Computational Aspects of Aggregation in Biological Systems

Vladik Kreinovich and Max Shpak

University of Texas at El Paso, El Paso, TX 79968, USA vladik@utep.edu, mshpak@utep.edu

Summary. Many biologically relevant dynamical systems are aggregable, in the sense that one can divide their variables $x_1, \ldots, x_n$ into several (k) non-intersecting groups and find functions $y_1, \ldots, y_k$ $(k < n)$ from these groups (macrovariables) whose dynamics only depend on the initial state of the macrovariable. For example, the state of a population genetic system can be described by listing the frequencies x_i of different genotypes, so that the corresponding dynamical system describe the effects of mutation, recombination, and natural selection. The goal of aggregation approaches in population genetics is to find macrovariables $y_a, \ldots, y_k$ to which aggregated mutation, recombination, and selection functions could be applied. Population genetic models are formally equivalent to genetic algorithms, and are therefore of wide interest in the computational sciences.

Another example of a multi-variable biological system of interest arises in ecology. Ecosystems contain many interacting species, and because of the complexity of multi-variable nonlinear systems, it would be of value to derive a formal description that reduces the number of variables to some macrostates that are weighted sums of the densities of individual species.

In this chapter, we explore different computational aspects of aggregability for linear and non-linear systems. Specifically, we investigate the problem of conditional aggregability (i.e., aggregability restricted to modular states) and aggregation of variables in biologically relevant quadratic dynamical systems.

1 Introduction

1.1 What is Aggregability

Dynamical systems: informal introduction. Many systems in nature can be described as *dynamical systems*, in which the state of a system at each moment of time is characterized by the values of (finitely many) variables $x_1, \ldots, x_n$, and the change of the state over time is described by an equation $x'_i = f_i(x_1, \ldots, x_n)$, where

- for continuous-time systems, in which the time t can take any real value, x'_i is the first time derivative of x_i:

$$\frac{dx_i}{dt} = f_i(x_1(t), \ldots, x_n(t)); \qquad (1)$$

- for discrete-time systems, in which the time t can only take integer values, x_i' is the change in the value of x_i between the given moment t and the next moment of time:

$$x_i(t+1) - x_i(t) = f_i(x_1(t), \ldots, x_n(t)). \qquad (2)$$

In the discrete-time case, we can also describe this dynamics as

$$x_i(t+1) = \widetilde{f}_i(x_1(t), \ldots, x_n(t)), \qquad (3)$$

where $\widetilde{f}_i(x_1, \ldots, x_n) \stackrel{\text{def}}{=} x_i + f_i(x_1, \ldots, x_n)$.

For example, the state of a biological population can be described by listing the amounts or relative frequencies x_i of different genotypes i; in this example, the corresponding functions $f_i(x_1, \ldots, x_n)$ describe the effects of mutation, recombination, and natural selection.

Dynamical systems: formal definitions. Let us describe the above ideas in precise terms.

Definition 1. *Let n be an integer. This integer will be called the* number of microvariables *(or* variables*, for short). These variables will be denoted by $x_1, \ldots, x_n$. By a* microstate *(or* state*), we mean an n-dimensional vector $x = (x_1, \ldots, x_n)$.*

Definition 2.

- *By a* discrete-time trajectory*, we means a function which maps integers t into states $x(t)$.*
- *By a* continuous-time trajectory*, we means a function which maps real numbers t into states $x(t)$.*

For each trajectory and for each moment of time t, the state $x(t)$ is called a state at moment *t.*

Definition 3. *For a given n, by a* dynamical system*, we mean a tuple $(n, f_1, \ldots, f_n)$, where $n \geq 1$ is an integer, and $f_1, \ldots, f_n : \mathbb{R}^n \to \mathbb{R}$ are functions of n variables.*

- *We say that a discrete-time trajectory $x(t)$ is* consistent *with the dynamical system $(n, f_1, \ldots, f_n)$ if for every t, we have $x_i(t+1) - x_i(t) = f_i(x_1(t), \ldots, x_n(t))$.*
- *We say that a continuous-time trajectory $x(t)$ is* consistent *with the dynamical system $(n, f_1, \ldots, f_n)$ if for every t, we have $\dfrac{dx_i(t)}{dt} = f_i(x_1(t), \ldots, x_n(t))$.*

Equilibria. In general, when we start in some state $x(t)$ at the beginning moment of time t, the above dynamics leads to a different state $x(t+1)$ at the next moment of time. In many practical situations, these changes eventually subside, and we end up with a state which does not change with time, i.e., with an *equilibrium* state. In the equilibrium state x, we have $x_i'(t) = \frac{dx_i}{dt} = 0$ or $x_i'(t) = x_i(t+1) - x_i(t) = 0$, i.e., in general, $x_i'(t) = f_i(x_1, \ldots, x_n) = 0$.

Need for aggregation. For natural systems, the number of variables is often very large. For example, for a system with g loci on a chromosome in which each of these genes can have two possible allelic states, there are $n = 2^g$ possible genotypes. For large g, due to the large number of state variables, the corresponding dynamics is extremely difficult to analyze.

This complexity of this analysis can often be reduced if we take into consideration that in practice, quantities corresponding to different variables x_i can be grouped into natural clusters. This happens, for example, when interactions within each cluster are much stronger than interactions across different clusters. In mathematical terms, this means that we subdivide the variables $x_1, \ldots, x_n$ into non-overlapping blocks $I_1 = \{i(1,1), \ldots, i(a,n_1)\}, \ldots, I_k = \{i(k,1), \ldots, i(k,n_k)\}$ $(k \ll n)$.

To describe each cluster I_a, it is often not necessary to know the value of each of its "microvariables" $x_{i(a,1)}, \ldots, x_{i(a,n_a)}$. It is sufficient to characterize this state by a single "macrovariable" $y_a = c_a(x_{i(a,1)}, \ldots, x_{i(a,n_a)})$ in such a way that the dynamics of these macrovariables is determined only by their previous values. In this case, equations (1) or (2) lead to simpler equations

$$\frac{dy_a}{dt} = h_a(y_1(t), \ldots, y_k(t)) \tag{4}$$

or, correspondingly,

$$y_a(t+1) - y_a(t) = h_i(y_1(t), \ldots, y_k(t)) \tag{5}$$

for appropriate functions $h_1, \ldots, h_k$. Dynamical systems with this property are called *decomposably aggregable.* Many biological systems (and in many systems from other fields such as economics [24] and queuing theory [3] etc.) are *decomposably aggregable* in this sense.

The aggregability property has been actively studied; see, e.g., [3, 4, 5, 13, 18, 19, 22, 23, 24].

Aggregability: formal definition. Let us describe a formalization of the above notions. We would like to avoid a degenerate case like $c_a = 0$, and make sure that at least one of the macrovariables depends on some microvariable x_{i_0}. To describe this degeneracy condition, let us fix a microvariable x_{i_0} (i.e., an index from 1 to n). In a partition, this microvariable can belong to one of the blocks. Without losing generality, let us assume that it belongs to the first block I_1 (if it belong to another block, we can simply rename the blocks).

Since i_0 belongs to the first block and the blocks do not overlap, only the first macrovariable y_1 can depend on this microvariable. We would also like to avoid a degenerate case in which the macrovariables do not depend on this microvariable x_{i_0} at all, so we require that y_1 actually depend on x_{i_0}. Let us describe this requirement in precise terms.

Definition 4. *Let us fix an index $i_0 \le n$. By a* partition, *we mean a tuple $(i_0, I_1, \ldots, I_k)$ $(k < n)$ where $I_1 \subseteq \{1, \ldots, n\}$, …, $I_k \subseteq \{1, 2, \ldots, n\}$ are non-empty sets such that $i_0 \in I_1$, $I_1 \cup \ldots \cup I_k = \{1, \ldots, n\}$, and $I_i \cap I_j = \emptyset$ for all $i \ne j$. For each partition, the number of elements in the set I_a will be denoted by n_a, and these elements will be denoted by $i(a, 1), \ldots, i(a, n_a)$.*

Definition 5. *We say that a function $c : \mathbb{R}^m \to \mathbb{R}$* actually depends *on the variable x_{i_0} if there exist real numbers $x_1, \ldots, x_{i_0-1}, x_{i_0}, x_{i_0+1}, \ldots, x_m$ and a real number $x'_{i_0} \ne x_{i_0}$ for which*

$$c(x_1, \ldots, x_{i_0-1}, x_{i_0}, x_{i_0+1}, \ldots, x_m) \ne c(x_1, \ldots, x_{i_0-1}, x'_{i_0}, x_{i_0+1}, \ldots, x_m).$$

Definition 6.

- *By a* decomposable aggregation, *we mean a tuple $(i_0, I_1, \ldots, I_k, c_1, \ldots, c_k)$, where $(i_0, I_1, \ldots, I_k)$ is a partition, and for each a from 1 to k, $c_a : \mathbb{R}^{n_a} \to \mathbb{R}$ is a function of n_a variables such that the function c_1 actually depends on x_{i_0}.*
- *For every microstate $x = (x_1, \ldots, x_n)$, by the corresponding* macrostate *we mean a tuple $y = (y_1, \ldots, y_k)$, where $y_a = c_a(x_{i(a,1)}, \ldots, x_{i(a,n_a)})$.*
- *We say that two microstates x and $\widetilde{x}$ are* macroequivalent *if they lead to the same macrostate $y = \widetilde{y}$.*

Definition 7. *We say that a decomposable aggregation $(I_1, \ldots, I_k, c_1, \ldots, c_k)$ is* consistent *with the dynamical system $(n, f_1, \ldots, f_n)$ if for every two trajectories x and $\widetilde{x}$ for which at some moment of time t, two microstates $x(t)$ and $\widetilde{x}(t)$ are macroequivalent, they remain macroequivalent for all following moments of time $t' > t$.*

Definition 8. *Let $k > 0$ be a positive integer.*

- *We say that a dynamical system is* decomposably k-aggregable *if it is consistent with some decomposable aggregation $(i_0, I_1, \ldots, I_k, c_1, \ldots, c_k)$.*
- *We say that a dynamical system is* decomposably $\le k$-aggregable *if it is decomposably ℓ-aggregable for some $\ell \le k$.*

Definition 9. *We say that a dynamical system is* decomposably aggregable *if it is decomposably k-aggregable for some integer k.*

1.2 Discussion

We can have intersecting blocks. Some practical systems have a similar aggregability property, but with possibly overlapping blocks I_a. In the general case, we have macrovariables $y_a = c_a(x_1, \ldots, x_n)$ each of which may depend on all the microvariables $x_1, \ldots, x_n$. We are still interested in the situation when the dynamics of the macrovariables is determined only by their previous values.

In some cases, such overlapping decomposabilities are useful; however, in general, they are not practical. For example, for every continuous-time dynamical system, we can define a macrovariable $y_1(x_1, \ldots, x_n)$ as the time t after (or before) which the trajectory starting at a state $(x_1, \ldots, x_n)$ reaches the plane $x_1 = c$ for some constant c (this y_1 is defined at least for values $x_1 \approx a$). The dynamics of the new macrovariable is simple: if in a state x, we reach $x_1 = c$ after time $t = y_1(x)$, then for a state x' which is t_0 seconds later on the same trajectory, the time to reaching $x_1 = c$ is $t - t_0$. In other words, the value of y_1 decreases with time t as $y_1(t_0) = y_1(0) - t_0$, or, in terms of the corresponding differential equation, $\dfrac{dy_1}{dt} = -1$.

From the purely mathematical viewpoint, we have an (overlapping) aggregation. However, the main objective of aggregation is to *simplify* solving the system of equations. In the above example, to find $y_1(x)$ for a given x, we, in effect, first need to solve the system – which defeats the purpose of aggregation.

In view of this observation, and taking into account that most practical aggregable systems are *decomposable* (i.e., the blocks do not intersect), in this chapter, we will concentrate on decomposable aggregations. For readability, unless otherwise indicated, we will simply refer to decomposable and aggregable systems as *aggregable*.

We can have strong interactions between clusters. In our motivations, we assumed that the interaction within each cluster is much stronger than the interaction among clusters. While this was indeed the original motivating example, the aggregability property sometimes occurs even when the interaction between clusters is strong – as long it can be appropriately "decomposed". In view of this fact, in the following precise definitions, we do not make any assumptions about the relative strengths of different interactions.

Approximate aggregability. It is worth mentioning that perfect aggregability usually occurs only in idealized mathematical models. In many practical situations, we only have *approximate* aggregability, so that the aggregate dynamics (4) or (5) differs only marginally from the actual microdynamics of the macrovariables variables $y_a = h_a(x_{i(a,1)}, \ldots, x_{i(a,n_a)})$.

Note that many dynamical systems are only approximately aggregable during certain time intervals in their evolution, or over certain subspaces of their state space [5, 24].

1.3 Linear Systems

Informal introduction. In principle, the functions $f_i(x_1, \ldots, x_n)$ can be arbitrarily complex. In practice, we can often simplify the resulting expressions if we expand each function $f_i(x_1, \ldots, x_n)$ in Taylor series in x_i and keep only terms up to a fixed order in this expansion. In particular, when the interactions are weak, we can often only keep the linear terms, i.e., get linear systems

$$x_i'(t) = a_i(t) + \sum_{j=1}^{n} c_{i,j} \cdot x_j(t). \tag{6}$$

In many practical cases, the i-th variable describes the absolute amount of i-th entity (such as i-th genotype). In this case, if we do not have any entities at some moment t, i.e., if we have $x_i(t) = 0$ for all i, then none will appear. So, we will have $x_i'(t) = 0$, and thus, $a_i(t) = 0$. In such cases, the above linear system takes an even simpler form

$$x_i'(t) = \sum_{j=1}^{n} c_{i,j} \cdot x_j(t). \tag{7}$$

Linear dynamical systems: formal definitions. Let us describe how the general definitions of dynamical systems look like in the linear case.

Definition 10. *We say that a dynamical system $(n, f_1, \ldots, f_n)$ is* linear *if all the functions f_i are linear, i.e., if $f_i = \sum\limits_{j=1}^{n} c_{i,j} \cdot x_j$ for some values $c_{i,j}$.*

This definition can be described in the following equivalent form.

Definition 11. *For a given n, by a* linear dynamical system, *we mean an $n \times n$ matrix c with entries $c_{i,j}$, $1 \le i \le n$, $1 \le j \le n$.*

- *We say that a discrete-time trajectory $x(t)$ is* consistent *with the linear dynamical system (n, c) if for every t, we have*

$$x_i(t+1) - x_i(t) = \sum_{j=1}^{n} c_{i,j} \cdot x_j(t).$$

- *We say that a continuous-time trajectory $x(t)$ is* consistent *with the linear dynamical system (n, c) if for every t, we have $\dfrac{dx_i(t)}{dt} = \sum\limits_{j=1}^{n} c_{i,j} \cdot x_j(t)$.*

Comment. In reality, the coefficients $c_{i,j}$ can be arbitrary real numbers. However, our main objective is to analyze the corresponding algorithms. So, instead of the actual (unknown) value of each coefficient, we can only consider the (approximate) value represented in the computer, which are usually rational numbers. In view of this fact, in the computational analysis of problems related to linear dynamical systems, we will always assume that all the values $c_{i,j}$ are rational numbers.

Equilibria. In particular, for such linear systems, equilibrium states $x = (x_1, \ldots, x_n)$ are states which satisfy the corresponding (homogeneous) system of linear equations

$$\sum_{j=1}^{n} c_{i,j} \cdot x_j = 0. \tag{8}$$

Of course, the state $x = (0, \ldots, 0)$ is always an equilibrium for such systems. In some physical systems, this trivial 0 state is the only equilibrium. However, in biology, there usually exist non-zero equilibrium states. In such cases, the matrix $c_{i,j}$ is singular.

In general, the set of all possible solutions of a homogeneous linear system is a linear space – in the sense that a linear combination of arbitrary solutions is also a solution. In every linear space, we can select a *basis*, i.e., a set of linearly independent vectors such that every other solution is a linear combination of solutions from this basis. The number of these independent vectors is called a *dimension* of the linear space. In principle, we can have matrices for which this linear space has an arbitrary dimension $\le n$. However, for almost all singular matrices, this dimension is equal to 1.

In view of this fact, it is reasonable to consider only such matrices in our analysis of biological systems. For such systems, all equilibria states $x = (x_1, \ldots, x_n)$ are proportional to some fixed state $\beta = (\beta_1, \ldots, \beta_n)$, i.e., they can be all characterized by an expression $x_i = y \cdot \beta_i$ for some parameter y.

Linear aggregation: definitions. For linear dynamical systems, it is reasonable to restrict ourselves to *linear* aggregations, i.e., to macrovariables y_a which linearly depend on the the corresponding microvariables x_i, i.e., for which $y_a = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$ for some coefficients ("weights") α_i. As a result, we arrive at the following definition:

Definition 12. *We say that a decomposable aggregation*

$$(i_0, I_1, \ldots, I_k, c_1, \ldots, c_k)$$

is linear *if all the functions c_a are linear, i.e., have the form $c_a(x_{i(a,1)}, \ldots, x_{i(a,n_a)}) = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$ for some coefficients α_i.*

This definition can be reformulated as follows:

Definition 13.

- *By a* linear (decomposable) aggregation*, we mean a tuple $(i_0, I_1, \ldots, I_k, \alpha)$, where $(i_0, I_1, \ldots, I_k)$ is a partition, and $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a tuple of real numbers for which $\alpha_{i_0} \ne 0$.*
- *For every microstate $x = (x_1, \ldots, x_n)$, by the corresponding* macrostate *we mean a tuple $y = (y_1, \ldots, y_k)$, where $y_a = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$.*

Definition 14. *Let $k > 0$ be a positive integer.*

- *We say that a dynamical system is* linearly k-aggregable *if it is consistent with some linear decomposable aggregation $(i_0, I_1, \ldots, I_k, \alpha)$.*
- *We say that a dynamical system is* linearly $\le k$-aggregable *if it is linearly ℓ-aggregable for some $\ell \le k$.*

Definition 15. *We say that a dynamical system is* linearly aggregable *if it is k-linearly aggregable for some integer k.*

Formulation of the problem. For every integer $k > 0$, we arrive at the following *linear k-aggregability* problem:

- *given* a linear dynamical system;
- *check* whether the given system is linearly k-aggregable.

When such an aggregation exists, the next task is to *compute* it, i.e., to find the partition $I_1, \ldots, I_k$ and the weights α_i which form the corresponding aggregation.

Analysis of the problem. In matrix terms, a linear dynamic equation has the form $x' = cx$. Once the partition $I_1, \ldots, I_k$ is fixed, we can represent each n-dimensional state vector x as a combination of vectors $x^{(a)}$ formed by the components x_i, $i \in I_a$. In these terms, the equation $x' = cx$ can be represented as $x'^{(a)} = \sum\limits_b c^{(a),(b)} x^{(b)}$, where $c^{(a),(b)}$ denotes the corresponding block of the matrix c (formed by elements $c_{i,j}$ with $i \in I_a$ and $j \in I_b$).

For the corresponding linear combinations $y_a = \alpha^{(a)\,T} x^{(a)}$, the dynamics takes the form $y'_a = \sum\limits_b \alpha^{(a)\,T} c^{(a),(b)} x^{(b)}$. The only possibility for this expression to only depend on the combinations $y_b = \alpha^{(b)\,T} x^{(b)}$ is when for each b, the coefficients of the dependence of y'_a on x_i, $i \in I_b$, are proportional to the corresponding weights α_i, i.e., when for every a and b, we have $\alpha^{(a)\,T} c^{(a),(b)} = \lambda_{a,b} \alpha^{(b)\,T}$ for some number $\lambda_{a,b}$. By transposing this relation, we conclude that

$$c^{(a),(b)\,T} \alpha^{(a)} = \lambda_{a,b} \alpha^{(b)}. \tag{9}$$

First known result: the problem is, in general, computationally difficult (NP-hard). The first known result is that in general, the linear aggregability problem is NP-hard even for $k = 2$ [6, 7]. This means that even for linear systems (unless P=NP), there is no hope of finding a *general* feasible method for detecting decomposable aggregability.

Second known result: once we know the partition, finding the weights α_i is possible. The above mentioned result is that in general, finding the partition under which the system is aggregate is computationally difficult (NP-hard).

As we have mentioned, in some practical situations, the partition comes from the natural clustering of the variables and is, therefore, known. In the case when the partition is found, it is possible to feasibly find the weights α_i of the corresponding linear macrocombinations y_a [6, 7].

The main idea behind the corresponding algorithm is as follows. From the above equation (9), for $a = b$, we conclude that $\alpha^{(a)}$ is an eigenvector of the matrix $c^{(a),(a)\,T}$. Since the weight vectors $\alpha^{(a)}$ are defined modulo a scalar factor, we can thus select one of the (easy-to-compute) eigenvectors of $c^{(a),(a)\,T}$ as $\alpha^{(a)}$.

Once we know $\alpha^{(a)}$ for one a, we can determine all other weight vectors $\alpha^{(b)}$ from the condition (9), i.e., as $\alpha^{(b)} = c^{(a),(b)\,T}\alpha^{(a)}$.

2 Conditional Aggregation

2.1 What is Conditional Aggregability: General Case

Aggregation: reminder. As we have mentioned, in practice, quantities corresponding to different variables x_i can be usually grouped into clusters $I_1, \ldots, I_k$ in such a way that interactions within each cluster are much stronger than interactions across different clusters. In the above text, we considered systems which are (decomposably) aggregable in the sense that in each block I_a, we can find an appropriate combination of variables $y_a = c_a(x_{i(a,1)}, \ldots, x_{i(a,n_a)})$ in such a way that for *all* possible states $x = (x_1, \ldots, x_n)$, the change in the new variables is only determined by the values of these new variables, i.e., that we have a simpler system $\frac{dy_a}{dt} = h_a(y_1, \ldots, y_k)$. This reduction to a simpler system drastically simplifies computations related to the dynamical behavior of the original system.

In practice, we can restrict ourselves to "modular" states. Systems which are, in this sense, "unconditionally" aggregable, i.e., aggregable for all possible states $x = (x_1, \ldots, x_n)$, are rather rare. However, in practice, we rarely encounter the need to consider arbitrary states x. Specifically, we know that the interaction within each cluster in much stronger than interactions across different clusters.

In the ideal case when a cluster does not interact with other clusters at all, the interaction within the cluster will lead to an equilibrium state of this cluster, i.e., the values of the corresponding microvariables variables $x_{i(a,1)}, \ldots, x_{i(a,n_a)}$ will stop changing with time and reach an equilibrium state: $x'_{i(a,k)}(t) = f_{i(a,k)}(x_{i(a,1)}(t), \ldots, x_{i(a,n_a)}(t)) = 0$. Since interactions across clusters are much weaker, it is reasonable to assume that in spite of this interaction, the state within each cluster is very close to an equilibrium state. In particular, in the first approximation, we can assume that within each cluster, we get the exact equilibrium.

It is therefore reasonable to restrict the decomposability property of the system only to the such states in which within each cluster, we have equilibrium.

Towards exact description of conditional aggregability. We have already mentioned that in general, a biologically relevant dynamical system has a 1-dimensional family of equilibrium states, i.e., a family which is determined by a single parameter y. The values of all other variables x_i are uniquely determined by this value y.

Thus, to describe the combination of equilibrium states corresponding to k different clusters, we must describe the values of the corresponding k variables y_a, $1 \le a \le k$ and the dependence $x_i = F_i(y_a)$ of each microvariable x_i on the "macrovariable" y_a of the corresponding cluster. In these terms, conditional (decomposable) aggregability means that there exist functions $h_a(y_1, \ldots, y_k)$ such that in the equilibrium state, the evolution of the macrovariables is determined by the system $\dfrac{dy_a}{dt} = h_a(y_1, \ldots, y_k)$, and in the new state, every cluster a in still in the equilibrium state determined by the new value $y_a(t+1)$ of the corresponding macrovariable.

Formal definition of conditional aggregability. The above analysis leads to the following definitions.

Definition 16.

- *By a* conditional aggregation*, we mean a tuple* $(i_0, I_1, \ldots, I_k, F_1, \ldots, F_n)$*, where* $(i_0, I_1, \ldots, I_k)$ *is a partition, and for each* i *from 1 to* n*,* $F_i : \mathbb{R} \to \mathbb{R}$ *is a function of one variable such that the function* F_{i_0} *actually depends on* x_{i_0}*.*
- *By a* macrostate*, we mean a tuple* $y = (y_1, \ldots, y_k)$*.*
- *By a* microstate *corresponding to a macrostate* y*, we mean a state* $x = (x_1, \ldots, x_n)$ *in which for every index* i*, we have* $x_i = F_a(y_a)$*, where* a *is the cluster containing* i *(*$i \in I_a$*).*
- *A microstate is called* modular *if it corresponds to some macrostate* y*.*

Definition 17. *We say that a conditional aggregation*

$$(i_0, I_1, \ldots, I_k, F_1, \ldots, F_n)$$

is consistent *with a dynamical system* $(n, f_1, \ldots, f_n)$ *if for every trajectory for which at some moment* t*, the microstate* $x(t)$ *is modular, it remains modular for all following moments of time* $t' > t$*.*

Definition 18. *Let* $k > 0$ *be a positive integer.*

- *We say that a dynamical system is* conditionally k-aggregable *if it is consistent with some conditional aggregation* $(i_0, I_1, \ldots, I_k, F_1, \ldots, F_n)$*.*
- *We say that a dynamical system is* conditionally $\le k$-aggregable *if it is conditionally* ℓ*-aggregable for some* $\ell \le k$*.*

Definition 19. *We say that a dynamical system is* conditionally aggregable *if it is conditionally* k*-aggregable for some integer* k*.*

Example of conditional aggregation: phenotype-based description of an additive genetic trait. In general, the description of recombination and natural selection is a quadratic dynamical system [14, 15, 16]. Specifically, from one generation t to the next one $(t{+}1)$, the absolute frequency p_i (number of individuals with genotype i in a population) changes as follows:

$$p_z(t+1) = \sum_i \sum_j w_i \cdot w_j \cdot p_i(t) \cdot p_j(t) \cdot R_{ij\to z},$$

where w_i is the fitness of the i-th genotype (probability of survival multiplied by the number of offsprings), and $R_{ij\to z}$ is the recombination function that determines the probability that parental types i and j produce progeny z.

Let us assume that we have two alleles at each of g loci. In this case, each genotype i can be described as a binary string. A frequent simplifying assumption in quantitative genetics is that the contribution of each locus to phenotype is equal. In precise terms, this means that the fitness w_i depends only on the number of 1s a_i in the corresponding binary string: $w_i = w_{a_i}$; different genotypes correspond to different numbers a of 1s in a genotype. In this case, since recombination at different loci are independent, the recombination function takes the form [1, 21] $R_{ij\to z} = R_{a_i a_j \to a_z}(L)$, where L is the number of common (overlapping) 1s between the binary sequences i and j: e.g., the sequences 1010 and 0011 have one overlapping 1 (in the 3rd place), and

$$R_{ab\to c}(L) = \left(\frac{1}{2}\right)^{a+b-2L} \binom{a+b-2L}{c-L}.$$

In this situation, since different genotypes i within the same phenotype a have the same fitness, it is reasonable to assume that all these genotypes have the same frequency within each phenotype class $p_i = p_{a_i}$. It is easy to see that this this equal-frequency distribution is an equilibrium, i.e., that if we start with equal genotype frequencies within each phenotype $p_i(t) = p_{a_i}(t)$, then in the next generation, we also have equal genotype frequencies $p_i(t+1) = p_{a_i}(t+1)$. It was shown [1] that for many reasonable fitness functions w_a, that this internal equilibrium solution is stable in the sense that if we apply a small deviations to this equilibrium, the system asymptotically returns to the equilibrium state.

In this case, the phenotype frequencies p_a are the macrovariables y_a, and each microvariable p_i is simply equal to the corresponding macrovariable $p_i = y_a$ (i.e., $F_i(y_a) = y_a$).

For the macrovariables p_a, the dynamic equations take the form

$$p_c(t+1) = \sum_a \sum_b w_a \cdot w_b \cdot p_a(t) \cdot p_b(t) \cdot R_{ab\to c},$$

where

$$R_{ab\to c} = \sum_L P(L) \cdot R_{ab\to c}(L)$$

and

$$P(L) = \frac{\binom{i}{L}\binom{g-i}{j-L}}{\binom{g}{j}}$$

is the probability that in the equal-frequency state, the overlap is L.

Possibility of multi-parametric families of equilibria states: a comment. It is worth mentioning that in some biologically important situations, we have multi-parametric families of equilibrium states. An example of such a situation is *linkage equilibrium* (see, e.g., [8, 9, 10, 20]), when to describe the equilibrium frequencies x_i of different genotypes i, it is sufficient to know the frequencies f_ℓ of alleles ℓ at different loci; then, for a genotype $i = \ell_1 \ldots, \ell_m$, the corresponding frequency is equal to the product of the frequencies of its alleles: $x_i = f_{\ell_1} \cdot \ldots \cdot f_{\ell_m}$.

If we have two alleles at each locus, then the sum of their frequencies is 1, so to describe the frequencies of these alleles, it is sufficient to describe one of the frequencies. In this case, for g loci with two alleles at each locus, there are $n = 2^g$ possible genotypes, so in general, we need 2^g different frequencies x_i to describe the state of this system. However, under the condition of linkage equilibrium, we only need g ($\ll 2^g$) frequencies $y_1, \ldots, y_g$ corresponding to g loci.

Such situations are not covered by our definitions and will require further analysis.

Conditional aggregation beyond equilibria. Our main motivation for the conditional aggregation was based on the assumption that within each each cluster, the state reaches an equilibrium. This assumption makes sense for situations in which within-cluster interactions are much stronger than between-cluster interactions. However, as we have mentioned, this is not a necessary condition for aggregation. In situations where the between-cluster interaction is not weak, we can still have conditional aggregation – with microstates no longer in equilibrium within each cluster.

To take this possibility into account, in the following text, we will call the corresponding states of each cluster *quasi*-equilibrium states.

2.2 Linear Case

Discission. The main idea behind conditional aggregation is that we only consider "modular" states, i.e., states in which an (quasi-)equilibrium is attained within each cluster. For linear systems, as we have mentioned earlier, (quasi-)equilibrium means that for each cluster I_a, we have $x_i = y_a \cdot \beta_i$ for all $i \in I_a$. Here, β_i are the values which characterize a fixed quasi-equilibrium state, and y_a is a parameter describing the state of the a-th cluster.

Since in such modular states, the state of each cluster is uniquely characterized by the value y_a, this value y_a serves as a *macrovariable* characterizing this state.

Formal definition. We thus arrive at the following definition.

Definition 20.
We say that a conditional aggregation $(i_0, I_1, \ldots, I_k, F_1, \ldots, F_n)$ *is* linear *if all the functions* F_i *are linear, i.e., if* $F_i(y_a) = \beta_i \cdot y_a$ *for all* i.

This definition can be reformulated in the following equivalent form.

Definition 21.

- *By a* linear conditional aggregation*, we mean a tuple* $(i_0, I_1, \ldots, I_k, \beta)$*, where* $(i_0, I_1, \ldots, I_k)$ *is a partition, and* $\beta = (\beta_1, \ldots, \beta_n)$ *is a tuple of real numbers for which* $\beta_{i_0} \neq 0$.
- *By a* microstate *corresponding to a macrostate* $y = (y_1, \ldots, y_k)$*, we mean a state* $x = (x_1, \ldots, x_n)$ *in which for every index* i*, we have* $x_i = y_a \cdot \beta_i$*, where* a *is the cluster containing* i $(i \in I_a)$.
- *A microstate is called* modular *if it corresponds to some macrostate* y.

Definition 22. *Let* $k > 0$ *be a positive integer.*

- *We say that a dynamical system is* linearly conditionally k-aggregable *if it is consistent with some conditional linear aggregation* $(i_0, I_1, \ldots, I_k, \beta)$.
- *We say that a dynamical system is* linearly conditionally $\leq k$-aggregable *if it is linearly conditionally* ℓ*-aggregable for some* $\ell \leq k$.

Definition 23. *We say that a dynamical system is* linearly conditionally aggregable *if it is conditionally linearly* k*-aggregable for some integer* k.

Formulation of the problem. For every integer $k > 0$, we arrive at the following *linear conditional* k*-aggregability* problem:

- *given* a linear dynamical system;
- *check* whether the given system is linearly conditionally k-aggregable.

When such an aggregation exists, the next task is to *compute* it, i.e., to find the partition $I_1, \ldots, I_k$ and the weights β_i which form the corresponding conditional aggregation.

Discussion. The main reason why we started discussing the notion of conditional aggregability is that the original notion of decomposable aggregability required decomposability for *all* possible states – and was, therefore, too restrictive. Instead, we require decomposability only for modular states, in which we have a quasi-equilibrium within each cluster. This part of the requirement of conditional aggregability is thus *weaker* than the corresponding condition of decomposable aggregability.

On the other hand, in decomposable aggregability, we only worried about the dynamics of macrostates, while in conditional aggregability, we also require that microstates also change accordingly (i.e., modular state are transformed into modular states). This part of the requirement of conditional aggregability is thus *stronger* than the corresponding condition of decomposable aggregability.

Since one part of the requirement is weaker and the other part of the requirement is stronger, it is reasonable to conjecture that the requirements themselves are of approximately equal strength. It turns out that, indeed, the two corresponding problems have the exact same computational complexity.

Main results. In this paper, we prove the following two results:

Proposition 1. *For every $k \geq 2$, the linear conditional k-aggregability problem is NP-hard.*

Proposition 2. *There exists an efficient (polynomial-time) algorithm that, given a linear dynamical system (n, c) and a partition $(i_0, I_1, \ldots, I_k)$ under which the system is linearly conditionally aggregable, returns the corresponding weights β_i.*

The proof of both results is based on the following auxiliary statement. For every matrix c, let c^T denote a transposed matrix, with $c^T_{i,j} \overset{\text{def}}{=} c_{j,i}$.

Proposition 3. *A linear dynamical system (n, c) is linearly decomposably aggregable if and only if the system (n, c^T) is linearly conditionally aggregable (for the same partition).*

These results show that not only the two above statements are true, but also that the problems of detecting linear decomposable aggregability and linear conditional aggregability have the exact same computational complexity. For example, if we can solve the problem of detecting linear decomposable aggregability, then we can apply this algorithm to the transposed matrix c^T and thus get an algorithm for detecting linear conditional aggregability. Vice versa, if we can solve the problem of detecting linear conditional aggregability, then we can apply this algorithm to the transposed matrix c^T and thus get an algorithm for detecting linear decomposable aggregability.

So, to prove Propositions 1 and 2, it is sufficient to prove the auxiliary Proposition 3.

Proof of Proposition 3. By definition, for a given partition $(i_0, I_1, \ldots, I_k)$, linear conditional aggregability means that for every macrostate $y = (y_1, \ldots, y_k)$, i.e., for all possible values $y_1, \ldots, y_k$, the equations of the dynamical system $x'_i = \sum\limits_{j=1}^{n} c_{i,j} \cdot x_j$ transform the corresponding modular state $x_j = y_a \cdot \beta_j$ ($j \in I_a$) into a modular state x'_i. In particular, for every cluster a ($1 \leq a \leq k$), the corresponding modular state takes the form $x_j = \beta_j$ for $j \in I_a$ and $x_j = 0$ for all other j. For this modular state, the new state x'_i takes the form $x'_i = \sum\limits_{j \in I_a} c_{i,j} \cdot \beta_j$.

This equation can be simplified if we use the notations that we introduced in our above analysis of linear dynamical systems. Specifically, we can represent each n-dimensional state vector x as a combination of vectors $x^{(a)}$ formed by the components x_i, $i \in I_a$. In these terms, the above equation takes the form $x'^{(b)} = c^{(a),(b)}\beta^{(a)}$ for all b. The new state x' must also be a modular state, so for every cluster b, the corresponding state $x'^{(b)}$ must be proportional to the fixed quasi-equilibrium state $\beta^{(b)}$ of this cluster: $x'^{(b)} = \lambda_{a,b}\beta^{(b)}$ for some constant $\lambda_{a,b}$. Thus, for every two clusters a and b, we must have

$$c^{(a),(b)}\beta^{(a)} = \lambda_{a,b}\beta^{(b)}. \tag{10}$$

Vice versa, if this equation is satisfied, one can easily check that for every macrostate y, the corresponding modular state is indeed transformed into a new modular state.

So, for a given partition $(i_0, I_1, \ldots, I_k)$, a linear dynamical system (n, c) is linearly conditionally aggregable if and only if there exist vectors $\beta^{(a)}$ for which the equations (10) hold for some values $\lambda_{a,b}$. As we have mentioned earlier, a system (n, c) is linearly decomposably aggregable if and only if there exist vectors $\alpha^{(a)}$ for which the equations (10) hold for some values $\lambda_{a,b}$. The only difference between the equations (10) and (9) (apart from different names for $\alpha^{(a)}$ and $\beta^{(a)}$) is that in (10), we have the original matrix c, while in (9), we have the transposed matrix c^T. Thus, the linear system (n, c) is linearly conditionally aggregable if and only if the system (n, c^T) with a transposed matrix c^T is linearly decomposably aggregable. The proposition is proven.

Corollary. In the practically important case when the matrix c describing a linear dynamical system is symmetric $c = c^T$, the above Proposition 3 leads to the following interesting corollary:

Proposition 4. *A linear dynamical systems (n, c) with a symmetric matrix c is linearly conditionally aggregable if and only if it is linearly decomposably aggregable.*

Approximate aggregability: observation. One of the main cases of conditional aggregation is when we have clusters with strong interactions within a cluster and weak interactions between clusters. Due to the weakness of across-cluster interactions, it is reasonable to assume that the state of each cluster is *close* to the equilibrium. In the above text, we assumed that the clusters are *exactly* in the (quasi-)equilibrium states. In real life, such systems are only *approximately* conditionally aggregable.

Examples of approximately conditionally aggregable systems are given, e.g., in [24]. For an application to population genetics see [23].

Is detecting *approximate* linear conditional aggregability easier than detecting the (exact) linear conditional aggregability? In our auxiliary result, we have shown that the problem of detecting linear conditional aggregability is equivalent to a problem of detecting linear decomposable aggregability (for a related linear dynamical system). One can similarly show that approximate linear conditional aggregability is equivalent to approximate linear decomposable aggregability. In [6, 7], we have shown that detecting approximate linear decomposable aggregability is also NP-hard. Thus, detecting approximate linear conditional aggregability is NP-hard as well – i.e., the approximate character of aggregation does not make the corresponding computational problems simpler.

3 Identifying Aggregations in Lotka-Volterra Equations with Intraspecific Competition

3.1 Formulation of the Problem

Motivations. In the previous sections, we mentioned that in general, identifying aggregations is a computationally difficult (NP-hard) problem. This means that we cannot expect to have a feasible aggregations-identifying algorithm that is applicable to an *arbitrary* dynamical system. We can, however, hope to get such a feasible algorithm for specific classes of biology-related dynamical systems.

In this paper, we start with possible most well-known biology-related dynamical systems: Lotka-Volterra equations; see, e.g., [11, 12].

Lotka-Volterra equations. The standard Lotka-Volterra equations for competition between multiple species x_i exploiting the same resource in a community is

$$\frac{dx_i}{dt} = r_i \cdot x_i \cdot \left(1 - \frac{\sum\limits_j a_{ij} \cdot x_j}{K_i}\right), \tag{11}$$

where K_i is the *carrying capacity* of the i-th species, and a_{ij} is a measure of the intensity of competition between species i and j. In this equation:

- the terms a_{ij} corresponding to $i \neq j$ describe *interspecific* competition, i.e., competition between different species, while
- the term a_{ii} describes *intraspecific* competition, i.e., competitions between organized of the same species.

In this paper, we will only consider the case where there is an intraspecific competition, i.e., where $a_{ii} \neq 0$ for all i.

Known aggregation results about Lotka-Volterra equations. The main known results about the aggregability of the Lotka-Volterra equations are described by Iwasa *et al.* in [4, 5]. Specifically, these papers analyze a simple case of aggregation when there are classes of competitors $I_1, \ldots, I_k$ such that:

- all the species i within the same class I_a have the same values of r_i and K_i;

- the interaction coefficients a_{ij} depend only on the classes I_a and I_b to which i and j belong, i.e., for all $i \in I_a$ and $j \in I_b$, the coefficient a_{ij} has the same value.

In this case, the actual aggregation of microvariables is simple and straightforward: we can have $y_a = \sum\limits_{i \in I_a} x_i$.

In [22, 23], it is shown that a similar "weighted" linear aggregation, with $y_a = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$ and possible different weights α_i, is sometimes possible in situations when the values a_{ij} are not equal within classes – namely, it is possible when the values a_{ij} satisfy some symmetry properties. In this section, we will analyze the general problem of linear aggregability of such systems of equations.

Restriction to practically important cases. Before we formulate a precise mathematical formulation of our result, let us once again recall why this problem is practically useful. As we have mentioned earlier, the main reason why aggregation is important is because aggregation simplifies the analysis of the complex large-size dynamical equations – by reducing them to simpler smaller-size ones, of size $k \ll n$. From this viewpoint, the fewer classes we have, the simpler the reduced system, and the more important its practical impact.

From this viewpoint, the most practically interesting reduction is the reduction with the smallest possible number of classes. In other words, it is important to know whether we can subdivide the objects into 10 classes or less – but once we know that we can subdivide the objects into 7 classes, then the problem of checking whether we can also have a non-trivial subdivision into 9 classes sounds more academic.

In view of this observation, instead of checking whether a given system can be decomposed into exactly k classes, we study the possibility of checking whether it can be subdivided into $\le k$ classes. Thus, we arrive at the following problem.

Exact formulation of the problem. For every integer $k > 0$, we arrive at the following *linear k-aggregability problem for Lotka-Volterra equations:*

- *given:* a Lotka-Volterra system, i.e., rational values n, r_i, K_i, $(1 \le i \le n)$, and a_{ij} $(1 \le i \le n,\ 1 \le j \le n)$;
- *check* whether the given system is linearly $\le k$-aggregable.

When such an aggregation exists, the next task is to *compute* it, i.e., to find the partition $I_1, \ldots, I_k$ and the weights α_i which form the corresponding conditional aggregation.

3.2 Analysis of the Problem

Linearization seems to indicate that this problem is NP-hard. One can easily check that if a non-linear system $(n, f_1, \ldots, f_n)$ is k-aggregable, then for each state $x^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$ and for the deviations $\Delta x_i \stackrel{\text{def}}{=} x_i - x_i^{(0)}$, the corresponding linearized system

$$\Delta x_i' = f_i(x^{(0)}) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \cdot \Delta x_j \tag{12}$$

is also k-aggregable.

In particular, if the Lotka-Volterra equation is k-aggregable, then the corresponding linearized system

$$\Delta x_i' = \left(r_i - \sum_{j=1}^{n} r_i \cdot a_{ij} \cdot K_i^{-1} \cdot x_j^{(0)} \right) \cdot \Delta x_i - r_i \dot{x}_i^{(0)} \cdot K_i^{-1} \cdot \sum_{j=a}^{n} a_{ij} \cdot \Delta x_j \tag{13}$$

should also be k-aggregable. Since in the general Lotka-Volterra equations, we can have an arbitrary matrix a_{ij}, the corresponding linearized systems can have an arbitrary matrix $c_{i,j}$.

We already know that for general linear systems, the general problem of detecting linear k-aggregability for an arbitrary matrix $c_{i,j}$ is NP-hard. So, at first glance, it may seem like for Lotka-Volterra equations, the problem of detecting linear k-aggregability should also be NP-hard.

Why the above argument for NP-hardness is not a proof. In spite of the above argument, we will show that a feasible algorithm is possible for detecting k-aggregability of Lotka-Volterra equations. This means that the above argument in favor of NP-hardness cannot be transformed into a precise proof.

Indeed, the result about NP-hardness of the linear problem means that it is computationally difficult to check k-aggregability of a *single* linear system. On the other hand, k-aggregability of a non-linear system means, in general, that *several* different linear dynamic systems are k-aggregable – namely, the linearized systems (12) corresponding to all possible states $x^{(0)}$. So, even if for some state $x^{(0)}$, it is difficult to check k-aggregability, we may be able to avoid this computational difficulty if for other states $x^{(0)}$, the corresponding linear system is easily proven *not* to be k-aggregable.

3.3 Main Result

Result. The main result of this section is that for every $k > 0$, there exists a feasible (polynomial-time) algorithm for solving the above problem:

Proposition 5. *For every $k > 0$, there exists a polynomial-time algorithm for solving linear k-aggregability problem for Lotka-Volterra equations.*

Practically useful corollary: how to compute the corresponding weights. As we will see from the proof, identifying the aggregating partition is feasible but rather complicated.

However, as well see from the same proof, once we know the aggregating partition $I_1, \ldots, I_k$, we have a straightforward formula for determining the wights α_i of the corresponding macrovariables $y_a = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$: namely, we can take $\alpha_i = r_i \cdot a_{ii} \cdot K_i^{-1}$.

Discussion. It may be worth mentioning that the approach behind our algorithm will be not work for a general recombination system (as described above). Specifically, in our algorithm, we essentially used the fact that in the Lotka-Volterra equations, all the quadratic terms in the expression for the new value x'_i are proportional to the previous value x_i of the same quantity. In contrast, in the recombination system, this is not necessarily the case, because a genotype z need not be a progeny of z and some other genotype.

3.4 Proof

Reduction to minimal aggregability. According to the precise formulation of our problem, we want to know, for a given $k > 0$, whether there exists a linear ℓ-aggregation for some $\ell \leq k$. If such a linear aggregation exists, then among all such aggregations we can select a *minimal* one, i.e., a linear aggregation for which no linear aggregation with fewer classes is possible. Thus, to check whether a system is linearly ℓ-aggregable for some $\ell \leq k$, it is sufficient to check whether it is minimally linearly ℓ-aggregable for some $\ell \leq k$.

Once we have feasible algorithms for checking minimal linear ℓ-aggregability for different ℓ, we can then apply these algorithms for $\ell = 1, 2, \ldots, k$ and thus decide whether the original system is $\leq k$-aggregable. For every given k, we have a finite sequence of feasible (polynomial-time) algorithms. The computation time for each of these algorithms is bounded by a polynomial of the size of the input. Thus, the total computation time taken by this sequence is bounded by the sum of finitely many polynomials – i.e., by a polynomial.

In view of this observation, in the following text, we will design, for a given integer $k > 0$, an algorithm for detecting minimal linear k-aggregability of a given Lotka-Volterra equation.

Simplification of the Lotka-Volterra equation. In order to describe the desired algorithm, let us first reformulate the Lotka-Volterra equations in a simplified form $x'_i = r_i \cdot x_i - \sum\limits_j b_{ij} \cdot x_j$, where $b_{ij} \stackrel{\text{def}}{=} r_i \cdot a_{ij} \cdot K_i^{-1}$.

Linear aggregability: reminder. Linear k-aggregability means that for the macrovariables $y_a = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$, their changes $y'_a = \sum\limits_{i \in I_a} \alpha_i \cdot x'_i$ are uniquely determined by the old ones $y_1, \ldots, y_k$. Substituting the expression for x'_i into the formula for y'_a, we conclude that

$$y'_a = \sum_{i\in I_a} \alpha_i \cdot r_i \cdot x_i - \sum_{i\in I_a} \sum_{j=1}^{n} \alpha_i \cdot b_{ij} \cdot x_i \cdot x_j. \tag{14}$$

Dividing the sum over all j into sums corresponding to different classes a, we conclude that

$$y'_a = \sum_{i\in I_a} \alpha_i \cdot r_i \cdot x_i - \sum_{i\in I_a} \sum_{j\in I_a} \alpha_i \cdot b_{ij} \cdot x_i \cdot x_j - \sum_{b\neq a} \sum_{i\in I_a} \sum_{j\in I_b} \alpha_i \cdot b_{ij} \cdot x_i \cdot x_j. \tag{15}$$

This expression must depend only on the values $y_1, \ldots, y_k$. Since the expression for y'_a in terms of microvariables x_i is quadratic, and $y_1, \ldots, y_k$ are linear functions of the microvariables, the dependence of y'_a on $y_1, \ldots, y_k$ must also be quadratic.

Since y'_a depends only on the variables x_i for $i \in I_a$, we can only have a linear term proportional to y_a. Similarly, since quadratic terms are proportional to x_i for $i \in I_a$, quadratic terms in the expression for y'_a must be proportional to y_a. So, we arrive at the following expression:

$$y'_a = R_a \cdot y_a + B_{aa} \cdot y_a^2 + \sum_{b\neq a} B_{ab} \cdot y_a \cdot y_b. \tag{16}$$

Substituting the expressions $y_a = \sum\limits_{i\in I_a} \alpha_i \cdot x_i$ into the right-hand side of the formula (16), we conclude that

$$y'_a = R_a \cdot \sum_{i\in I_a} \alpha_i \cdot x_i + B_{aa} \cdot \left(\sum_{i\in I_a} \alpha_i \cdot x_i \right)^2 +$$

$$\sum_{b\neq a} B_{ab} \cdot \left(\sum_{i\in I_a} \alpha_i \cdot x_i \right) \cdot \left(\sum_{j\in I_b} \alpha_j \cdot x_j \right). \tag{17}$$

Aggregability means that the right-hand sides of the expressions (16) and (17) must coincide for all possible values of the microvariables x_i. Both right-hand side are quadratic functions of x_i. For the quadratic functions to coincide, they must have the exact same coefficients at x_i and the exact same coefficients at all the products $x_i \cdot x_j$. Let us see what we can conclude about the system from this condition.

Possibility of zero weights: analysis of the degenerate case. Let us first take into account that, in general, it is possible that the weight α_j of some variables is 0; our only restriction is that $\alpha_{i_0} \neq 0$ for a fixed microvariable i_0.

By definition of linear aggregation, the fact that $\alpha_j = 0$ for some j means that none of the macrovariables $y_1, \ldots, y_k$ depend on the corresponding microvariable x_j and thus, the expression y'_a also cannot depend on x_j.

From the above expression for y'_a, we can thus conclude that for every i for which $\alpha_i \neq 0$, we must have $b_{ij} = 0$. Thus, if $\alpha_i \neq 0$ and $b_{ij} \neq 0$, then we must have $\alpha_j \neq 0$.

As we have just mentioned, we have $\alpha_{i_0} \neq 0$. So, if $b_{i_0 j} \neq 0$, we must have $\alpha_j \neq 0$; if for such j, we have $b_{jk} \neq 0$, then we must have $\alpha_k \neq 0$, etc. This fact can be described in graph terms if we form a directed graph with the microvariables $1, \ldots, n$ as vertices, and a connection $i \to j$ if and only if $b_{ij} \neq 0$. In terms of this graph, if there is a path (sequence of connections) leading from i_0 to j, then $\alpha_j \neq 0$.

It is known that in polynomial time, we can find out whether every vertex can be thus reached; see, e.g., [2]. For this, we first mark i_0 as reachable. At each stage, we take all marked vertices, take all edges starting with them,a nd mark their endpoints. Once there are no new vertices to mark, we are done: if all vertices are marked, this means that all vertices are reachable, otherwise this means that some vertices are not reachable.

At each stage except for the last one, we add at least one vertex to the marked list; thus, the number of steps cannot exceed the number n of vertices. Each step requires polynomial time; thus, overall, this graph algorithm takes polynomial time.

If all states are reachable from i_0, this means that in every aggregation, we must have $\alpha_i \neq 0$. If some states are not reachable, then for these states, we can set $\alpha_i = 0$ and keep the aggregation.

Reduction to non-degenerate case. In view of the above, to check for the existence of a linear aggregation, it is sufficient to first mark all reachable vertices and then to restrict ourselves only to reachable vertices.

For these vertices, $\alpha_i \neq 0$. So, in the following text, we will assume that all the vertices are reachable and all the weights α_i are non-zeros – i.e., that we have a "non-degenerate" situation.

For this non-degenerate situation, let us make conclusions from the equality of the coefficients at x_i and at $x_i \cdot x_j$ in the right-hand sides of the formulas (16) and (17).

Comparing coefficients at x_i. Comparing coefficients at x_i, we get $\alpha_i \cdot r_i = R_a \cdot \alpha_i$. Since $\alpha_i \neq 0$, we can divide both sides of this equality by α_i and conclude that $r_i = R_a$, i.e., that for all i from the same class $i \in I_a$, we have the same value r_i.

Comparing coefficients at x_i^2. Comparing coefficients at x_i^2, we get $\alpha_i \cdot b_{ii} = B_{aa} \cdot \alpha_i^2$. Since $\alpha_i \neq 0$, we conclude that $b_{ii} = B_{aa} \cdot \alpha_i$. Since we only consider the situations with intraspecific competition $b_{ii} \neq 0$, and we know that $\alpha_i \neq 0$, we thus conclude that $B_{aa} \neq 0$ for all a.

Let us use non-uniqueness in y_a to further simplify the formulas. The macrovariables y_a are not uniquely determined. In principle, instead of the original macrovariables y_a, we can consider new macrovariables $\widetilde{y}_a = k_a \cdot y_a$ for arbitrary constants $k_a \neq 0$. Let us use this non-uniqueness to further simplify our equations.

Specifically, we will consider the new macrovariables $\widetilde{y}_a = B_{aa} \cdot y_a$. From the original equation

$$y'_a = R_a \cdot y_a + B_{aa} \cdot y_a^2 + \sum_{b \neq a} B_{ab} \cdot y_a \cdot y_b,$$

we conclude that

$$\widetilde{y}'_a = B_{aa} \cdot y'_a = B_{aa} \cdot R_a \cdot y_a + B_{aa}^2 \cdot y_a^2 + \sum_{b \neq a} B_{aa} \cdot B_{ab} \cdot y_a \cdot y_b.$$

Representing the values y_a and y_b in the right-hand side in terms of the new macrovariables $\widetilde{y}_a$ and $\widetilde{y}_b$, as $y_a = \dfrac{\widetilde{y}_a}{B_{aa}}$ and $y_b = \dfrac{\widetilde{y}_b}{B_{bb}}$, we conclude that

$$\widetilde{y}'_a = R_a \cdot \widetilde{y}_a + y_a^2 + \sum \widetilde{B}_{ab} \cdot \widetilde{y}_a \cdot \widetilde{y}_b,$$

where

$$\widetilde{B}_{ab} \stackrel{\text{def}}{=} \frac{B_{ab}}{B_{bb}}.$$

For these new macrovariables, $\widetilde{B}_{aa} = 1$.

Thus, without losing generality, we can conclude that $B_{aa} = 1$ for all a. In this case, the above conclusion $b_{ii} = B_{aa} \cdot \alpha_i$ takes a simplified form $\alpha_i = b_{ii}$.

Comparing coefficients at $x_i \cdot x_j$ when i and j are in different classes. When $i \in I_a$ and $j \in I_b$ $(a \neq b)$, comparing coefficients at $x_i \cdot x_j$ leads to $\alpha_i \cdot b_{ij} = B_{ab} \cdot \alpha_i \cdot \alpha_j$. Since $\alpha_i \neq 0$, this results in $b_{ij} = B_{ab} \cdot \alpha_j$. We already know that $\alpha_j = b_{jj}$, so we can conclude that for every i and j from different classes $a \neq b$, the ratio

$$r_{ij} \stackrel{\text{def}}{=} \frac{b_{ij}}{b_{jj}}$$

takes the same value B_{ab}, irrespective of the choice of $i \in I_a$ and $j \in I_b$.

Comparing coefficients at $x_i \cdot x_j$ when i and j are in the same class. When $i, j \in I_a$, comparing coefficients at $x_i \cdot x_j$ (and at the same term $x_j \cdot x_i$) and using the fact that $B_{aa} = 1$ leads to the equation

$$\alpha_i \cdot b_{ij} + \alpha_j \cdot b_{ji} = 2\alpha_i \cdot \alpha_j.$$

Dividing both sides of this equality by $\alpha_i = b_{ii}$ and $\alpha_j = b_{jj}$, we conclude that

$$\frac{b_{ij}}{b_{jj}} + \frac{b_{ji}}{b_{ii}} = 2.$$

Using the notation r_{ij} that we introduced in the previous section, we conclude that $r_{ij} + r_{ji} = 2$.

Summarizing the analysis. Combining the analysis of all linear and quadratic terms, we conclude that for the aggregating partition into classes $I_1, \ldots, I_k$, the following must be true:

- for all i within each class I_a, the values r_i are the same: $r_i = R_a$ (for some value R_a);
- for all $i, j \in I_a$, we have $r_{ij} + r_{ji} = 2$;
- for every $a \neq b$, for all $i \in I_a$ and $j \in I_b$, the ratios r_{ij} are the same: $r_{ij} = B_{ab}$ (for some value B_{ab}).

Vice versa, if we have a partition for which these properties are satisfied, then, as one can easily see, we have an aggregation.

Taking minimality into account. As we have mentioned in the beginning of this proof, we are looking for a *minimal* aggregation, i.e., for an aggregation with the smallest possible number of classes. This means, in particular, that if we simply combine two classes $a \neq b$ into a single one, we will no longer get an aggregation. This means, in turn, that one of the three above conditions is not satisfied for the new class, i.e., that (at least) one of the following three things is happening:

- either $R_a \neq R_b$;
- or $B_{ab} + B_{ba} \neq 2$;
- or for some $c \neq a$, $c \neq b$, we have $B_{ac} \neq B_{bc}$ or $B_{ca} \neq B_{cb}$.

Towards an algorithm for distinguishing $i \notin I_a$ versus $i \notin I_b$. To exploit this consequence of minimality, let us select a points s_a in each class I_a. Let us show that once we know these points, we can use the above property to tell, for every two classes $a \neq b$ and for each i, whether $i \notin I_a$ or $i \notin I_b$.

Indeed, at least one of the above three properties holds for $a \neq b$. If this property is $R_a \neq R_b$, then we cannot have both $r_i = R_a = r_{s_a}$ and $r_i = R_b = r_{s_b}$. So:

- if $r_i \neq r_{s_a}$, we have $i \notin I_a$;
- if $r_i \neq r_{s_b}$, we have $i \notin I_b$.

If this property is $B_{ab} + B_{ba} \neq 2$, this means that:

- for $i \in I_a$, we have $r_{is_a} + r_{s_a i} = 2$ but $r_{is_b} + r_{s_b i} = B_{ab} + B_{ba} \neq 2$;
- for $i \in I_b$, we have $r_{is_b} + r_{s_b i} = 2$ but $r_{is_a} + r_{s_a i} = B_{ab} + B_{ba} \neq 2$.

Thus:

- if $r_{is_a} + r_{s_a i} \neq 2$, we have $i \notin I_a$;
- if $r_{is_b} + r_{s_b i} \neq 2$, we have $i \notin I_b$.

If this property is $B_{ac} \neq B_{bc}$, this means that for $i \in I_a$, we have $r_{is_c} = B_{ac} \neq B_{bc}$, while for $i \in I_b$, we have $r_{is_c} = B_{bc} \neq B_{ac}$. Thus:

- if $r_{is_c} \neq r_{s_a s_c} = B_{ac}$, we have $i \notin I_a$;
- if $r_{is_c} \neq r_{s_b s_c} = B_{bc}$, we have $i \notin I_b$.

As a result, we arrive at the following auxiliary algorithm.

Auxiliary algorithm. In this algorithm, we assume that we have selected a representative s_a from each class I_a. This algorithm enables us, given $a \neq b$ and i, to check whether $i \notin I_a$ or $i \notin I_b$. This algorithm works as follows.

On the first stage of this algorithm, we compare r_i with r_{s_a} and r_{s_b}:

- if $r_i \neq r_{s_a}$, we conclude that $i \notin I_a$ (and stop);
- if $r_i \neq r_{s_b}$, we conclude $i \notin I_b$ (and stop);
- otherwise (i.e., if $r_i = r_{s_a} = r_{s_b}$), we go to the next stage.

On the second stage, we do the following:

- if $r_{is_a} + r_{s_a i} \neq 2$, we conclude that $i \notin I_a$ (and stop);
- if $r_{is_b} + r_{s_b i} \neq 2$, we conclude that $i \notin I_b$ (and stop);
- otherwise (i.e., if $r_{is_a} + r_{s_a i} = r_{is_b} + r_{s_b i} = 2$), we go to the next stage.

On the third stage, for all $c \neq a, b$, we compute the values r_{is_c}, $r_{s_c i}$, $r_{s_a s_c}$, $r_{s_c s_a}$, $r_{s_b s_c}$, $r_{s_c s_b}$.

- If for some c, we get $r_{is_c} \neq r_{s_a s_c}$ or $r_{s_c i} \neq r_{s_c s_a}$, we conclude that $i \notin I_a$.
- If for some c, we get $r_{is_c} \neq r_{s_b s_c}$ or $r_{s_c i} \neq r_{s_c s_b}$, we conclude that $i \notin I_b$.

(Due to the above minimality property, this algorithm always decides whether $i \notin I_a$ or $i \notin I_b$.)

For every i, a, and b, this algorithm requires that we compute at most 6 values r_{xs_c} or $r_{s_c x}$ for each of k classes c, to the total of $\leq 6k$ computational steps.

Once we know representatives $s_1, \ldots, s_k$, we can determine the partition $(I_1, \ldots, I_k)$. Let us now show that once we know the representatives $s_1, \ldots, s_k$, we can assign each element i to the appropriate class I_a as follows.

In the beginning, we only know that i belongs to one of the classes I_a, where a belongs to the k-element set $S = \{1, \ldots, k\}$. We will show how we can sequentially decrease this set – until we get a set consisting of a single element.

Indeed, if the set S of possible classes containing i contains at least two different classes $a \neq b$, then we can use the above algorithm to check whether $i \notin I_a$ or $i \notin I_b$. Whichever of the two conclusions we make, in both cases we delete one element from the set S. So, after $k-1$ steps, we get a set S consisting of a single class a – and thus, we have computed the class to which i belongs.

This computation takes $k-1$ applications of the above auxiliary algorithm. So, overall, it takes $(k-1) \cdot 6k = O(k^2)$ steps. For a given k, this is simply a constant.

Once we know a partition, we can check whether it leads to the aggregation. In accordance with the above characterization of the aggregating partition, once we know a partition $I_1, \ldots, I_k$, in order to determine whether it leads to an aggregation, we need to check the following conditions:

- for all i within each class I_a, the values r_i are the same: $r_i = R_a$ (for some value R_a);
- for all $i, j \in I_a$, we have $r_{ij} + r_{ji} = 2$;
- for every $a \neq b$, for all $i \in I_a$ and $j \in I_b$, the ratios r_{ij} are the same: $r_{ij} = B_{ab}$ (for some value B_{ab}).

This checking requires checking all pairs (i, j), $1 \leq i, j \leq n$, so it takes $O(n^2)$ computational steps.

Final algorithm. For a given k, to check k-aggregability of a given Lotka-Volterra system, we try all possible combinations of points $s_1, \ldots, s_k$ ($1 \leq s_a \leq n$). For each of these combinations, we find the corresponding partition and check if it leads to an aggregation.

If one of these partitions leads to an aggregation, the system is aggregable. In the process, we have computed the partition, and we know the weights $\alpha_i = b_{ii}$.

If none of the partitions leads to an aggregation, this means that the original Lotka-Volterra system is not linearly k-aggregable.

Computation time. For each class a, there are n values choices of s_a. We need to make this choice for k different classes, so we test n^k possible tuples $(s_1, \ldots, s_k)$. For each tuple, we take $O(n^2)$ time, so the overall computation time is $n^k \cdot O(n^2) = O(n^{k+2})$.

For a fixed k, this is polynomial time. The proposition is proven.

Comment. It is important to emphasize that while for every given k, the algorithm is polynomial, but its computation time grows exponentially with k. It is not clear whether it is possible to have an algorithm whose computation time grows polynomially with k as well.

Conclusions and Open Problems

Aggerability is an important property of biological systems, a property that simplifies their analysis. In view of this importance, it is desirable to be able to detect aggregability of a given system.

In our previous papers, we analyzed the problem of detecting and identifying aggregability for linear systems. We showed that this problem is, in general, computationally difficult (NP-hard). We also showed that, once an aggregating partition of microvariables $x_1, \ldots, x_n$ into classes $I_1, \ldots, I_k$ is identified, we can efficiently compute the weights α_i describing the corresponding macrovariables $y_a = \sum\limits_{i \in I_a} \alpha_i \cdot x_i$.

In this paper, we extend our analysis in two different directions. First, we consider *conditional aggregability*, i.e., aggregability of modular states. For linear systems, we get results similar to general (unconditional) aggregability: the problem of identifying conditional aggregability is, in general, NP-hard, but once a partition is identified, we can efficiently compute the corresponding weights.

Second, we consider a biologically important case of non-linear systems: Lotka-Volterra systems with interspecific competition. For such systems, we have designed an efficient (polynomial-time) algorithm for identifying aggregability and computing the corresponding weights.

It is desirable to further extend these results. For conditional aggregability, it would be great to extend our results about conditional aggregability to situations like linkage equilibrium, when we have a non-linear relation dependence of microvariables on the macrovariables. For non-linear systems, it is desirable to extend our non-linear results to Lotka-Volterra systems without intraspecial competition, and to other biologically relevant classes of non-linear systems. It is also desirable to extend our results to aggregations in which blocks I_a are allowed to overlap but remain smaller than the set of all the microvariables.

Acknowledgments.

This work was supported in part by NSF grants HRD-0734825, EAR-0225670, and EIA-0080940, and by Texas Department of Transportation grant No. 0-5453.

References

1. Barton NH, Shpak M (2000) The stability of symmetric solutions to polygenic models. Theoretical Population Biology 57:249–263
2. Cormen T, Leiserson CE, Rivest RL, Stein C (2001), Introduction to Algorithms, MIT Press, Cambridge, MA
3. Courtois PJ (1977) Decomposability: queueing and computer system applications. Academic Press, New York
4. Iwasa Y, Andreasen V, Levin SA (1987) Aggregation in model ecosystems. I. Perfect aggregation. Ecological Modelling 37:287–302
5. Iwasa Y, Levin SA, Andreasen V (1989) Aggregation in model ecosystems. II. Approximate aggregation. IMA Journal of Mathematics Applied in Medicine and Biology 6:1–23
6. Kreinovich V, Shpak M (2006) Aggregability is NP-hard. ACM SIGACT, 37(3):97–104
7. Kreinovich V, Shpak M (2007) Decomposable aggregability in population genetics and evolutionary computations: algorithms and computational complexity. In: Kelemen, A (ed.), Computational Intelligence in Medical Informatics, Springer-Verlag (to appear).
8. Laubichler MD, Wagner GP (2000) Organism and character decomposition: Steps towards an integrative theory of biology. Philosophy of Science 67:289–300.
9. Lewonwtin RC (1974) The Genetic Basis of Evolutionary Change, Columbia University Press, New York

10. Lewonwtin RC, Kojima K-i (1960) The evolutionary dynamics of complex polymorphisms. Evolution 14(4):485–472
11. MacArthur R, Levins R (1967) The limiting similarity, convergence and divergence of coexisting species. American Naturalist 101:377385.
12. May, RM (1973) Stability and Complexity in Model Ecosystems, Princeton University Press, Princeton, NJ
13. Moey CCJ, Rowe JE (2004). Population aggregation based on fitness. Natural Computing 3(1):5–19
14. Rabani Y, Rabinovich Y, Sinclair A (1995) A computational view of population genetics. Proceedings of the 1995 Annual ACM Symposium on Theory of Computing, Las Vegas, Nevada, 83–92
15. Rabani Y, Rabinovich Y, Sinclair A (1998) A computational view of population genetics. Random Structures & Algorithms 12(4):313–334
16. Rabinovich Y, Sinclair A, Wigderson A (1992) Quadratic dynamical systems. Proc. 33rd Annual Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, California, 304–313
17. Rowe J (1998) Population fixed-points for functions of unitation. In: Reeves C, Banzhaf W (Eds), Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, Vol. 5
18. Rowe JE, Vose MD, Wright AH (2005). Coarse graining selection and mutation. In: Proceedings of the 8th International Workshop on Foundations of Genetic Algorithms FOGA'2005, Aizu-Wakamatsu City, Japan, January 5–9, 2005, Springer Lecture Notes in Computer Science, Vol. 3469, pp. 176–191
19. Rowe JE, Vose MD, Wright AH (2005) State aggregation and population dynamics in linear systems. Artificial Life 11(4):473–492
20. Shpak M, Gavrilets S (2005) Population genetics: multilocus. In: Encyclopedia of Life Sciences, Wiley, Chichester, 2005, http://www.els.net/
21. Shpak M, Kondrashov AS (1999) Applicability of the hypergeometric phenotypic model to haploid and diploid production. Evolution 53(2):600–604
22. Shpak M, Stadler PF, Wagner GP, Hermisson J (2004) Aggregation of variables and system decomposition: application to fitness landscape analysis. Theory in Biosciences 123:33–68
23. Shpak M, Stadler PF, Wagner GP, Altenberg L (2004). Simon-Ando decomposability and mutation-selection dynamics. Theory in Biosciences 123:139–180
24. Simon H, Ando F (1961) Aggregation of variables in dynamical systems. Econometrica 29:111–138

A What is NP-Hardness: A Brief Description

For readers who are not very familiar with the notion of NP-hardness, this Appendix provides a brief and informal explanation.

The complexity of a computational problem is usually described by the computation time that is needed to solve this problem. This time grows with the number of variables (and with the number of bits which are needed to represent each of these variables).

For some computational problems, this time grows as a polynomial of the size n of the input. For example, the standard algorithms for multiplying two $n \times n$ matrices or solving a system of linear equations with n unknowns grows as const $\cdot n^3$. For large n, this is still feasible. These problems are called *polynomial time* (or P, for short), and the class of such problem is denoted by P.

For some other computational problems, however, the computation time grows exponentially with n, as 2^n or even faster. For example, this growth occurs when we need to search for a subset of the set of n variables, and we need to do an exhaustive search over all 2^n subsets in order to find the desired set. For such algorithms, for reasonable $n \approx 300 - 400$, the number of computational steps exceeds the number of particles in the Universe; thus, such exhaustive-search algorithms are not practically feasible.

In most of these problems, once we have guessed a solution, we can check, in feasible (polynomial) time, whether this guess is indeed a correct solution. Such problems can be "solved" in polynomial time on a hypothetical "non-deterministic" machine, i.e., on a Turing machine that allows non-deterministic (= guess) steps. Because of this possibility, these problems are usually called *non-deterministic polynomial* (NP, for short), and the class of such problems is denoted by NP.

Most computer scientists believe that there are problems in the class NP which cannot be solved in polynomial time, i.e., that NP≠P; however, this has not been proven yet.

Not all the problems from the class of NP are of the same complexity. Some problems from the class NP – e.g., the problem of solving systems of linear equations – are relatively easy in the sense that they can be solved by polynomial time algorithms. Some problem are more difficult than others – because we can reduce every particular case of the first problem to special cases of the second problem. For example, we can reduce solution of a system of linear equations to solving quadratic equations (with 0 coefficients at x_i^2); this means that the general problem of solving systems of quadratic equations is more difficult (or at least not less difficult) than the general problem of solving systems of linear equations.

Some general problems from the class NP are known to be the most difficult ones, in the sense that every other problem from the class NP can be reduced (in the above sense) to this particular problem. Such problems are called *NP-complete*. A similar notion of complexity can be extended to problems outside the class NP, for which we may not know how to check the correctness of the proposed solution in polynomial time. If any problem from the class NP can be reduced to such a problem, then this problem is called *NP-hard*. In these terms, a problem is NP-complete if it is NP-hard and belongs to the class NP.

It is worth mentioning that even if a general problem is NP-hard, its particular instance may be easy to solve. There may be efficient algorithms which solve particular instances from an important subclass of an NP-hard problem, there may be efficient heuristics which, in many cases, solve these problems.