this implies that the coefficients in the f_i equations of a PBN become real-valued, but the process of building the f_i equations is unchanged.

The time-evolution equations F of a PBN in general contain eigenmodes having real-valued eigenvalues whose modulus is on the interval $[0, 1]$. (A modulus larger than 1 would represent a mode growing without bound, which is impossible because in the state-space basis the fraction of the population in each state b_i is restricted to the interval $[0, 1]$). Therefore, unlike a deterministic network, a PBN can have slowly-decaying modes with eigenvalues between 0 and 1. For PBNs we generalize our equation-reduction method to identify decaying modes before all f_i have been solved (i.e. before F is a square matrix), by identifying modes m having the property $m \cdot F = \lambda\,[m\ 0]$. We discard these modes after they have become sufficiently small, defined by $e^{-\lambda(t - \max_m(t_{f_m}))} < \epsilon$ where t_{f_m} are the 'starting times' of the involved equations and ϵ is a user-defined threshold.

Large populations of asynchronous networks behave identically to large populations of PBNs [26] if we define a uniform time step: the likelihoods of the various possible updates give the state-transition weights in the corresponding synchronous PBN. Therefore our analysis also applies to large populations of asynchronous networks. The conversion of an asynchronous updating scheme to a PBN lowers some of the eigenvalues that would otherwise be of magnitude 1 in a synchronous version of the network, and therefore our equation-reduction method can prune asynchronous networks more aggressively than their synchronous counterparts.

Calculational notes When we attempt to simplify a network by removing dependencies, the order in which we calculate the time evolution of new variables and look for new dependencies greatly influences the total number of variables that will be involved before the linear system closes. That is because the constraints coming from different dependencies simplify the system to different degrees. In general, constraints on variables having the fewest indices are most helpful, because they factorize the largest part of the variable space. Following this rule of thumb, our implementation solves only the fewest-index variables between each search for new dependencies. Additionally, we add low-index factors of new variables even if they were not directly involved in earlier equations. In our tests, this prioritization method greatly speeds up the calculation.

Certain constraints can multiply the prefactors of some variables in such a way that the prefactors can exceed the integer limit if enough of these constraints are applied. For example, the constraint $x_1 = x_2 + x_3$ takes $x_{123} \to 2x_{123}$. Typically the variable being multiplied would be found to be zero later in the calculation, so having a coefficient of 2 is not an error in the math, but it can make calculations impractical if the doubling happens repeatedly and the coefficient becomes very large. Our solution is to require the right-hand side R of the constraint equation to be binary when the left-hand side L is binary, implying that $R^2 - R = 0$ (see Appendix 2). This 'corollary constraint' slows the calculation so we only use it with small constraint equations or equations containing many subscripts.

To avoid rounding errors we perform most calculations using rational arithmetic, where each number a/b is represented using a pair of integers (a, b). However the singular values calculations (used to find fast-decaying modes) require real arithmetic, and we currently require the use of

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

a rational approximation function to convert these real-valued singular values into fractions.

Our product basis has the property that any product of variables is itself a variable in the basis, so that polynomials need never contain products of these variables. In the product basis, multiplication is interpreted as a *union* operation on the indices of the variables being multiplied. To place these rules in a mathematical context, this algebra is a commutative ring with an idempotent multiplication ($x_i^2 = x_i$).

Discussion

Our product-basis method allows the simulation of highly heterogeneous populations, including the transient processes that are generally ignored by analytic methods, as well as the steady states and limit cycles. This approach can be used to follow single variables of the system over time, as well as the correlations between these variables that are both necessary and sufficient to fully describe the dynamics of the population. It can account for the affect of mutations as well as variability in network state throughout the population, and can exactly model very rare occurrences or subpopulations. Our approach can be applied to simulate large populations of practically any sort of Boolean networks. An extension of the method allows the set of equations to be simplified while still capturing the long-term behavior.

The advance in our method is to write the time-evolution equations as a linear system, but in a different basis than the usual state space basis. Our variables have several advantages. First, descriptors of a mixed population naturally use words that correspond more closely to our variables than to individual states. For example, we might specify that half the population starts with both genes A and B on, which implies that $x_{AB} = 0.5$ but is agnostic about the state of other variables. Another advantage is that our equations often close using relatively few of our product variables for any mixed population, whereas the number of equations required in the state space basis scales with the heterogeneity of the population: the simulations we showed in Figure 1 would require all 2^{40} state space variables. Thus our choice of variables is superior for modeling very heterogeneous populations. Finally, our basis allows some variables to factorize others, allowing us to vastly simplify the calculation in many cases where we only care about the long-term behavior.

We acknowledge that our method can become intractable for complicated networks due to the fact that the construction of these simulations is potentially an exponential problem. A full simulation can require up to 2^N equations to model, and even the attractor analysis is known to be NP-hard [27]. Large size, complicated logic rules and certain types of feedback loop in particular seem to make the analysis difficult. These are fundamental limitations. However, the attractor analysis depends on an equation-reduction scheme that is somewhat of an art, and we anticipate that future work will greatly improve this part of the calculation for typical network models.

Our method can be applied to any system involving heterogeneous populations, as long as the individuals in a population can be modeled using Boolean logic. Heterogeneity plays a major role in such varied systems as healthy and cancerous tissues, evolution at the organism scale, and the social dynamics of unique individuals [28]. In all of these cases, rare and unexpected dynamics are difficult to capture by simulations of individuals, while pure attractor analyses may miss important aspects of the dynamics. We believe that the methodology outlined here can help to capture these important but rare events.

Appendices

Appendix 1: correlation variables form a complete and independent basis

For N Boolean variables there are 2^N variables in the state space basis (b variables): this is just the number of states of the system. Likewise there are 2^N variables in the product space basis (x variables) because there are 2^N combinations of subscripts on these variables: each of N subscripts may be present or absent. Therefore the two spaces have the same number of variables, but this does not prove that the product-basis spans the entire Boolean space. In order to show that the x-variables form a complete basis in b-space, we imagine explicitly writing the transformation matrix from b-space to x-space. For example, for the case of three Boolean variables this matrix is:

$$T_{xb} = \begin{array}{c|cccccccc} & b_\emptyset & b_A & b_B & b_{AB} & b_C & b_{AC} & b_{BC} & b_{ABC} \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_A & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ x_B & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ x_{AB} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ x_C & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ x_{AC} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ x_{BC} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ x_{ABC} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

The transformation matrix is upper-triangular, with ones on the diagonal. The reason is that each x-variable is turned on by the b-variable having the same indices

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

(hence the ones along the diagonal), and by any b-variables containing additional indices implying a higher position in the matrix (since the indices are arranged in binary order of their subscripts); however an x-variable is never turned on by a b-variable that is *missing* one of its indices, which is why the lower triangular block is empty. Since the matrix is triangular the eigenvalues are the ones on the diagonal, so the determinant is one, the transformation is non-degenerate (and volume-preserving) and therefore the x-space spans the b-space.

Appendix 2: the corollary constraint

Our starting equations are guaranteed to give $f_i = (0, 1)$ for any possible state where all of the x_i are also 0 or 1, simply because the equations must work for every state of a single Boolean network. However, once we have applied constraint equations this is no longer true: certain combinations of the x_i forbidden by the constraint can lead to non-Boolean values for certain f_i. This can give us additional constraints.

Each constraint begins with a degeneracy of the form

$$c_a x_a + c_b x_b + \ldots + c_n x_n = 0$$

where x_a, x_b, etc. each are variables having different combinations of indices (for example, x_a might be x_{BFJL}). From this degeneracy we get up to n constraints of the form

$$x_i = -\sum_{j \neq i} \frac{c_j}{c_i} x_j.$$

Any instance of a variable x_i, or any variable $x_{\ldots i \ldots}$ of which x_i is a factor, will be multiplied by this polynomial to enforce the constraint.

It turns out that each constraint may imply a set of 'corollary' constraints that must hold in order for the constraint variable x_i to be Boolean. For example, if the constraint is $x_i = x_1 + x_2$ then our corollary is that $x_{12} = 0$; otherwise x_i could become 2, which is impossible. To get the corollary constraints we require that each constraint variable x_i obey $x_i^2 - x_i = 0$. Applied to the right-hand side of the constraint equation, our constraint implies that

$$\begin{aligned} 0 &= \sum_{j \neq i} \left(\frac{c_j^2}{c_i^2} x_j^2 + \frac{c_j}{c_i} x_j \right) + \sum_{\{j,k\} \neq i, j \neq k} \frac{c_j c_k}{c_i^2} x_{jk} \\ &= \sum_{j \neq i} \frac{c_j (c_i + c_j)}{c_i^2} x_j + \sum_{\{j,k\} \neq i, j \neq k} \frac{c_j c_k}{c_i^2} x_{jk} \end{aligned}$$

(Here x_{jk} is the variable having the subscripts of both x_j and x_k: in other words $x_{jk} = x_j \cdot x_k$.) To simplify, we note that our original degeneracy also implies that

$$\begin{aligned} x_j &= -\sum_{k \neq j} \frac{c_k}{c_j} x_k \\ &= -\frac{c_i}{c_j} x_i - \sum_{k \neq i, k \neq j} \frac{c_k}{c_j} x_k \end{aligned}$$

which we use to constrain x_j. Multiplying by c_i:

$$\begin{aligned} 0 = &-\sum_{j \neq i} \frac{c_i (c_i + c_j)}{c_i} x_{ij} - \sum_{\{j,k\} \neq i, j \neq k} \frac{c_k (c_i + c_j)}{c_i} x_{jk} \\ &+ \sum_{\{j,k\} \neq i, j \neq k} \frac{c_j c_k}{c_i} x_{ijk} \\ &= \sum_{j \neq i} (c_i + c_j) x_{ij} + \sum_{\{j,k\} \neq i, j \neq k} c_k x_{jk} \\ &= \sum_{j \neq i} (c_i + c_j) x_{ij} + \sum_{\{j,k\} \neq i, j < k} (c_j + c_k) x_{jk} \end{aligned}$$

where in the last step the double-summation ranges only over unique pairs of indices j and k.

Appendix 3: algorithm and code

Pseudocode is given in Algorithm 1. The full code is available at: `https://github.com/heltilda/ProBaBool`.

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Algorithm 1 build closed system of equations F

```
1: Initialize set of unsolved variables with variables of interest: $X \leftarrow \{x_1, x_2, ..., x_n\}$
2: Initialize set of update rules for variables $X$: $F \leftarrow \emptyset$
3: Initialize set of constraints: $C \leftarrow \emptyset$
4: Initialize simulation start time: $t_{sim} \leftarrow 1$
5: Initialize set of equation start times of $F$: $T_F \leftarrow \emptyset$
6: Initialize set of constraint start times: $T_C \leftarrow \emptyset$

7: while $X$ is not empty do

8:     % Reduce equations if necessary
9:     if $size(F) > equation\_reduction\_threshold$ then
10:        $F_{t<t_{sim}} \leftarrow \{f_i \in F \text{ s.t. } t_{f_i} < t_{sim}\}$
11:        $D \leftarrow$ set of linear-dependencies within $F_{t<t_{sim}}$ using SVD/QR factorizing
12:        if no linear dependencies found and $size(F_{t<t_{sim}}) < size(F)$ then
13:            $D \leftarrow$ set of linear-dependencies within $F$ using SVD/QR factorizing
14:            if linear dependencies found then
15:                $t_{sim} \leftarrow t_{sim} + 1$
16:            end if
17:        end if
18:        for each $d_i \in D$ do
19:            NewConstraints$(d_i, t_{sim})$
20:        end for
21:        Sort $C$ by number of indices
22:        $F \leftarrow$ Constrain$(F, 1)$
23:        $X \leftarrow X \cup$ lowest-index factors of $X$
24:        $F \leftarrow$ Constrain$(F, 1)$
25:        Sort $X$ by number of indices
26:    end if

27:    % Add new equations
28:    for each $x_i \in X$ do
29:        $f_i \leftarrow 1$
30:        for each Boolean factor $x_b$ of $x_i$ do
31:            $f_i \leftarrow f_i \cdot f_b$
32:        end for
33:        $F \leftarrow F \cup f_i$
34:        $F \leftarrow$ Constrain$(F, 1)$
35:    end for

36: end while
```

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

```
37: function NewConstraints($(d, t_c)$)
38:     for each $x_j \in d$ do
39:         $RHS_j \leftarrow$ solve $(d = 0)$ for $x_j$
40:         if $\{x_j\} \neq$ Constrain$(\{RHS_j\}, t_c)[1]$ then
41:             NewConstraints$(RHS_j^2 - RHS_j, t_c)$
42:             for each $x_k =$ multiples of $x_j$ do
43:                 $X_{new} \leftarrow$ new terms in $x_k \cdot RHS_k$
44:                 $X \leftarrow X \cup X_{new}$
45:                 $A \leftarrow \{a_l =$ coefficients of $x_k$ in $F\}$
46:                 $F \leftarrow F + A \cdot (RHS_k - x_k)$
47:                 $(t_{f_l}$ for all $l$ such that $a_l \neq 0) \leftarrow t_c$
48:             end for
49:             for each prior constraint $(x_p = RHS_p)$ do
50:                 if $x_p$ is factorized by $x_j$ and $x_p \cdot RHS_j = RHS_p$ then
51:                     $C \leftarrow C - \{c_p\}$
52:                 end if
53:             end for
54:             $C \leftarrow C \cup \{x_j = RHS_j\}$
55:         end if
56:     end for
57: end function

58: function Constrain($(Polys, t_c)$)
59:     for each $poly_i \in Polys$ do
60:         for each unconstrained variable $x_j$ with coefficient $a_j$ in $poly_i$ do
61:             for each constraint $x_p = RHS_p$ do
62:                 if $x_j$ is factorized by $x_p$ then
63:                     $poly_{new} \leftarrow x_j \cdot RHS_p$
64:                     if $x_j \not\subset poly_{new}$ then
65:                         $poly_i \leftarrow (poly_i - a_j \cdot x_j) \cup poly_{new}$
66:                         $t_{f_i} \leftarrow t_c$
67:                     end if
68:                 end if
69:             end for
70:         end for
71:     end for
72:     return $Polys$
73: end function
```

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Acknowledgments

This work is supported by the Boettcher Foundation (J.C.) and NIH grant 2T15LM009451 (B.R.).

References

[1] István Albert et al. "Boolean network simulations for life scientists". *Source code for biology and medicine* 3.1 (2008), p. 16.

[2] David J Earl and Michael W Deem. "Monte Carlo simulations". *Molecular modeling of proteins* (2008), pp. 25–36.

[3] Jonathan R Karr et al. "A whole-cell computational model predicts phenotype from genotype". *Cell* 150.2 (2012), pp. 389–401.

[4] Akshata R Udyavar et al. "Novel hybrid phenotype revealed in Small Cell Lung Cancer by a transcription factor network model that can explain tumor heterogeneity". *Cancer Research* (2016), canres–1467.

[5] Li Ding et al. "Clonal evolution in relapsed acute myeloid leukaemia revealed by whole-genome sequencing". *Nature* 481.7382 (2012), pp. 506–510.

[6] Ash A Alizadeh et al. "Toward understanding and exploiting tumor heterogeneity". *Nature medicine* 21.8 (2015), pp. 846–853.

[7] Brett E Johnson et al. "Mutational analysis reveals the origin and therapy-driven evolution of recurrent glioma". *Science* 343.6167 (2014), pp. 189–193.

[8] Jun S Liu. *Monte Carlo strategies in scientific computing.* Springer Science & Business Media, 2008.

[9] Jorge GT Zañudo and Réka Albert. "An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks". *Chaos: An Interdisciplinary Journal of Nonlinear Science* 23.2 (2013), p. 025111.

[10] Alan Veliz-Cuba et al. "Steady state analysis of Boolean molecular network models via model reduction and computational algebra". *BMC bioinformatics* 15.1 (2014), p. 1.

[11] Aurélien Naldi et al. "A reduction of logical regulatory graphs preserving essential dynamical properties". *International Conference on Computational Methods in Systems Biology.* Springer. 2009, pp. 266–280.

[12] Stefan Bornholdt. "Boolean network models of cellular regulation: prospects and limitations". *Journal of the Royal Society Interface* 5.Suppl 1 (2008), S85–S94.

[13] Takeyuki Tamura and Tatsuya Akutsu. "Detecting a singleton attractor in a Boolean network utilizing SAT algorithms". *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 92.2 (2009), pp. 493–501.

[14] Elena Dubrova and Maxim Teslenko. "A SAT-based algorithm for finding attractors in synchronous boolean networks". *IEEE/ACM transactions on computational biology and bioinformatics* 8.5 (2011), pp. 1393–1399.

[15] Desheng Zheng et al. "An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks". *PloS one* 8.4 (2013), e60593.

[16] Abhishek Garg et al. "Synchronous versus asynchronous modeling of gene regulatory networks". *Bioinformatics* 24.17 (2008), pp. 1917–1925.

[17] Alan Veliz-Cuba. "Reduction of Boolean network models". *Journal of theoretical biology* 289 (2011), pp. 167–172.

[18] Alan Veliz-Cuba, Abdul Salam Jarrah, and Reinhard Laubenbacher. "Polynomial algebra of discrete models in systems biology". *Bioinformatics* 26.13 (2010), pp. 1637–1643.

[19] Vincent Devloo, Pierre Hansen, and Martine Labbé. "Identification of all steady states in large networks by logical analysis". *Bulletin of mathematical biology* 65.6 (2003), pp. 1025–1051.

[20] Aurélien Naldi, Denis Thieffry, and Claudine Chaouiya. "Decision diagrams for the representation and analysis of logical models of genetic networks". *International Conference on Computational Methods in Systems Biology.* Springer. 2007, pp. 233–247.

[21] Ilya Shmulevich et al. "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks". *Bioinformatics* 18.2 (2002), pp. 261–274.

[22] Roberto Serra, Marco Villani, and Anna Salvemini. "Continuous genetic networks". *Parallel computing* 27.5 (2001), pp. 663–683.

[23] Steffen Klamt et al. "A methodology for the structural and functional analysis of signaling and regulatory networks". *BMC bioinformatics* 7.1 (2006), p. 56.

[24] Michael Lynch. "Rate, molecular spectrum, and consequences of human mutation". *Proceedings of the National Academy of Sciences* 107.3 (2010), pp. 961–968.

bioRxiv preprint doi: https://doi.org/10.1101/110817; this version posted February 22, 2017. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

[25] Panuwat Trairatphisan et al. “Recent development and biomedical applications of probabilistic Boolean networks”. *Cell communication and signaling* 11.1 (2013), p. 46.

[26] Ilya Shmulevich and John D Aitchison. “Deterministic and stochastic models of genetic regulatory networks”. *Methods in enzymology* 467 (2009), pp. 335–356.

[27] Qianchuan Zhao. “A remark on “Scalar equations for synchronous Boolean networks with biological Applications” by C. Farrow, J. Heidel, J. Maloney, and J. Rogers”. *IEEE Transactions on Neural Networks* 16.6 (2005), pp. 1715–1716.

[28] Jurek Kolasa and C David Rollo. “Introduction: the heterogeneity of heterogeneity: a glossary”. *Ecological heterogeneity*. Springer, 1991, pp. 1–23.