5-14-2023

The biophysics of bacterial collective motion: Measuring responses to mechanical and genetic cues

Merrill Einar Asp
Syracuse University

Abstract

Mechanobiology is an emerging field investigating mechanical signals as a necessary component of cellular and developmental regulation. These mechanical signals play a well-established role in the differentiation of animal cells, whereby cells with identical genes specialize their function and create distinct tissues depending on the physical properties of their environment, such as shear stiffness. These differences arise from the cell's ability to use those incoming signals to inform which genes it expresses and what molecular machinery it builds and activates. Understanding the various missing factors that cause cells with specific genes to express an emergent phenotype is termed the genotype-to-phenotype problem, and mechanical signaling pathways present themselves as a significant piece of this puzzle. Despite the strong evidence for mechanosensing in eukaryotes, the pathways by which prokaryotes respond to mechanical stimuli are still largely unknown. Bacteria are among the simplest and yet most abundant forms of life. Many of their survival strategies depend on multicellular development and the coordinated formation of a colony into functional structures that may also feature cellular differentiation. This dissertation employs bacteria as a model system to investigate multiple biophysical questions of collective motion through novel experimental and analytical techniques. This work addresses the understudied mechanical relationship between a bacterial colony and the substrate it colonizes by asking "what is the effect of substrate stiffness on colony growth?" This is done by measuring bacterial growth on hydrogel substrates that decouple the effects of substrate stiffness from other material properties of the substrate that vary with stiffness. We report a previously unobserved effect in which bacteria colonize

stiffer substrates faster than softer substrates, in opposition to previous studies done on agar, where permeability, viscoelasticity, and other material properties vary with stiffness.

A second theme of this work probes the genetic inputs to the genotype-to-phenotype problem in multicellular development. The bacterial species *Myxococcus xanthus* producing macroscopic aggregates called fruiting bodies is used as a model organism for these studies. It has long been conjectured that genes may stand in for each other functionally, allowing for development to be more consistent and stable, but the extent of this redundancy has resisted measurement. We approach the question "how does redundancy among related genes lead to robust collective behavior?" by quantifying developmental phenotype in a large dataset of time lapse microscopy videos that show development in many mutant strains. We observe that when knocking out multiple genes that have a common origin (i.e. homologous genes), the resulting phenotypes differ from wild-type in a similar way. These phenotype clusters also differ from knockouts from other homologous gene families. These distinct phenotypic clusters provide evidence for the existence of networks of redundant genes that are larger than could previously be tested directly. Because of this robustness, the effects of individual gene mutations can be hidden or damped. We thus develop our analytical techniques further to address the question "how can subtle changes in phenotype be measured?" This involves quantifying the breadth of variation observed in wild-type development and creating a statistical technique to distinguish probabilistic distributions of phenotypic outcomes. We present a coherent method of visualizing large phenotypic datasets that include multiple metrics that we use to distinguish small developmental differences from wild-type, giving each mutant strain a phenotypic

fingerprint that can be used in future studies on gene annotation and environmental impacts on phenotype.

THE BIOPHYSICS OF BACTERIAL COLLECTIVE MOTION: MEASURING RESPONSES TO MECHANICAL AND GENETIC CUES

by

Merrill E. Asp

B.S., Brigham Young University, 2016

Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Physics.

Syracuse University

May 2023

To my mother,

a steadfast source of love and support,

till pappa,

som alltid trodde på mig,

and to Chelsea,

my closest friend and constant companion.

Acknowledgements

No Ph.D. candidate can complete their work without a network of resources and support. I want to thank my advisor Alison Patteson for her patience and tireless support in guiding me into the interdisciplinary world of biophysics and for her help with the work presented in this dissertation. I also want to acknowledge the other examples of excellence in the many facets of academia I have learned from, such as my committee members Roy Welch, Arvind Gopinath, Lisa Manning, Teng Zhang, and Joey Paulsen. I want to thank my lab mates for working alongside me to build the Patteson lab and for their feedback on this dissertation. I could not have made the most of my Ph.D. without the thoughtfulness of mentors like Walter Freeman and Jenny Ross, and the help of our office staff, especially Patty Whitmore, Yudaisy Salomón Sargentón, Juliette Rawda, and Cassandra Ellis.

I would like to thank the Syracuse University Physics Department and BioInspired institute for supporting my work and professional development, and the National Science Foundation for providing the grant funding that supported my research.

I am indebted to my close friends, Ohana Benevides Rodrigues and Sarthak Gupta, for support, collaboration, and many wonderful discussions as we worked through our time as Ph.D candidates together.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Mechanobiology and the genotype-to-phenotype problem

Although every cell in a multicellular organism shares identical genes, different cells specialize their behaviors and structure, and groups of cells can self-organize into distinct and diverse tissues with a range of properties and functions [1]. The cell genome, the full sequence of DNA base pairs stored in a cell, encodes all the proteins and molecular machinery a cell can synthesize and express. Traditionally, the function of each gene is deduced by examining the impact of the mutation of that gene on cell and tissue morphology and development. A complete description of development was thus once thought to be found in studying the genome with sufficient depth and rigor.

However, even when we know an organism's full genome, it is often not sufficient to predict cellular and developmental behaviors. This is referred to as the genotype-to-phenotype problem. A phenotype is any observable characteristic of a living system, and it is clear that an organism's phenotype is a function of both its genome and its environment. Plants grow toward light, fungi form spores when starved, and bacteria differentiate into multicellular biofilm colonies when making contact with a surface. External physical and chemical environmental cues shape developmental decisions.

Cells evolved in a physical world and have developed ways to sense and respond to environmental cues to survive. Cells have built sensors that can detect two general types of external signals: chemical, physical, or combination of the two. Chemical signal sensors are typically protein receptors that impact intracellular signaling pathways by binding to specific

external chemical stimulants. A classic example is bacterial chemotaxis, in which cells show directed motility toward or away from a chemical gradient. The responses of cells to chemical signals are relatively well-characterized and understood through the lens of molecular biochemistry and genetics. Compared to chemical signaling, the cellular response to physical stimuli is far less studied [2]. The most described physical sensors detect mechanical features of the extracellular environment and typically rely on coupled motion of a cellular organelle with the environment. Cells translate these mechanical cues into biochemical signals to adapt their behavior, a process known as mechanotransduction. Examples include deflection of primary cilia in ear cells to hear sound [3] and membrane-based sensors that detect pressure gradients across the cell membrane [4].

The field of mechanobiology has emerged in the last twenty years as evidence has grown of the impact of mechanics on cell phenotype. This work has primarily focused on human cells and animal models, which exhibit marked behavioral changes when grown in tissues of differing stiffness values [5]. Tissue stiffness is a critical component of diagnosing diseases such as cancer and fibrosis which cannot be explained in the traditional language of biochemistry. New work is beginning to highlight how bacteria can also sense and respond to the mechanical features of their environment, especially in the context of collective cell morphogenesis and development.

The work presented in this dissertation presents bacterial systems as an attractive model for identifying fundamental mechanisms of mechanosensing, and it explores the applications of bacterial multicellular development in addressing the genotype-to-phenotype problem. In particular, we test the hypothesis that the physical properties of biofilm substrates affect phenotype by designing developmental assays where the stiffness of the substrate is controlled

independently from its other material properties. We also investigate evidence for the hypothesis that redundancy between collections of genes is a mechanism of developmental robustness by designing analytical tools to quantify phenotypic similarity in large datasets. Finally, this work is expanded to test the hypothesis that developmental dynamics can be used to distinguish the subtle changes caused by single-gene mutations with statistical significance. The work done in response to these questions represents a new approach to the genotype-to-phenotype problem that emphasizes mechanobiology and can be used to measure the impact of environmental and genetic changes across a wide variety of living systems.

1.2 Rheology and viscoelasticity

When a living system is interacting with its environment, we should consider what formal definition of substrate mechanical properties is relevant to this interaction. The field of rheology studies the flow and deformation of matter, and it supplies the concepts needed for our current purposes. Macosko [6] is used as a reference text for definitions throughout this section. When interactions vary with time, one useful definition is the dynamic shear storage modulus, G', where "storage" refers to the stored energy in the deformed substrate. Generally speaking, G' is a measure of substrate stiffness. We can think of stiffness as a measure of how much force is required to deform a sample by a fixed amount – stiff materials require much force per unit deformation, and soft materials require only a little force to cause the same change in shape. In rheology, deformation is formalized with a dimensionless quantity called "strain," symbolized γ, which is the ratio of the shear displacement and the height of the sample. In turn, we measure the intensive quantity of "stress," or force per unit area,

symbolized σ, that is caused by this deformation. The shear stiffness G is defined in the static, equilibrium case as the ratio of stress to strain:

$$G \equiv \sigma/\gamma. \tag{1.1}$$

The quantity G' is called a "dynamic" storage modulus because it is measured under dynamic shear of the sample. We now define it alongside a complementary quantity, the dynamic shear loss modulus, G'', which measures the loss or viscous dissipation of energy in the deformed substrate. Specifically, a parallel plate rheometer is used to twist a cylindrical sample of material back and forth at a fixed frequency ω, as shown in Figure 1.1.

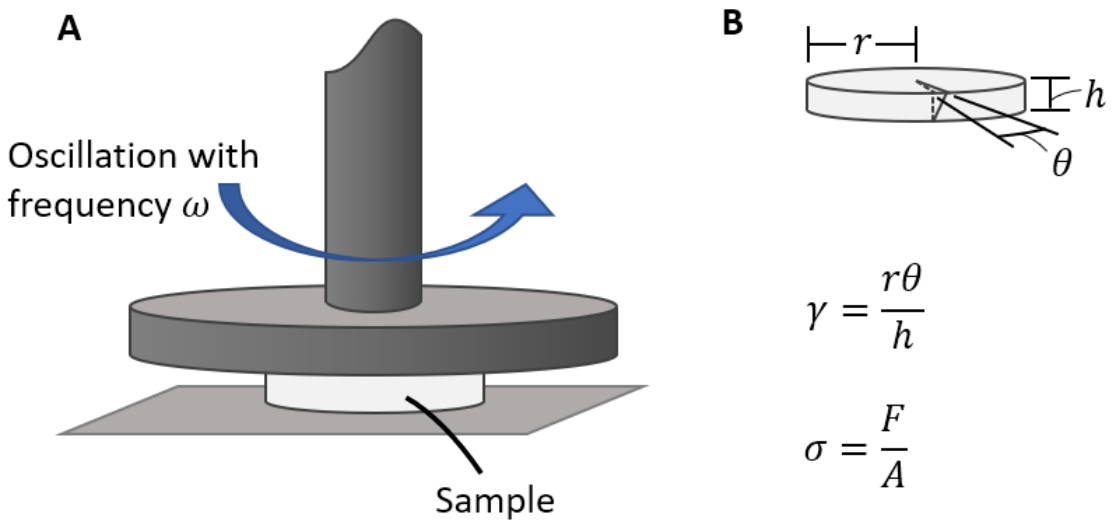


Figure 1.1. Basic shear rheology using cylindrical sample geometry. **(A)** A rheometer consists of a fixed lower plate and a rotating upper plate, with a sample sandwiched in between that has cross-sectional area A. The upper plate can be driven to twist the sample back and forth at a specific frequency ω. A force transducer measures the force F needed to induce this twisting, which can be used to calculate the material properties of the sample. **(B)** The shear deformation caused by twisting the sample depends on θ, the angular displacement of the top of the sample relative to the bottom, and the radius r and height h of the sample. These values are combined into a dimensionless strain γ, which along with the stress σ define the shear moduli of the sample.

The deformation exerted by the rheometer is described as a sinusoidal axial shear strain on the sample, and it is written with the equation

$$\gamma(t) = \gamma_0 \sin(\omega t). \tag{1.2}$$

When deforming a material, there are two useful cases to consider for the material's internal stress response: the purely elastic case and the purely viscous case. In the purely elastic case, the stress is precisely in-phase with the strain. That is, more deformation causes more internal forces at each instant. In this case, the stress is also sinusoidal:

$$\sigma_{elastic}(t) \propto sin(\omega t).$$

However, in the purely viscous case, the stress response depends on the strain *rate* according to Newton's law of viscosity. That is, only a *change* in deformation causes internal forces within a fluid. This derivative induces a 90° phase separation between stress and strain:

$$\sigma_{viscous}(t) \propto cos(\omega t).$$

The class of materials that fall in between these two idealized cases are often modelled as so-called "linear viscoelastic materials" because of their dual elastic (or solid-like) and viscous (or liquid-like) natures. This allows us to write the general stress response as a sum of two stress responses

$$\sigma(t) = \sigma_{elastic}(t) + \sigma_{viscous}(t).$$

When we divide this relation by the amplitude of the sinusoidal shear strain, we obtain the definitions of G' and G'' comparable to equation (1.1) as

$$\frac{\sigma(t)}{\gamma_0} = G'\sin(\omega t) + G''\cos(\omega t). \tag{1.3}$$

Adding the functions $\sin(\omega t)$ and $\cos(\omega t)$ with different positive weights will produce a new sinusoidal function with a phase shift δ between 0 and 90°, depending on the magnitude of the weights for each function. According to the trigonometric angle addition formula

$$\sin(\omega t + \delta) = \cos(\delta)\sin(\omega t) \; + \; \sin(\delta)\cos(\omega t),$$

we see that G' and G'' relate to the phase shift δ in the strain function $\sigma(t)$ as

$$\tan(\delta) \; = \; G''/G'.$$

As a rule of thumb, when $\tan(\delta) \geq 0.1$, a solid material has enough viscous dissipation to be considered "viscoelastic." We note here that both G' and G'' can vary depending on the frequency and amplitude of the strain being used to probe the sample. If a material under dynamic shear is shown to have increasing G' with increasing strain amplitude, the material is said to exhibit "shear thickening," a significant rheological characteristic, and one example of a nonlinear elastic response. Shear thickening can be demonstrated by biofilms such as those produced by *Pseudomonas aeruginosa* [7].

There are several basic rheological models that capture viscoelastic behavior. The Kelvin-Voigt model is a simple, two-parameter model for viscoelastic solids. It is often schematically represented as an idealized Hookean spring connected in parallel with an idealized Newtonian dashpot, as shown in Figure 1.2A.

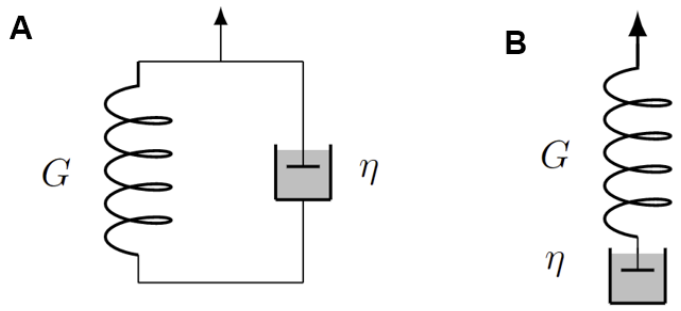


Figure 1.2. Basic viscoelastic model schematics. **(A)** Schematic of the Kelvin-Voigt model of a viscoelastic material under shear. The model combines a Hookean spring with shear modulus G in parallel with a Newtonian dashpot with viscosity η. **(B)** Schematic of the Maxwell model of a viscoelastic material under shear. This model instead combines the spring and dashpot in series, resulting in different model behaviors.

The spring (when considering shear deformation) has shear modulus $G = \sigma_s/\gamma$ in terms of the stress on the spring σ_s, and the dashpot has viscosity η, defined via $\sigma_d = \eta\dot{\gamma}$ in terms of the stress on the dashpot σ_d. Although this schematic is illustrated with axial compressive strain instead of shear strain, the mathematical representation is equivalent for the cylindrical torsion geometry we consider. Because the elements are connected in parallel, we have one common strain γ, and the overall stress $\sigma = \sigma_s + \sigma_d$, since the two forces add at a point of common cross-sectional area. This gives us the basic model equation

$$\sigma = G\gamma + \eta\dot{\gamma}. \tag{1.4}$$

Under sinusoidal strain, we can combine equations (1.2), (1.3), and (1.4) to show

$$G' = G$$

$$G'' = \eta\omega.$$

In this way, G' is considered a measure of how "solid-like" a material is, and G'' is considered a measure of how "liquid-like" it is. We can also see how a Kelvin-Voigt material would respond

to a creep test — that is, a rheological test involving the application of a constant stress. Assuming σ is a constant, we can solve equation (1.4) for $\gamma(t)$ and show that

$$\gamma(t) = \frac{\sigma}{G}(1 - e^{-\frac{G}{\eta}t}).$$

That is, a Kelvin-Voigt material subjected to a constant stress starts from zero strain and eventually saturates upwards to a maximum strain of σ/G along an inverted exponential decay curve. This is one reason the Kelvin-Voigt model is appropriate for viscoelastic solids, since viscoelastic liquids would instead flow indefinitely under a constant stress.

Another simple viscoelastic model is the Maxwell model, which instead combines the Hookean spring and Newtonian dashpot in series, as shown in Figure 1.2B. In this case, there is one common stress σ between the two elements due to Newton's third law of motion, setting

$$\sigma = G\gamma_s = \eta\dot{\gamma}_d, \tag{1.5}$$

where the spring and dashpot have their own separate strains, γ_s and γ_d respectively. These two strains sum to the overall strain of the material, giving $\gamma = \gamma_s + \gamma_d$. We can then use equation (1.5) to express the Maxwell model equation

$$\frac{\dot{\sigma}}{G} + \frac{\sigma}{\eta} = \dot{\gamma}. \tag{1.6}$$

When we submit this model equation to a creep test by setting σ equal to a constant, solving equation (1.6) for $\gamma(t)$ gives the result

$$\gamma(t) = \frac{\sigma}{\eta}t.$$

This shows that the Maxwell model is appropriate for viscoelastic fluids that deform indefinitely at a constant strain rate under the influence of a constant stress.

For either the Kelvin-Voigt or Maxwell models to capture nonlinear behavior, the model parameters G and η must vary with the frequency and/or amplitude of the shear strain.

Other viscoelastic models with more elements, such as the Jeffreys [8] or Burger models [9], have been used to model the viscoelasticity of bacterial biofilms. Other viscoelastic models, such as the generalized Maxwell model, combine multiple relaxation times explicitly by including many elements with a variety of fixed parameters [10]. Continuum models have also been used to capture viscoelasticity [11].

1.3 Bacteria as model systems

The motivations for studying bacteria in and of themselves are manifold. Bacteria are simultaneously one of the simplest and most abundant forms of life, entering every ecological niche where life is found [12], flourishing in the oceans, forming communities with other species, and coevolving with host organisms. This success, while being genetically much simpler than other organisms, posits bacteria as a unique window into the principles that living systems use to survive. The ubiquity of bacteria in natural and man-made environments, indeed even in our own bodies, allows them to fulfill both useful and harmful roles with enormous impact on daily life, with the biofuel industry, water treatment practices, antibiotic resistance and other bacterial diseases, as just a few examples. Greater understanding and control of these roles requires new studies into bacteria.

The perspective of physicists can provide new understanding at a pivotal time when novel quantitative methods are being deployed to study the complexities of living systems [13–15]. The forces that mediate physical interactions between bacterial colonies and the substrates they colonize are a promising object of study that is often not included in biophysical models. Bacteria adhere to and colonize a staggering variety of environments, from mucus and the soft tissues of the GI tract [16] to the hard keratinous shells of crustaceans [17], plastic waste [18], and stainless steel [19]. A focus on the softer environments that mimic biological tissue is especially relevant to understanding the role of bacteria in disease. For example, cystic fibrosis is a condition in which the viscosity of mucus is increased. This change results in chronic infections of *Pseudomonas aeruginosa* biofilms in the lungs of cystic fibrosis patients that are extremely difficult to treat but that occur only rarely in people with normal mucus viscosity [20].

Bacteria also provide insights into fundamental questions about living systems in general. They present a model system that is amenable to environmental manipulation and genetic manipulation using well-established techniques such as plasmid insertion [21]. The speed with which bacteria reproduce also allows for many experimental replicates to be performed in a short period of time. This provides enough data to resolve key effects from noisy biophysical processes.

1.4 Bacterial motility, force generation, and force sensing

Bacteria have evolved a number of different structures and appendages that allow individual cells to generate and sense forces. The classical flagellar motor, powered by proton motive

forces, is a bacterial analog of a mechanical motor, complete with rotor and stator components. It turns helical flagella to allow swimming motility in liquid environments. Pioneering work with *E. coli* bacteria that had their flagella tethered to a fixed substrate demonstrated that the flagellar motor operates at near constant torque [22,23]. More recent work shows that stator components are dynamically recruited to the flagellar motor in response to changes in mechanical load [24]. Transmembrane ion channels are ion pumps powered by adenosine triphosphate (ATP), an energy-storing molecule ubiquitous in living systems. These channels were found to allow bacteria to adapt to changes in osmotic pressure by actively forcing ions out of the cell. When a bacterium encounters an environment with very low salinity for example, this reduction in cellular Na^+ and Cl^- ions can prevent the osmosis that would otherwise swell the cell with water and cause it to burst [4]. Pili are an important bacterial appendage for generating force and cell motility at a surface. Twitching motility is powered by the extension and retraction of type IV pili, which can attach to a surface and pull cells along. Active assembly and disassembly of pilin subunits is also likely powered by ATP [25]. Internal cellular structures allow for gliding motility, a general term for smooth motility modes along surfaces that vary from species to species and are the subject of ongoing research. In *Myxococcus xanthus*, gliding motility involves the rotation of a cytoskeletal helix, which drives individual cells forward almost like a corkscrew tank. There is evidence that proton motive forces also power gliding motility [26].

Beyond individual cells, bacterial colonies collectively expand and adapt to environmental stimuli. Swarming motility is powered by the interaction of flagellar activity between cells in a dense swarm. This motility mode involves bacteria changing phenotype, becoming

hyperflagellated when conditions are right, such as when high concentrations of nutrients and moisture are present [27]. Motility modes such as spreading and darting are powered by the buildup of pressure from cell division and the manipulation of local surface properties by the secretion of extracellular polymeric substances (EPS) and surfactants [28]. In many species, a bacterial colony can also create a biofilm, a microenvironment created and inhabited by the bacteria that is ubiquitous in nature. The EPS-composed matrix that houses the bacteria has been shown to vary in stiffness and strength in differing environments [29]. When biofilms grow in flowing liquid, characteristic streamers form, resilient filaments of bacteria and EPS that adapt their material properties to have higher yield stress when flow is increased [7,30].

This evidence for bacterial response to mechanical signals raises the question as to what mechanism underlies these effects. Type IV pili are one of the most useful vectors of information for these signaling networks, bridging between a bacterium, its environment, and its neighbors. Some known signaling pathways involve type IV pili, such as the Chp system, which signals when a bacterium encounters a surface and begins the process of biofilm formation [31]. Because pili are dynamically assembled and disassembled from pilin subunits, the local density of those subunits inside the cell acts as a readout for how far the pilus is extended, giving the cell access to information about its mechanical microenvironment. The existence of other signaling pathways that can resolve more fine-grained mechanical information or other mechanisms of adaptation to mechanical signals is an open topic of research.

1.5 Biophysics of biofilms

To understand the growth and development of biofilms, we should consider the underlying physical mechanisms by which biofilms expand. When a few cells are introduced to a solid nutritive surface, the cells begin to multiply and divide as cellular behavior transitions from a planktonic, free-swimming state to a biofilm-forming state. While the nutrients that allow for cell division are necessary for a biofilm to grow, the proliferation of cells is not the only primary driving factor in biofilm growth. Although colonies of some bacterial species do grow exclusively by cell division (this passive motility mechanism – spreading – is explained in the previous section), biofilms expand much faster through other means. Mature biofilms are composed primarily of fluid, followed by extracellular matrix components excreted by the bacteria, and the bacterial cells themselves take up the smallest fraction of volume. Depending on environment, different factors can shape the developing biofilm morphology. In the case of biofilms grown in fluid flow, whether through large pipes or in microfluidic channels, polymers produced by the bacteria allow cells to adhere end-to-end, and as many thin filaments build up on each other, they elongate along the direction of flow, and form so-called “streamers” [7,30].

In this work, we consider in more depth the growth of biofilms on a solid interface, where fluid is available from the environment but is not being pushed by externally driven flow. In this case, biofilm growth occurs when the production of extracellular polymeric substances (EPS) induces an osmotic pressure gradient that causes the biofilm to swell with fluid and expand [32,33]. This pressure is generated by the high concentration at which the molecules of EPS are produced. Like water flowing spontaneously into a briny region enclosed by a permeable membrane, diffusion of water into the biofilm is driven by the tendency at equilibrium for the

concentration of EPS molecules to be uniform everywhere [33–35]. The EPS molecules are produced by the most metabolically active cells at the nutrient-rich edges of the colonized region [36,37]. This is illustrated schematically in Figure 1.3.

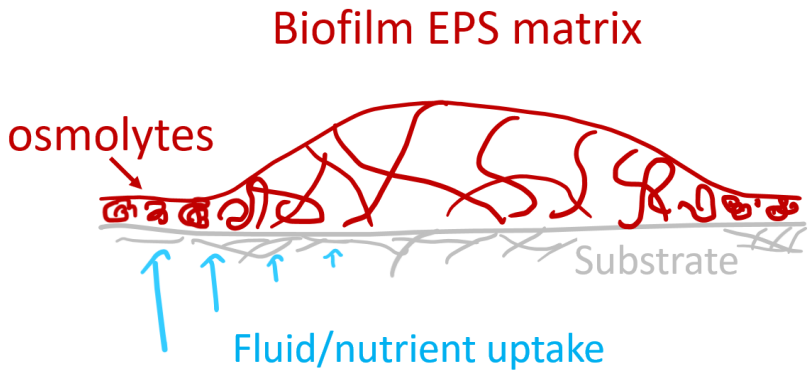


Figure 1.3. Schematic of biofilm components. As a biofilm grows, its structure is defined by the network of extracellular polymeric substances (EPS) produced by the bacteria. At the edges of the biofilm, metabolically active bacteria synthesize available nutrients into EPS, producing these molecules at high enough concentrations to drive fluid out of the substrate and into the biofilm via osmotic pressure. This osmotic pressure decreases as the biofilm swells with fluid, driving its outward expansion.

Because nutrient uptake powers the production of new osmolytes, these nutrients become locally depleted until the biofilm can colonize new, nutrient-rich locations. The concentration of osmolytes powers the fluid flow that swells the biofilm, and as fluid flows into the biofilm, the local osmotic pressure is decreased as the system attempts to reach equilibrium. This requires continuous production of osmolytes to power continued expansion. The interplay between these factors is captured by analytical models such as this one proposed by Mahadevan [37] that predicts three different phases in biofilm development.

Other relevant physical factors have been considered in biophysical models and experimental studies of biofilm expansion and development. For example, the surface tension of a substrate can be influenced by surfactants produced by bacteria. Surfactant-deficient mutants of

Pseudomonas aeruginosa produce biofilms with less structure [38], which impedes the flow of fluid within the biofilm and may inhibit their ability to colonize surfaces. Other examples include surface friction reducing biofilm area and increasing the slope of the leading edge of the colony [36], and increased bacterial adhesion causing cells to experience greater shear, which then express more of the signaling protein cyclic-di-GMP that upregulates biofilm formation. An overview of other relevant studies can be found in the introductory materials of [39]. Less explored is the interaction between the biofilm and the substrate it grows on, especially when that substrate is soft, although a body of literature in this line of questioning is beginning to emerge [39–41], which we contribute to with this work.

1.6 Robustness in collective bacterial behavior

The ability of bacteria to coordinate as a colony in order to survive depends on a complex network of chemical and mechanical signals sent between cells that regulate the expression of genes to modulate individual cellular behavior. Despite the many potential failing points of such an intricate network, these behaviors are consistently effective under a wide range of conditions. Biofilms resist environmental shear, the influence of antibiotics, and the response of immune systems. In *Myxococcus xanthus*, one mode of biofilm development is the fruiting body, an aggregate of EPS and cells, some of which undergo programmed cell death, and some of which differentiate into quiescent myxospores that can germinate and form a new colony under suitable conditions [42]. Evolutionary pressure has given rise to these behaviors, but what mechanisms bacteria employ for this robustness is an open question.

One hypothesis for this robustness that is addressed in Chapter 3 is the formation of redundant gene networks. In this scenario, proteins that perform a specific cellular function, although specialized, may be able to stand in for other proteins that are prevented from performing their function. One way that this may occur is via the creation of homologous genes, which are derived from identical copies of the same gene occasionally left behind by the apparatus of DNA repair processes when DNA breakage occurs. These identical copies, when present for long enough in the genome, can collect independent mutations over time, allowing them to specialize their function, but possibly leaving other fundamental behaviors intact. This form of redundancy therefore exists at the individual cellular level.

Another more general path to robustness is existence of multiple developmental paths by which a desired outcome may be achieved. The methods presented in Chapter 4 provide evidence that consistent fruiting body formation can be achieved with varying dynamics. This form of developmental redundancy exists at the colony level.

In general, multicellular development is driven by stochastic processes at all scales, from gene expression itself, to the cell-cell contacts that convey signaling information, to the mechanical interactions of neighboring cells encountering each other and their substrate. The fact that bacteria exhibit robust development in spite of this inherent stochasticity points to fundamental principles that await discovery with the proper tools. The work presented in this dissertation provides tools towards that purpose.

1.7 Primary Component Analysis (PCA)

One useful tool that can be used to decrease the complexity of high dimensional data while preserving its structure is Primary Component Analysis (PCA) [43]. This is a deterministic technique with no input parameters that is based on elementary linear algebra and statistics. This method begins with an input dataset, where each datapoint $\boldsymbol{x}$ is a vector in an N-dimensional vector space. PCA first identifies among all the directions in this space the single direction $\boldsymbol{a}$ that has the most variance across the dataset. This is equivalent to fitting the data to a line using the least squares method to minimize the error, where the line points along $\boldsymbol{a}$. This direction is called the first primary component, or PC1. The datapoints can be projected onto this line to obtain a one-dimensional version of the data. The data is now distributed along one axis, and the value of each datapoint along this axis is often referred to as the value of PC1. This process is visually summarized in Figure 1.4.

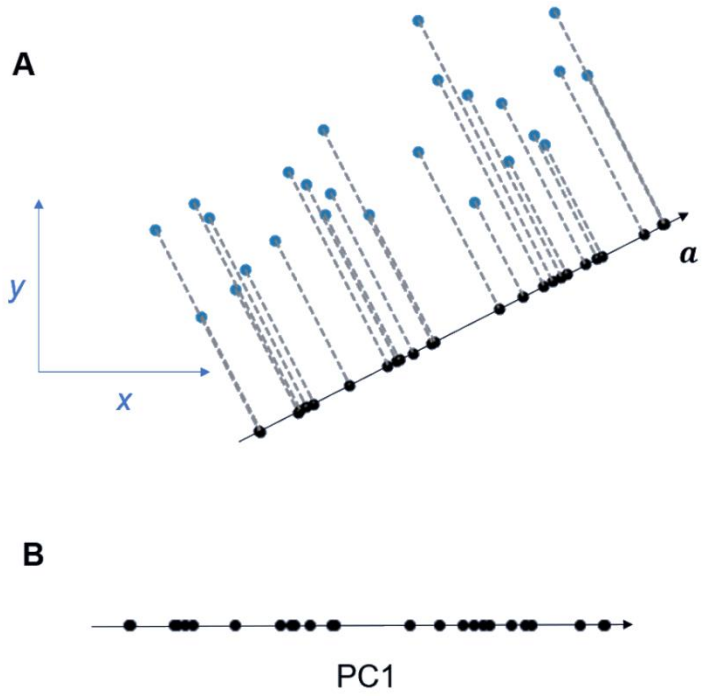


Figure 1.4. Dimensional reduction of a 2D dataset to one dimension using PCA. **(A)** From an initial 2D dataset in *x*-*y* space (blue points), PCA first identifies the direction $\boldsymbol{a}$ that maximizes the variance across the dataset, i.e. where the data is most widely distributed. The direction of maximum variance is called PC1, or the first primary component. Then the data points are projected onto this line. **(B)** The direction PC1 defines a new axis, and the value of the datapoints along this axis (which can be referred to as the values of PC1) constitutes the reduction of the data from two dimensions (*x* and *y*) to one dimension (PC1).

A reduction to one dimension often discards too much information, so more primary components can be identified. In the $(N-1)$-dimensional subspace of all directions perpendicular to PC1, the direction with the next most variance is called PC2. This process of identifying primary components can be continued for as along as desired up to N primary components, with each additional primary component adding more of the total variance but increasing the complexity of the reduced dataset, illustrated in Figure 1.5.

Because the overall structure of the data is changed in this process as little as possible, using PCA to reduce a dataset to a visualizable number of dimensions such as two or three can also have the effect of revealing correlations or clustering in multivariate data, important properties

of high-dimensional datasets that are often not immediately clear. The directions of the primary components themselves also reveal which or which combination of the original N dimensions contains the most variation and thus the most information across the dataset.

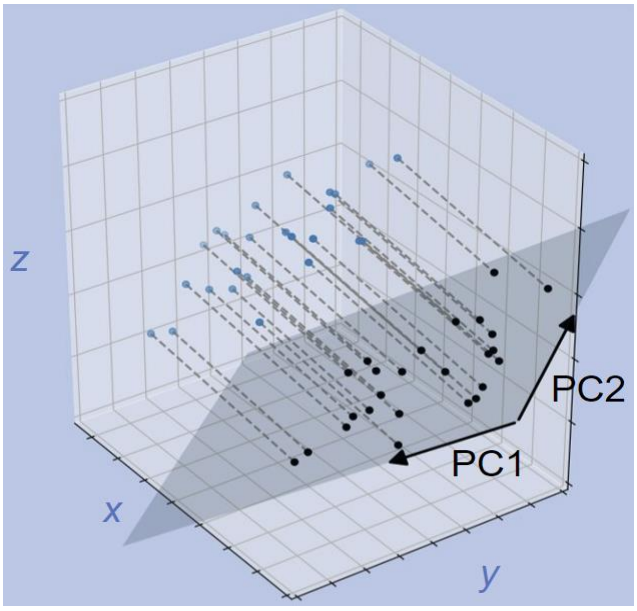


Figure 1.5. Dimensional reduction via Primary Component Analysis (PCA). An initial dataset (blue points) in *x*-*y*-*z* space is projected onto a two dimensional plane. This plane is defined by two directions: PC1, or the first primary component, is the direction of highest variance across the dataset, and PC2, or the second primary component, is the direction of second highest variance and is perpendicular to PC1. This is the orthogonal projection that preserves as much of the structure of the dataset as possible.

The primary components themselves are calculated as the eigenvectors of the covariance matrix $\boldsymbol{S}$ of the dataset, with the eigenvalues giving the variance of each. To demonstrate this, we first consider the definition of variance, using the value of the datapoint $\boldsymbol{x}$ projected onto the unit vector $\boldsymbol{a}$, that is $\boldsymbol{a} \cdot \boldsymbol{x} = a_i x_i$ (using the Einstein summation convention):

$$\begin{aligned} var(a_i x_i) &= \langle (a_i x_i)^2 \rangle - \langle a_i x_i \rangle^2 \\ &= a_i a_j \langle x_i x_j \rangle - a_i a_j \langle x_i \rangle \langle x_j \rangle \\ &= a_i a_j S_{ij} \end{aligned} \tag{1.6}$$

where the brackets $\langle \cdot \rangle$ denote an average across the datapoints, and we also obtain the definition of the covariance matrix $S_{ij} = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle$. To maximize the variance, we differentiate with respect to the direction $\boldsymbol{a}$, using a Lagrange multiplier λ to constrain $\boldsymbol{a}$ to be a unit vector, that is

$$a_i a_i = 1. \tag{1.7}$$

Doing this, we then obtain the condition

$$\begin{aligned} \partial_{a_i}[a_i a_j S_{ij} - \lambda(a_i a_i - 1)] &= 0 \\ S_{ij} a_j - \lambda a_i &= 0 \\ S_{ij} a_j &= \lambda a_i, \end{aligned} \tag{1.8}$$

showing that any direction $\boldsymbol{a}$ that is a local extremum of variance is an eigenvector of $\boldsymbol{S}$ with eigenvalue λ. We interpret the eigenvalue by combining equations (1.6), (1.7) and (1.8):

$$\begin{aligned} var(a_i x_i) &= a_i a_j S_{ij} \\ &= a_i \lambda a_i = \lambda. \end{aligned}$$

This shows that each primary component is an eigenvector of $\boldsymbol{S}$, ordered by the variance given by their respective eigenvalues. Since the covariance matrix is symmetric, these eigenvectors are all perpendicular to each other.

Because the calculation of primary components involves foundational techniques of linear algebra, implementations of PCA are found in the standard libraries of many programming languages, such as MATLAB and Numpy for Python.

1.8 Outline

This body of work explores the biophysical questions of collective bacterial motion by presenting new experimental evidence and analytical techniques. Chapter 2 features an experimental study of biofilms produced by multiple bacterial species that reveals a previously unobserved effect wherein bacteria can generate greater forces and colonize stiffer substrates more quickly than softer substrates in a particular window of stiffnesses that are relevant to biofilm formation on biological tissues. Chapter 3 begins the focus on a particular organism, *Myxococcus xanthus*, and displays a technique for quantitative phenotyping of a characteristic multicellular developmental behavior expressed by this species. This technique is used to test long-standing hypotheses in prokaryotic genetics and measure the breadth of networks of redundant genes. Chapter 4 develops these analysis techniques further and also presents a high-throughput experimental system for future experiments, including a measure of the variation of wild-type developmental phenotypes, and validation with subtle mutant phenotypes that are difficult to measure with standard statistical techniques. In the concluding chapter we recapitulate the findings presented thus far and survey possible future experiments.

2. Bacterial colony growth increases with substrate stiffness

This chapter is based on the article "Spreading rates of bacterial colonies depend on substrate stiffness and permeability" previously published in 2022 in PNAS Nexus and coauthored by myself, Alison Patteson, Minh Tri Ho Thanh, Arvind Gopinath, Danielle Germann, Robert Carroll, Alana Franceski, and Roy Welch. The experiments were designed by Alison Patteson and myself. I performed the experimental work and analysis. The manuscript was written by Alison Patteson, Minh Tri Ho Thanh, who contributed to the section on traction force microscopy, Arvind Gopinath, who contributed analysis of gel permeability, and myself. Minh Tri Ho Thanh and Danielle Germann contributed to traction force microscopy experiments and analysis. Alana Franceski and Roy Welch assisted in experimental work with M. xanthus. Robert Carroll contributed rheological characterization of the polyacrylamide hydrogels and agar.

2.1 Introduction

Biofilm formation is an important process in the bacterial lifecycle. Biofilms are multi-cellular communities of bacteria commonly attached to an external surface [44,45]. Emerging evidence indicates that bacteria sense and respond to variations in the mechanical properties of the surrounding environment, resulting in changes to cell physiology and biofilm morphology [46–49]. When a bacterium makes contact with a surface, it initiates a program of gene expression that promotes colonization and secretion of extracellular polymeric substances that self-encapsulate the cells and gives the biofilm its structure [50,51]. The biofilm thus consists of both cells and EPS components, growing as a result of both cell division and EPS deposition [38]. Colony growth is aided by the production of surfactants [38] and EPS-generated osmotic pressure gradients, which facilitate nutrient uptake from the substrate [32,33]. Thus, the physical properties of the underlying substrate have the potential to disrupt structural and functional aspects of cell attachment and function that contribute to biofilm phenotypes.

The vast majority of biofilm experiments are conducted on the surface of an agar gel. Agar was introduced in 1882 by Angelina Fanny Hesse and gained popularity through Robert Koch [52] because it is inert to bacteria degradation. However, agar is isolated from marine algae and is an undefined media, as its chemical composition is not entirely known [53]. Agar variability from the isolation process makes it difficult to define and reproduce its chemical and physical properties [54,55].

A common feature of these studies is that biofilm expansion decreases with increasing agar concentration [33,49,56,57]. Increasing agar concentration increases agar network stiffness but

also impacts other properties of the gel, such as the hydrogel pore size [33]. Agar is typically prepared in the range 0.5% - 2% agar in a nutrient-rich media, forming a hydrogel comprised of a porous solid network and the nutrient-rich interstitial fluid that permeates through the network. On stiff agar, the pore size is smaller and the rate of nutrient transport through the substrate and to the biofilm decreases [33,37]. A number of studies have attributed this inhibited biofilm growth on stiff agar to lack of nutrients rather than stiffness per se [33,37]. On the other hand, there are studies indicating substrate stiffness can separately modify biofilm shape and expansion by mediating adhesion [39,58] and frictional forces between the biofilm and the substrate [36]. The extent to which biofilm growth depends on the combined effects of substrate stiffness and nutrient availability is thus an open question, and current bacteria culture substrates largely cannot separate the effects of these two properties on biofilm growth.

Here we report the development of polyacrylamide (PAA) hydrogels with tunable matrix stiffness and matrix porosity to determine their integrated effects on biofilm growth. We identify a new regime in the limit of purely elastic substrates in which bacteria colonies spread out faster on stiffer substrates compared to softer ones, which is opposite of conventional agar. Our study focuses on the bacterium *Serratia marcescens*, which is a common model organism for collective motion and behavior [48,59–61], but we also show that *Pseudomonas aeruginosa*, *Proteus mirabilis*, and *Myxococcus xanthus* expand faster on stiffer substrates than soft ones. A major advantage of polyacrylamide gels is that unlike agar they linearly deform in response to a wide range of stress, which enables facile definable force calculations. Using traction force microscopy-based techniques, we show that bacteria colonies generate transient surface forces

correlated over length scales much larger than a single bacterium and that the magnitude of these forces increases with increasing substrate stiffness. Our results are consistent with a model in which biofilm development is impacted by osmotic pressure gradients between the biofilm and the substrate and the substrate's poroelastic response.

2.2 Results

Design and characterization of polyacrylamide hydrogels

In this study, we used both polyacrylamide gels and conventional agar as a point of comparison. To characterize the mechanical properties of the gels, we measured under shear their elastic storage modulus G', which quantifies their resistance to shear deformations, and their viscous loss modulus G'', which quantifies viscous energy dissipation, with an oscillatory rheometer. As shown in Fig. 2.1a, agar exhibits non-linear shear softening; its shear modulus decreases from approximately 10 to 1 kPa as shear strain rises from 2 to 50%. The mechanical response of PAA to shear strain differs from agar. As shown in Fig. 2.1b, polyacrylamide gels form linearly elastic gels, with near constant G' over the applied strain range.

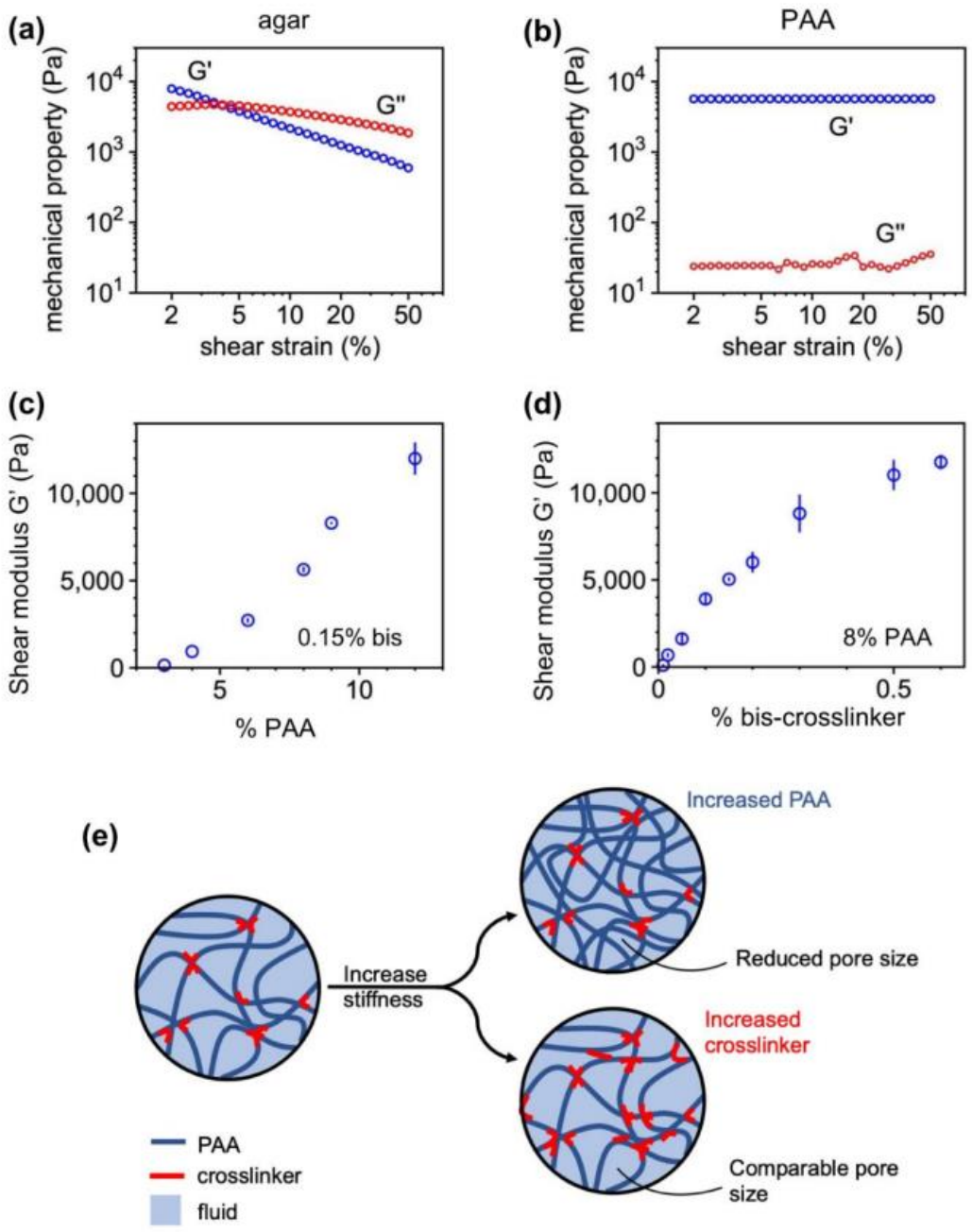


Figure 2.1. Substrate characterization using a stress-controlled rheometer. The shear storage modulus G' and loss modulus G'' as a function of shear strain for both **(a)** agar and **(b)** polyacrylamide gels. PAA gels are prepared by either **(c)** increasing PAA concentration or **(d)** chemical crosslinker bis-acrylamide. **(e)** Schematic representation illustrating the effects of increasing PAA polymer concentration vs chemical crosslinker on fluid permeability in the network. Error bars denote standard error from 3+ independent trials per condition.

The shear modulus G' sets the extent to which a material deforms under an applied shear stress. The non-linear shear softening is a property of complex materials and demonstrates that

agar is softer when probed at higher deformations compared to small ones. If biofilms deform their substrate at magnitudes that vary over time or under different experimental conditions, then the shear modulus of the agar substrate will vary in response to the applied deformation and the biofilm would experience a different mechanical resistance from the substrate.

Agar also exhibits significant viscoelasticity, with a viscous loss modulus G'' of 10%-50% of the storage modulus G' at least for small strain values (2-5%) at a frequency of 1 Hz. This data suggests that as a substrate for biofilm growth, agar dissipates energy and relaxes applied stresses that might be relevant to outward growth of the colony. PAA gels, in contrast, exhibit negligible viscous dissipation, consistent with prior work [62–64].

Unlike agar and most other bacterial growth substrates, the shear (elastic) moduli of PAA gels can be tuned by either crosslinker concentration or polymer concentration, which allows tunable control of matrix stiffness and matrix pore size [65]. In order to distinguish between the effects of substrate stiffness and substrate permeability on biofilm growth, we thus designed PAA gels (Supp. Table 2.2) with shear moduli G' ranging from 100-10,000 Pa by varying either the amount of acrylamide or the amount of the chemical crosslinker bis (Fig. 2.1c&d). These two parameters, acrylamide concentration and crosslinker concentration, have two different effects on network permeability (Fig. 2.1e) (28). Increasing the concentration of acrylamide monomer results in a denser, stiffer polyacrylamide network with a smaller pore size and thus lower permeability. Increasing the concentration of crosslinker links together the same density of polyacrylamide polymers at a greater number of sites, increasing the network stiffness without significantly changing pore size. These effects are illustrated schematically in Fig. 2.1e. We confirmed these expectations using effective diffusion and force indentation experiments

to estimate the effective pore sizes of the polyacrylamide gels (Table 2.1, Supp. Fig. 2.8&9). Here, we note here that diffusion of nutrients through the network depends on both molecular diffusion and poroelastic transport of solvent as gels swell or deswell. We refer to nutrient transport and diffusivity in terms of effective diffusivities that combine the effects of both. The effectivity diffusivities measured here range from approximately 70-175 $\mu m^2/s$, which is consistent with prior literature values for PAA gels [66] accounting for differences in the shear modulus G' of the gels [67].

% PAA	% Bis	Effective Diffusivity ($\mu m^2/s$)	Pore size (nm)
3	0.15	170±22	22 ± 5.6
12	0.15	80±20	0.85 ± 0.1
8	0.085	75±10	1.5 ± 0.1
8	0.45	70±10	0.9 ± 0.2

Table 2.1. Effective pore size measurements of polyacrylamide gels (Details in Supplementary Materials).

Substrate stiffness increases biofilm expansion rates

Our experimental protocol consists of directly observing the growth of Serratia marcescens colonies on the surface of hydrogel substrates with time-lapse microscopy (Methods). Before inoculation, the PAA gels are soaked multiple times in LB nutrient-rich broth. We deposit a small inoculum of bacteria on the gel surfaces and track x-y positions of the resulting biofilm boundary as it expands over 15-hour time periods, relevant to prior literature reports [36,68] (Fig. 2.2). The biofilm boundary is tracked by a custom semi-automated Python script we developed for these videos (Methods, Supp. Fig. 2.10).

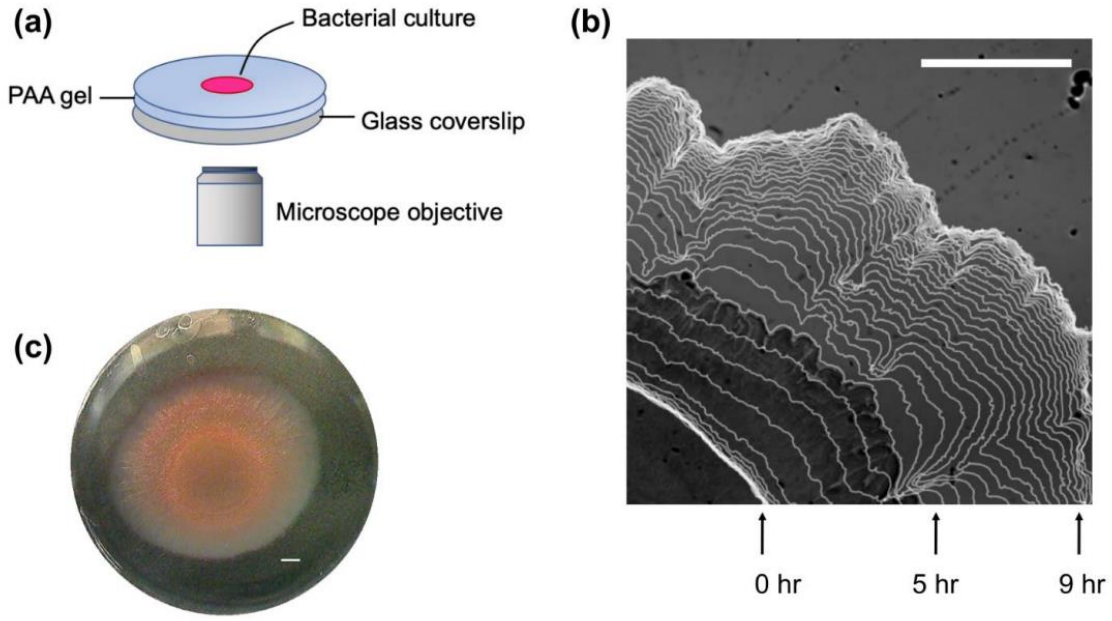


Figure 2.2. (a) Schematic diagram of the experimental setup. Aliquots of Serratia marcescens were placed on polyacrylamide (PAA) substrates that were 0.8 mm in height. The substates were maintained at 37 °C in a humid stage top incubator. **(b)** Visualization of the growing biofilm boundary overlaid on a sample image of the biofilm. Images were acquired at 10-minute increments, and boundaries shown here are displayed at 20-minute increments. **(c)** Color image of full bacterial colony after 15 hours of growth. Scale bars, 1mm.

Representative biofilm time lapse images on a soft (G' = 0.9 kPa) and a stiff (G' = 3 kPa) PAA gel are shown in Fig. 2.3, with videos available as supplementary materials (Video S2&S3 respectively). There are notable differences in colony morphology and the collective cell migration speed between the two gel types. While the biofilm surface expansion speed encompasses the collective effects of rates of EPS production, cell division, and cell surface motility, the colony expansion rate is faster on the stiffer PAA gel compared to the softer one, opposite of the behavior on conventional agar substrates.

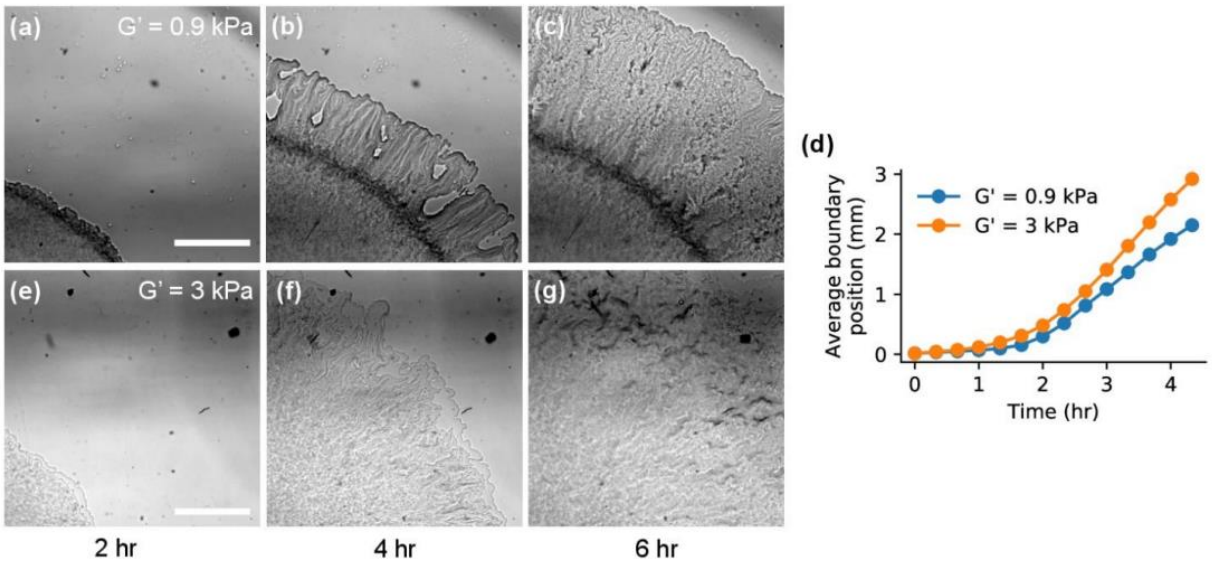


Figure 2.3. Representative bright field images of Serratia marcescens colonies growing across a soft **(a-c)** (SI Video 1) and stiff **(e-g)** (SI Video 2) PAA gel. Variations in gray intensity correlate with the amount of transmitted light though the biofilm, which depends upon both colony density and height. Both the biofilm structure and expansion rate **(d)** depend on substrate stiffness. Scale bar, 1 mm. Soft gel: G'=0.9 kPa, 4% PAA, 0.15% bis-crosslinker. Stiff gel: G'=3 kPa, 6% PAA, 0.15% bis-crosslinker.

A central feature of biofilm formation is the production of extracellular polymeric substances (EPS), which adheres cells to each other and external surfaces. EPS production allows for vertical growth of the colony [69] and also mediates osmotic spreading of the colony edge [32,37]. To determine whether the bacteria colonies were producing EPS, we stained the colonies with a fluorescent biofilm matrix stain and found EPS deposition throughout the colony (Supp. Fig. 2.11). We also observed wrinkles and surface corrugations on the colony surface, characteristic of EPS production and biofilm formation. To visualize the 3D colony structure, we used a white-light interferometer to map the 3D colony verticalization (Supp. Fig. 2.12). The colonies on soft substrates were more vertical than colonies on stiff substrates, with colony heights of approximately 25 µm on soft substrates compared to 5 µm on stiff substrates. Compared to 3D imaging methods such as confocal microscopy or white light interferometry,

wide-field imaging of the colonies can be gathered in larger numbers with an automated multi-point microscope. Thus, here, we focus on the 2D colony expansion rates as a high throughput metric to screen the effects of substrate stiffness on colony surface dispersal.

This phenomenon is highlighted in Figure 2.4, which shows snapshots of whole biofilm colonies on PAA gels and agar substrates taken several hours after inoculation. We note that the reduced biofilm growth on stiff agar substrates compared to soft agar is a common feature of many different bacteria species, such as *Vibrio cholerae*, *Proteus mirabilis*, *Myxococcus xanthus*, and *Salmonella enterica* [33,49,56,57]. This inhibited biofilm growth on agar has been attributed to the reduction in substrate permeability found in more concentrated agar substrates, which limits the transport of fluid and nutrients from the substrate into the biofilm [33]. Another physical factor contributing to biofilm expansion rates is the surface tension between the biofilm and the surface, where decreasing surface tension increases biofilm expansion, by allowing the leading edge to propagate and advance faster [70]. Therefore, we measured the surface tension between a fluid droplet and the hydrogel surfaces with a contact angle goniometer (Supp. Table 2.3). We found that the contact angle increased from 15° ± 2° to 23.5° ± 1° for 4% and 8% PAA, respectively, suggesting that the effects of surface tension would lead to increased biofilm expansion on softer PAA gels. The strong increase in biofilm expansion on more concentrated PAA gels is thus unexpected from the effects of the hydrogel itself on surface tension (Fig. 2.3&4). We note that the colony expansion rates here are significantly slower than swarming expansion rates and no vortical collective flows are observed, which indicate that the colony is in a biofilm state in contrast to a swarming

state [37,71]. Cell motility, however, likely does contribute to the expansion process, as some subfraction of cells in a biofilm maintain a motile state [72].

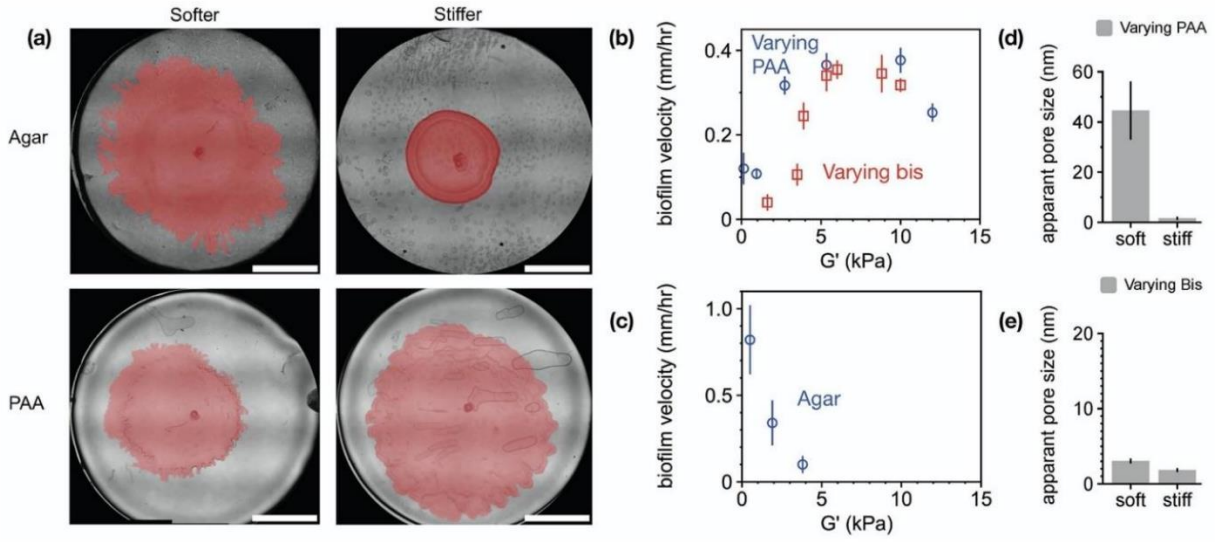


Figure 2.4. (a) Representative whole-colony pictures of Serratia marcescens grown on soft and stiff agar and PAA substrates. While colonies size decreases on stiffer agar substrates, the opposite occurs on purely elastic PAA substrates. The biofilms are manually traced and pseudo-colored pink to enhance imaging contrast. Scale bar, 5 mm. **(b&c)** Biofilm expansion velocity as a function of substrate stiffness (G') for Serratia marcescens colonies grown on **(b)** PAA and **(c)** agar substrates, measured approximately two hours post inoculation. The varying PAA data is gathered at 0.15% Bis; the varying bis data, at 8% PAA. A full list of the hydrogel compositions are presented in Supp. Table 2.3. Data points represent n ≥ 8 measurements taken in N = 3 independent trials. Estimates of the apparent pore size for gels comprised of **(d)** varying PAA and **(e)** varying bis-crosslinker concentration (SI). The soft varying PAA gels have the largest pore size.

To quantify the above observations, we calculate the initial biofilm expansion rates by calculating the boundary velocities from the tracking data. The biofilm velocity is defined here as the average radial displacement of the biofilm boundary over a time interval of Δt = 20 min. Here, we use the biofilm boundary velocity as a metric of wild-type collective expansion, not a direct measure of single cell surface motility or bacteria doubling time.

Figure 2.4 shows the initial colony expansion velocities on PAA and agar substrates when radial expansion of the biofilms are first beginning to be observed, approximately two hours post inoculation.

Figure 2.4A shows the surface expansion rate for polyacrylamide hydrogels of increasing PAA concentration. We find that there is a significant increase in colony expansion rate with increasing substrate stiffness, particularly for substrate stiffness G' < 5 kPa. For substrate stiffness greater than 5 kPa, the colony velocity seems to saturate with substrate stiffness and then begins to slowly decline on PAA gels (Fig. 2.4a). These results are strikingly similar for PAA gels with the same range of substrate stiffness but prepared by increasing bis-crosslinker (Fig. 2.4B). We note that Fig. 2.4b serves as a control. Unlike increases in PAA, increasing the bis crosslinker does not significantly modify the network pore size, indicating a distinct effect of substrate stiffness on colony expansion. These results are entirely different from biofilm growth on agar substrates: the colony velocity decreases dramatically with increased agar concentration (Fig. 2.4c), even for substrate stiffness less than 5 kPa.

We note that we do observe differences between hydrogels prepared by varying PAA and varying bis of the same hydrogel stiffness G'. These differences are most evident for soft gels, where G' < 5 kPa (Fig. 2.4a&b): colonies on the varying PAA gels expand more quickly than colonies on the varying bis gels ($p < 0.01$ for G' ≈ 3kPa). In this regime, the varying PAA gels have larger pore sizes (22 nm) than the varying bis gels (1.5 nm) (Table 2.1, Supp. Table 2.4). This result is thus consistent with the idea that larger network pore sizes increase colony expansion rate, allowing more nutrient-rich fluid to flow from the substrate into the colony. For G' < 5 kPa, the expansion rates saturate to approximately the same magnitude, 0.3 mm/hr, for each case.

Taking into account the diverse bacterial strains that colonize agar, we selected three additional bacterial species to test on PAA gels: *Pseudomonas aeruginosa*, *Proteus mirabilis*, and

Myxococcus xanthus (Methods). Given the strong effect of substrate stiffness on *Serratia marcescens* surface expansion (Fig. 2.4), we selected a soft (G' ≈ 0.5 kPa) and stiff (G' ≈ 5 kPa) PAA gel to culture these three species (Fig. 2.5). In each case, we found that biofilm expansion was faster on stiffer PAA than softer PAA (Methods). *Pseudomonas aeruginosa* and *Proteus mirabilis* are both gram-negative bacteria known to cause disease in humans. Here, we use *Pseudomonas aeruginosa* Xen05, which is derived from a human septicemia isolate, and *Proteus mirabilis* BB2000. *Proteus mirabilis* are well-known for their ability to swarm, a flagella-based rapid surface motility mode, which they are capable of doing over a striking range of surfaces [73]. *Myxococcus xanthus*, a member of the δ-Proteobacteria, displays a wide range of multicellular emergent behaviors [42,74]. *M. xanthus* have two well-characterized motility modes, social (S)-motility powered by type IV pili [75] and adventurous (A)-motility, powered by an inner membrane motor that applies force to the substrate at adhesions [26,76]; they do not have flagella. While *M. xanthus* is well-known for its display of dynamic fruiting body formation when starved [42,74], here we focus on its collective biofilm expansion in growth media. Given the different motility modes of three different bacteria species, our results suggest that for polyacrylamide hydrogels increasing biofilm expansion rates on substrates with increasing stiffness is a more general phenomenon and is not unique to *Serratia marcescens*.

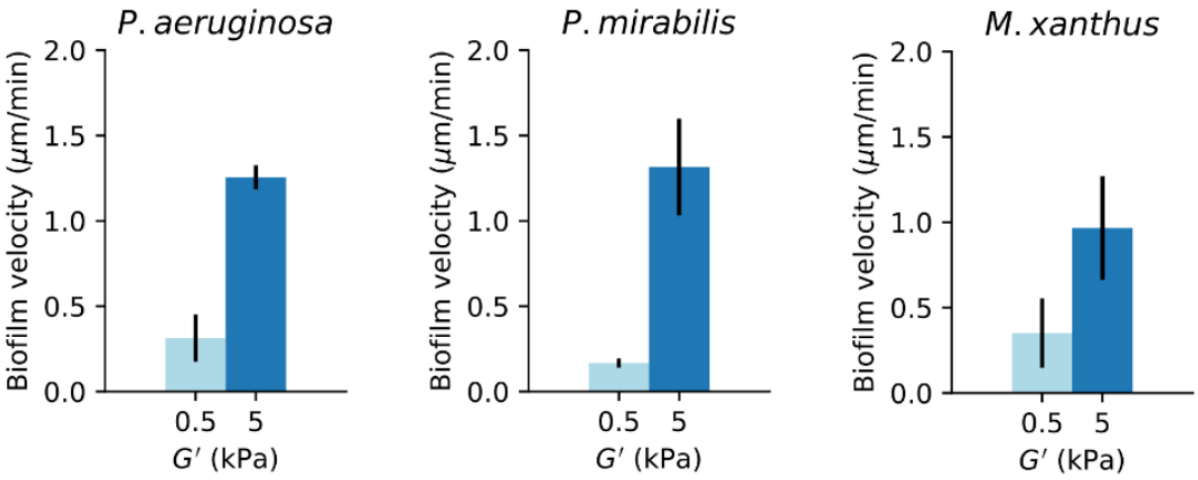


Figure 2.5. Biofilm expansion rates on soft (G'=0.5 kPa) and stiff (G'=5 kPa) PAA gels for *P. aeruginosa*, *P. mirabilis*, and *M. xanthus* bacteria species. Data reflects n ≥ 8 measurements over N = 2 independent trials. Error bars denote standard error.

Biofilm force generation and associated substrate deformations

The observed increase in biofilm edge velocity with substrate stiffness might be surprising given that biofilm expansion rates decrease on substrates of increasing agar concentration. Increasing agar concentration has the combined effect of increasing substrate stiffness, the viscous loss modulus G'', and decreasing substrate permeability, which hinders the flow of nutrients to the biofilm. Thus, the effects of stiffness cannot be unambiguously related to colony expansion rates.

What may cause substrate elasticity to increase biofilm expansion rates on polyacrylamide gels? To better understand the observed enhancement in biofilm expansion with increasing substrate stiffness, we performed experiments in which biofilm-generated substrate displacements could be directly visualized via traction force microscopy-based techniques (Fig.

2.6). PAA deforms in proportion to applied forces and recovers completely and instantaneously on the release of the force. The displacements of PAA substrates are thus related to the surface stress, and the surface stress can be reconstructed from the displacement fields based on the theory of linear elastostatics [77–79]. However, if the substrate is non-linear or viscoelastic (such as agar), then the relationship between stress and strain is much more complicated and time-dependent, and the substrate stress cannot be directly reconstructed from the substrate displacements.

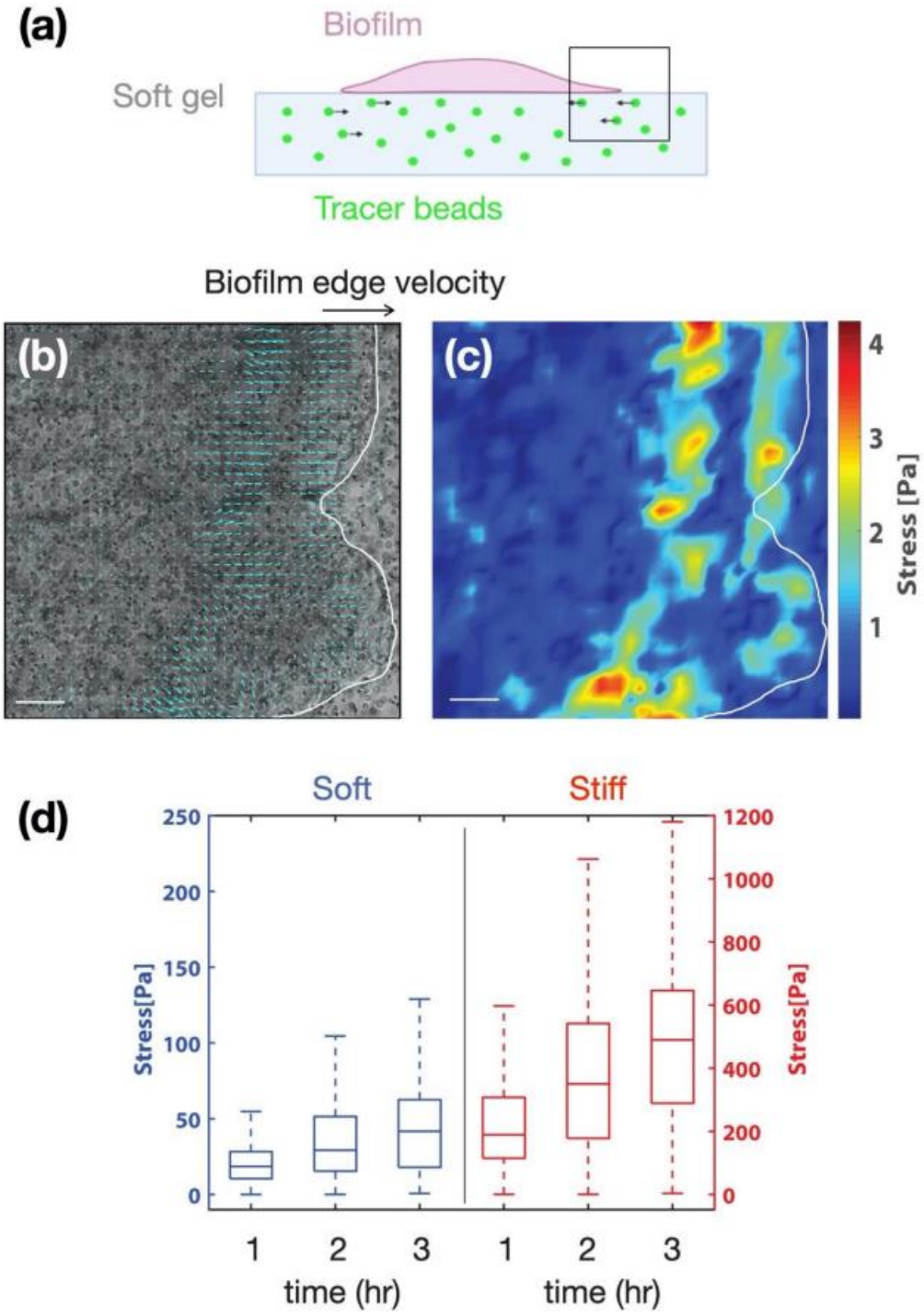


Figure 2.6. *Serratia marcescens* generate more force on stiffer substrates. **(a)** Schematic of the beads' displacement underneath the biofilm during expansion. **(b)** Instantaneous displacement field (blue arrows) and **(c)** stress map of two consecutive frames during biofilm expansion. There exists a small but significant contraction towards the edge of the biofilm in the direction of the expansion. Scale bar, 200 µm. **(d)** Box-and-whisker plots of total accumulated traction stress applied by *S. marcescens* on soft and stiff polyacrylamide gels over three hours. Data obtained from a minimum of 7 samples from at least 2 independent trials per condition.

To determine whether substrate stiffness impacted the colony's ability to generate surface forces, we used traction force microscopy (TFM) based techniques to measure the surface stress exerted on the substrate by the expanding biofilms (Fig. 2.6&7). To visualize the deformations of the substrate displacement, the polyacrylamide gels were embedded with 4.8 μm fluorescence beads, which were tracked over time. Using this technique, we observed two main types of substrate displacement. The first type occurred in the vicinity of the expanding edge of the colony as transient localized hot spots on the scale of 20 μm, much larger than an individual bacterium (Fig. 2.6c&d). These localized regions were reminiscent of traction hotspots observed generated by collective motion *Myxococcus xanthus* cells [80]. Here, these transient localized pulses were more evident on soft substrates (G' = 0.5kPa) for *Serratia marcescens* colonies than on stiff ones (G' = 5 kPa). In addition to these hot spots, we observed a slower – but more consistent – inward motion of the beads toward the center of the colony (Fig. 2.6d&f, Fig. 2.7), consistent with the build-up of a bulk inward contractile force [63]. In some of the TFM-based experiments, the fluorescence beads are applied only to the surface of the gel to more precisely track motion only at the surface of the gel. The fluorescence particles typically remained in focus throughout the entire experiment, suggesting minimal z-displacements. Assuming perfect focus at the start of the experiment, then the particles are displaced from center focus to half the depth of field, or 4.5 μm (0.5 x 8.5 μm) for the 10x objective used in these experiments, which is consistent with vertical substrate deformations on the order of 1-10 μm observed for *Vibrio cholerae* and *Pseudomonas aeruginosa* biofilms [81].

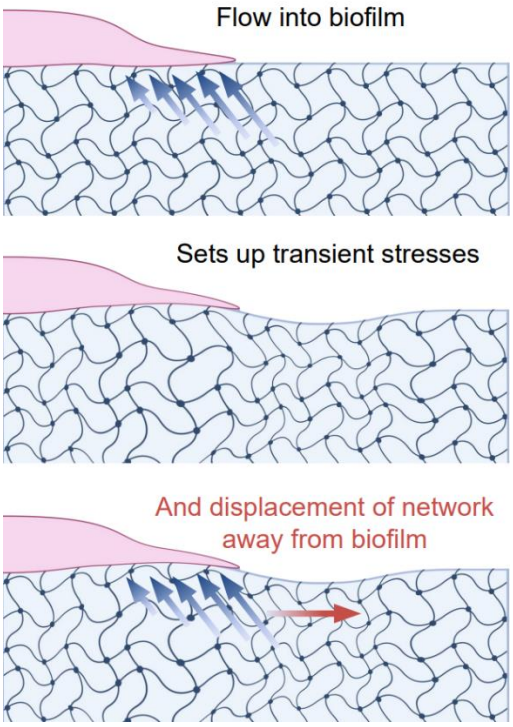


Figure 2.7. Schematic representation of poroelastic stresses associated with a growing biofilm front.

To estimate the surface forces exerted by the colony on the substrate, the surface stress was reconstructed from the tracer displacement maps via the finite element method [82–84] (Methods). The surface stress increased over time, and – surprisingly - the surface stress was 10-fold higher for colonies on stiff substrates compared to soft ones (Fig. 2.6d): the average surface stress was approximately 500 Pa on the stiff gel compared to 40 Pa on the soft gel (3 hr time point). Typical surface strains ϵ also increased for biofilms on stiff substrates compared to soft ones, with surface strains at approximately 2% on stiff substrates compared to 0.5% on soft substrates (Methods). The large difference in bacteria force generation on soft versus stiff substrates indicates a strong role of substrate stiffness on biofilm expansion and biofilm force generation.

To interpret these results (Fig. 2.4, 2.6, and 2.7), we suggest a minimal model that treats substrate deformations as a signature of poroelastic stresses in the network driven by osmotic pressure differences across the growing biofilm front. In this picture, the hydrogel substrate behaves as a poroelastic material permeated by nutrient fluid that can move relative to this network. Biofilm growth proceeds as bacteria divide and begin to excrete extracellular polymers. Before these polymers assemble into the extracellular matrix network, they act as osmolytes that set up an osmotic pressure difference between the biofilm and the substrate [32,37,56]. Gradients in osmotic pressure draw up fluid and nutrients into the biofilm, which allows the biofilm front to grow and expand. Osmotic spreading of biofilms was first observed for bacteria on agar substrates [32,33]. As seen on agar, decreasing network pore size reduces fluid permeability and diminishes colony expansion. The detailed 3D flows on agar have not been fully resolved, and we note that on viscoelastic substrates, such as agar, understanding the flows and stresses in the network are complicated by the non-linear mechanics and viscous dissipation that alleviate stresses over time.

Here we use polyacrylamide gels without viscous dissipation and tunable pore sizes to quantify biofilm expansion rates. Our results suggest that the larger stresses induced on stiffer substrates provides higher nutrient fluid flows that induces higher rates of biofilm growth. Motivated by our findings and the results of others [37,81], we propose that this fluid flow sets up transient stresses in the substrate network, which could drive substrate displacements in regions surrounding the biofilm. In a poroelastic material, fluid flows and network deformations are coupled. This is shown schematically in Fig. 2.6b, in which vertical indentations of the substrate are exaggerated to illustrate the effect of the colony osmotic pressure on the

substrate. Based on our experimental observations (Fig. 2.4&6), we infer that stiffer substrate networks more efficiently couple with fluid flows, increasing transmission of forces through the network and driving enhanced transport of the fluid through the network. This is shown schematically in Figure 2.7. On soft substrates, in contrast, local strains decay faster, resulting in reduced propagation and transmission of stress. The flow of fluid to relax the applied stress is thus localized to smaller regions resulting in reduced fluid and nutrient flux. In this way, substrate network stiffness may act to increase initial biofilm growth rates. If the substrate is viscoelastic, such as agar, then viscous stress dissipation might further reduce flow. Taken together, our experiments highlight complementary roles played by fluid flows and network strength properties of substrates on which biofilms growth. For growing Serratia marcescens colonies, increased substrate stiffness enhances biofilm growth rates.

2.3 Discussion

Bacteria are capable of transducing mechanical signals from their environment and responding to those cues [40,85,86], but the precise mechanisms remain largely unclear. In this article, we investigated the effects of substrate material properties on the biofilm expansion of *Serratia marcescens*. Using polyacrylamide hydrogels of varying composition, we found that substrate stiffness and porosity tune the spread of growing biofilm colonies. Our results indicated that increasing substrate stiffness enhances biofilm expansion rates in the limit of purely elastic substrates, unlike conventional agar substrates.

Taken together, our results suggest that substrate stiffness and substrate pore size have two different effects on colony expansion. Increasing pore size enhances biofilm expansion (Fig.

2.4). This result is largely expected as a larger network pore sizes allow for enhanced diffusion and flow of nutrients from the substrate into the biofilm. An unexpected finding here is that substrate stiffness can have as big an impact on biofilm expansion rates as network pore size, and increasing substrate stiffness increases colony expansion (Fig. 2.4), even when network pore size is controlled and accounted for.

In the case of agar, substrate stiffness and pore size are coupled. Decreasing pore size and permeability of the network may be the limiting factor in growth, as the biofilms have limited access to nutrients. Another factor on agar substrates is its nontrivial mechanical properties [87]. Agar has a viscous loss modulus that is as large as 50% its elastic storage modulus (Fig. 2.1). Thus, agar behaves as a viscoelastic solid and will dissipate applied stress over timescales of minutes relevant to biofilm growth. Since our data suggests that poroelastic stress in the network promotes fluid flows that deliver nutrients to the biofilm, the effect of viscous dissipation in agar substrates might further limit biofilm growth.

Our interpretation of colony surface expansion data makes a number of assumptions and simplifications. Here, we measure the expansion rate of the biofilm as an important metric of cooperative surface dispersal. We do not ascertain to which degree expansion of the colony arises from increases in cell motility, cell division rates, EPS production, surfactants, or the amount of nutrient availability in the substrate. We do not measure specific genes transcribed during initial cell attachment that are required for biofilm differentiation [88]. Biofilms are known to preferentially form under conditions of external fluid shear flows [89], where there may also be continual renewal of nutrients [90]. Biofilms grown under shear conditions are known to express global gene expression profiles that differ from planktonic bacteria and

colonies grown under agar [90]; however, the production of EPS in our colonies (Supp. Fig. 2.11) indicates some of the bacterial cells show some characteristics of biofilm growth.

An emerging number of studies indicate that bacteria sense surfaces by translating mechanical cues presented by the surrounding environment into biochemical signals through mechanosensitive signaling pathways [31,85,86]. At the scale of an individual bacterium, there are now several molecular machines identified that can read-out mechanical signals, such as the bacteria flagella [91–93], pili [31,85], and cell envelope ion-channels [94–96]. These signals allow bacteria to modulate gene expression, cellular differentiation, and virulence factors [85] in response to physical changes in their environment. An advantage of polyacrylamide gels is that we can comprise gels of increasing stiffness with minimal changes in the surface network by modulating the cross-linker density instead of the monomeric acrylamide. It has been hypothesized that biofilm activation might be faster on stiffer substrates, because bacteria make contact with the substrate network through force-sensitive appendages, such as flagella or pili, at a higher frequency, increasing the possible input for cells to differentiate into a biofilm state. In contrast to the view, here we find that enhanced surface dispersal of EPS-producing colonies occurs even under cases when the surface network is relatively unchanged (Fig. 2.4b).

Our results are consistent with a model of bacteria colony expansion driven by osmotic swelling [32,33] and the poroelastic response of the underlying substrate. Here, we propose that in the context of bacteria colonies one source of surface stress is osmotic pressures that drive swelling and deswelling deformations of the gel substrate. Interestingly, swelling and deswelling deformations have also been recognized in traction force-based experiments in epithelial cell sheet systems [97]. Bacteria colonies are thought to exert different types of

surfaces stresses, including osmotic stress [32,33] but also friction [36] and internal contractile forces [81]. Our TFM based measurements represents a superposition of these effects, and there is currently no obvious way to decouple the stress from these different sources experimentally. As shown in Fig. 2.6, the substrate displacement maps measured here are consistent with localized transient osmotic pressures at the expanding colony edge and a long time global contractile force.

Here we demonstrate that *Serratia marcescens* colonies are capable of responding to changes in substrate stiffness by modulating the amount of surface stress that they exert on their substrates, enhancing the applied stress with increasing substrate stiffness (Fig. 2.6&7). One possible reason is an increased activation in biofilm formation genes that increases rates of EPS production, which form a filamentous network that is better able to transmit forces within the colony and to the surface. We also find a correlation between the colony substrate stress and colony expansion: colony expansion is faster when the colony stress is high. Both colony expansion rates and colony stress increase with substrate stress, but precisely how colony stress and substrate stiffness modulate expansion rates is not yet clear.

There are number of human infections involving biofilms [45,88]. Biofilms are implicated in cystic fibrosis, gingival disease, pneumonia, urinary tract infections, ear infections, and implant infections [45,88]. Serratia marcescens used in this study is an opportunistic bacterium implicated in a range of infections, including urinary and respiratory infections [98]. While the genes required for biofilm formation have been extensively studied from the point of view of

the microbe, there is much less known regarding the requirements for bacteria to infect the soft tissue of their host. A number of recent studies have illuminated the role of substrate stiffness on cell attachment [39,58] and growth and have demonstrated that bacteria can exert direct forces that remodel and disrupt host tissue [81]. Human tissues that bacteria infect vary in shear stiffness, ranging from 10-100 Pa for mucus and 10 kPa for lung to 100-1000 kPa for skin and gut [99–101]. Inflammation and disease can further alter host tissue stiffness [101–103]. Our work here shows that the mechanical properties of extracellular environment impact colony expansion, which has important implications for understanding the infection of soft tissues in vivo.

Our results provide compelling evidence that biofilms can respond to the mechanical properties of their environment beyond single cells and at the collective cell level. Our results suggest new models of biofilm growth that explicitly account for the effects of substrate stiffness and poroelastic substrate remodeling. Much more work is needed of course, and in this regard we note that the polyacrylamide gels presented here can be adapted to investigate the effect of specific adhesion factor presented on the surface or to systemically introduce substrate viscoelasticity [62,104]. Polyacrylamide hydrogels offer a conceptually simple platform for studying how substrate stiffness impacts bacteria surface dispersal and guiding our understanding of collective colony growth.

2.4 Methods

Cell culture: There were four strains of bacteria used in this study: *Serratia marcescens* (274 ATCC), *Pseudomonas aeruginosa* (Xen05), *Proteus Mirabilis* (BB2000), and *Myxococcus xanthus*

(DK1622). With the exception of *M. xanthus*, bacteria cells were inoculated and grown in LB medium with shaking at 37 °C overnight. *M. xanthus* was inoculated and grown in CTTYE medium. The cell density was measured at OD_{600} using 1-cm cuvettes (Globe Scientific 112137) and a spectrophotometer (Thermo Fisher Scientific Genesys 50). Cell suspensions were then diluted to 0.6 at OD_{600} in cell medium. For all bacterial strains, 5 µL of inoculum was spotted on growth substrates (agar or PAA gels of varying stiffness). Cultures were then maintained at 37°C (or 30°C for *M. xanthus*) for up to 15 hours. *Pseudomonas aeruginosa* Xen05 was kindly provided by Dr. Robert Bucki (Medical University of Białystok), and *Proteus Mirabilis* (BB2000) by Dr. Karine Gibbs (Harvard University).

Gel preparation: To prepare hydrogels of varying stiffness, polyacrylamide gels were prepared as described previously [105,106]. Briefly, polyacrylamide gels were prepared by mixing together acrylamide, bis-acrylamide, and distilled water at various ratios. Polymerization was initiated by the addition of 0.5 µL electrophoresis grade tetramethylethylenediamine (TEMED) followed by 1.5 µL of 2% ammonium per-sulfate (APS) per 200 µL of final gel solution. 200 µL of the solution were then pipetted between two glass coverslips, one treated with glutaraldehyde (bottom) and the other SurfaSil-treated (top) and allowed to polymerize for 20 minutes. Then, the top cover slip was removed from the gels, and the final dimensions of the hydrogel formed a disc, 18 mm in diameter and 0.8mm in height. For TFM experiments, fluorescent beads (4.8µm diameter Fluroro-Max polymer microspheres) were embedded in the gels by using a 1:20 dilution of the bead solution in distilled water. The dilution was performed after centrifuging the bead solution and replacing the supernatant surfactant with distilled water. For

a complete list of the gel formulations used in this manuscript, please see Supp. Table 2.2 in the Supplementary Materials.

Rheological characterization: Rheology measurements were performed on a Malvern Panalytical Kinexus Ultra+ rheometer equipped with a 20 mm diameter plate. The elastic gel solutions were polymerized at room temperatures between the rheometer plates at a gap height of 1 mm (30 minutes). The shear modulus was then measured as a function of shear strain from 2-50% at a frequency of 1 rad/sec. For agar, G' was chosen as the shear stiffness in the limit of 0% shear strain.

Substrate preparation and inoculation: To prepare PAA substrates for inoculation, we followed a protocol previously described by Tuson et al [107]. The PAA gels were washed three times (two ten-minute washes and one overnight wash) in phosphate-buffered saline (or TPM buffer for *M. xanthus*). The washes were then repeated with LB medium (or CTTYE medium for *M. xanthus*). Before inoculation, the substrates were removed from growth medium, allowed to dry for 20 minutes at room temperature, and then treated with UV sterilization for an additional 20 minutes. The prepared bacterial solution was inoculated onto the center of each gel in a 5 µL droplet. After placing the droplets, 2 µL of liquid was removed from each droplet with a pipette to bring bacteria in closer contact with the gel surface.

Imaging: Time-lapse imaging was performed with a Nikon Ti-E inverted microscope equipped with a 4X objective. The cultures were maintained at 37°C (or 30°C for *M. xanthus*) using a Tokai-Hit stage top incubator. Images were taken every 10 minutes for 15 hours using a motorized stage to capture growth at four positions along the edge of each biofilm. After the 15 hours had elapsed, full colony images were taken with a MotiCam camera or using NIS Elements software to automatically stitch together multiple images taken with a 2X objective.

Motility measurements: Time lapse images were loaded in custom Python scripts that allowed manual supervision of automated boundary detection (Video S1). The boundaries were fit to circular arcs, and the average length of multiple radial lines connecting subsequent arcs determined the biofilm velocity. (Fig. 2.2b) Velocities are measured over 20-minute time increments. The colony expansion rate was measured at four different imaging windows along the periphery of each colony, and the mean expansion rate was computed for each colony. Velocities were measured at the earliest time at which expansion was present across gel conditions, which varied by species. For the data reported in Fig. 2.4 (*S. marcescens*), velocities were measured after two hours of growth. For the data reported in Fig. 2.5, velocities were measured at six hours for *P. aeruginosa*, three hours for *P. mirabilis*, and ten hours for *M. xanthus*. Colony expansion rate data presented in Fig. 2.4&5 are computed from the mean of 3-6 independent bacteria colonies per condition. Each experiment condition was verified from at least two separate inoculations of the bacterial stock on different days. Error bars denote the standard errors of the mean.

Traction force microscopy based methods: For TFM, bacteria were placed on PAA gels with embedded 4.8 μm fluorescent beads. Deformation of the PAA gel was captured by time-lapse imaging of the fluorescent beads during biofilm growth. The displacement field on the PAA gel generated by biofilm traction was calculated by correlating time-lapse fluorescence images relative to the first frame of the sequence with particle imaging velocimetry (PIV) [108]. The displacement field is then corrected for stage drift by subtracting the displacement field generated from the fluorescent beads' images of the stress-free region of the PAA gel (far from the biofilm). The stresses that the biofilm exerts on the substrate can then be reconstructed from this deformation field using the finite element method (FEM) [82–84]. In brief, the gel was modeled as a 3D block with a thickness of 1mm. The biofilm and PAA gel were meshed with four-noded tetrahedral 3D solid elements using a meshing algorithm. Forces with the same magnitude but opposing direction to the local stresses were applied to each node. Internal strains and stresses were then computed based on the geometry and elastic properties of the gel. The stress calculated is measured relative to the (prestressed) first frame of the imaging sequence (accumulated stress). The computation routine was performed using MATLAB and ANSYS Mechanical APDL. For instantaneous stresses, the displacement field is generated by comparing two consecutive frames of the captured fluorescent beads images. Subsequent analysis follows the same protocol described for accumulated stress above. Surface shear strain ϵ was estimated using the relation $\tau=G\epsilon$, where τ is the surface stress given by TFM and G is the shear modulus measured from rheology.

2.5 Supplementary Materials

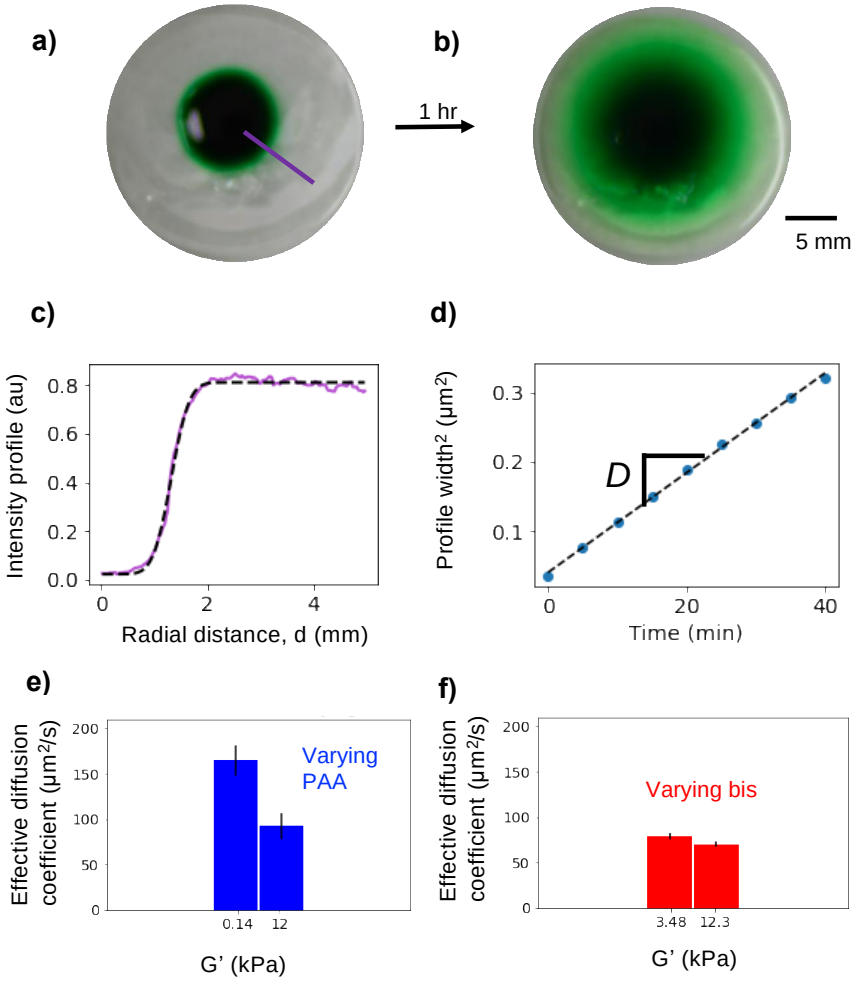
% PAA	% Bis	G' (kPa)
3	0.15	0.14 ± 0.02
3.5	0.15	(0.5)
4	0.15	0.94 ± 0.09
6	0.15	2.72 ± 0.13
8	0.15	5.34 ± 0.35
10	0.15	(10)
12	0.15	12.0 ± 0.93
8	0.02	0.68 ± 0.14
8	0.05	1.62 ± 0.31
8	0.085	(3.5)
8	0.2	6.02 ± 0.59
8	0.3	8.82 ± 1.09
8	0.45	(10)
8	0.6	11.8 ± 0.42

Supplementary Table 2.2. Summary of hydrogel compositions and their corresponding storage modulus G'. Reported values are mean ± standard deviation, as measured by Malvern Panalytical Kinexus Ultra+ stress-controlled rheometer (details in Methods section). Values in parentheses are estimated based on linear fits between experimental points from Figure 2.2c and 2.2d as appropriate.

% PAA	% Bis	Contact Angle
4	0.15	15° ± 2°
8	0.15	23.5° ± 1°

n ≥ 3 measurements

Supplementary Table 2.3. Summary of contact angle (mean ± standard deviation) of water on the surface of PAA hydrogels. These measurements were made using a contact angle goniometer.



Supplementary Figure 2.8. Characterizing effective transport of a dye through polyacrylamide gels. (a&b) A drop of green food-coloring dye spreads through a PAA gel, pictured at **(a)** initial and **(b)** final (60 min) time points. The transport of the dye is due to fluxes arising as the gel changes state and swells or de-swells as well as due to molecular transport. **(c)** The intensity profile of the diffusing dye is fit to an error function with a specific width w (initial profile shown). **(d)** The slope of w^2 over time gives the effective (transport) diffusion constant D for the hydrogel. **(e&f)** The effective diffusion constant for gels with varying **(e)** PAA concentration versus varying **(f)** bis concentration. Results show that D decreases with increasing PAA concentration, but is approximately constant for gels with increasing bis concentration. These data span the stiffness range used in this work. The formulation for the gels shown in (e) and (f) are in order from left to right: 3% PAA, 0.15% Bis; 12% PAA, 0.15% Bis; 0.085%, 8% PAA; and 0.45% Bis, 8% PAA.

Quantifying matrix permeability

To quantify the permeability of the gel networks, we used a simple dye diffusion experiment (Supp. Fig. 2.8). Drops of green-food coloring dye were placed on the surface of the gels and

allowed to disperse through the gel over time. The evolution of the dye was tracked via images taken 10 minutes apart (Supp. Fig. 2.8a&b). Here, the image intensity correlates with the concentration of the dye. The intensity profile across the expanding edge of the droplet was well-approximated by an error function. Thus, we fit the intensity profile curves to an error function to obtain a profile width *w* over time (Supp. Fig. 2.8c&d). The width of the intensity profile increased over time and the effective diffusion *D* of the dye through the gel could be estimated by the slope of the w^2 over time (Supp. Fig. 2.8d). The diffusivity values measured in our gels varies from approximately 50 to 150 $\mu m^2/s$.

Here, the diffusion coefficient that we measure is an effective dispersion coefficient, which combines these effects of molecular diffusion and characterizes the transport of solvent as the gel swells. Our results suggest that a significant component of the transport is due to fluxes arising from a redistribution of fluid in the gel as it swells.

Estimating effective matrix pore size

Our results suggest that the development of biofilms on soft, porous substrates depends on both the elasticity of the gel and its permeability. The permeability of the substrate controls the resistance to the flow of nutrient fluxes through the gel matrix. Next, we combined rheological data and diffusion experiments to compute a network permeability *k* and estimate an effective pore size r_p of the networks relevant to osmotic-induced spreading of bacteria colonies.

Here, we assume that the polyacrylamide substrates act as poroelastic gels that can swell. Poroelastic relaxations are associated with fluxes that aim to re-establish chemically

equilibrated states occur in a characteristic time τ_p. During the deformation or stressing of highly swollen polymer networks, mechanical deformation is coupled to the mass transport of solvent through the network. Localized stresses such as those exerted by a large area of the biofilm results in substrate deformation and causes a chemical potential gradient to develop within the hydrogel close to the interface. Over time, the hydrogel establishes a new chemical equilibrium via migration of liquid (here the nutrient solution) away or from the region under deformation. Due to this process, the hydrogel undergoes a load relaxation that reflects the time-scale to establish this new equilibrium τ_p. The volume of the deformed region establishes the volume of the nutrient that must migrate, the relaxation process depends on the area over which the gel is deformed or chemical stressed are induced

To estimate the permeability *k* of a gel, we first note the equation relating the (poroelastic) diffusivity D to the shear modulus G, the Poisson ratio ν, the permeability k, and the medium viscosity η,

$$D = \frac{2(1-\upsilon)}{(1-2\upsilon)} \frac{Gk}{\eta} .$$

The diffusivity provides the length scale $\sqrt{Dt}$ over which nutrient concentration is transported due to fluid fluxes (local induced pressure gradients) in a time t due to imposed deformations. The Poisson ratio ν characterizes the ability of the gel to swell. The permeability, Poisson ratio and the shear modulus depend on the degree of swelling of the gel.

The effective poroelastic diffusion coefficients for different PAA substrates D, are roughly estimated by analyzing the radial spread of a small molecule dye (Supp. Fig. 2.8) that can be transported by and move with fluid in the presence of fluxes, and the shear modulus *G* is

measured via oscillatory tests (Fig. 2.1). Previous studies reported an estimated Poisson ratio of around 0.25-0.4 for PAA gels with concentrations of 5 wt% monomer when swollen significantly, which we choose to measure here a compression experiment using the rheometer (Supp. Fig. 2.9). Knowing D, G, υ, and using the viscosity of the solvent permeating the gel (approximating it to be that of water), we can compute the permeability k.

To calculate the Poisson ratio, we use a parallel-plate rheometer to apply a small uniaxial compression. The gel thickness, H = 1.0 mm, is compressed by a vertical amount $\delta \ll H$. The top plate of the rheometer is a flat disc of radius R and the volume of the indented region is $\pi R^2 \delta$. The normal force on the top plate from the sample is measured over time (Supp. Fig. 2.10). Here the gel is unjacketed and kept in a medium that allows for motion of fluid through the outer boundaries.

Upon compression, the normal force exhibits a steep instantaneous rise in value, which subsequently relaxes over time. At short times, the gel behaves as an incompressible material since the fluid does not have time to flow out of the region – this provides an instantaneous load F_0 that eventually relaxes to a long-time limiting value F_∞. Ignoring the effects of tortuosity and assuming that the final state of indentation allows the gel to relax to a Poisson ratio that is its equilibrium value, we use the relationships [109–111]

$$\frac{F_0}{F_\infty} \approx 2(1 - \nu)$$

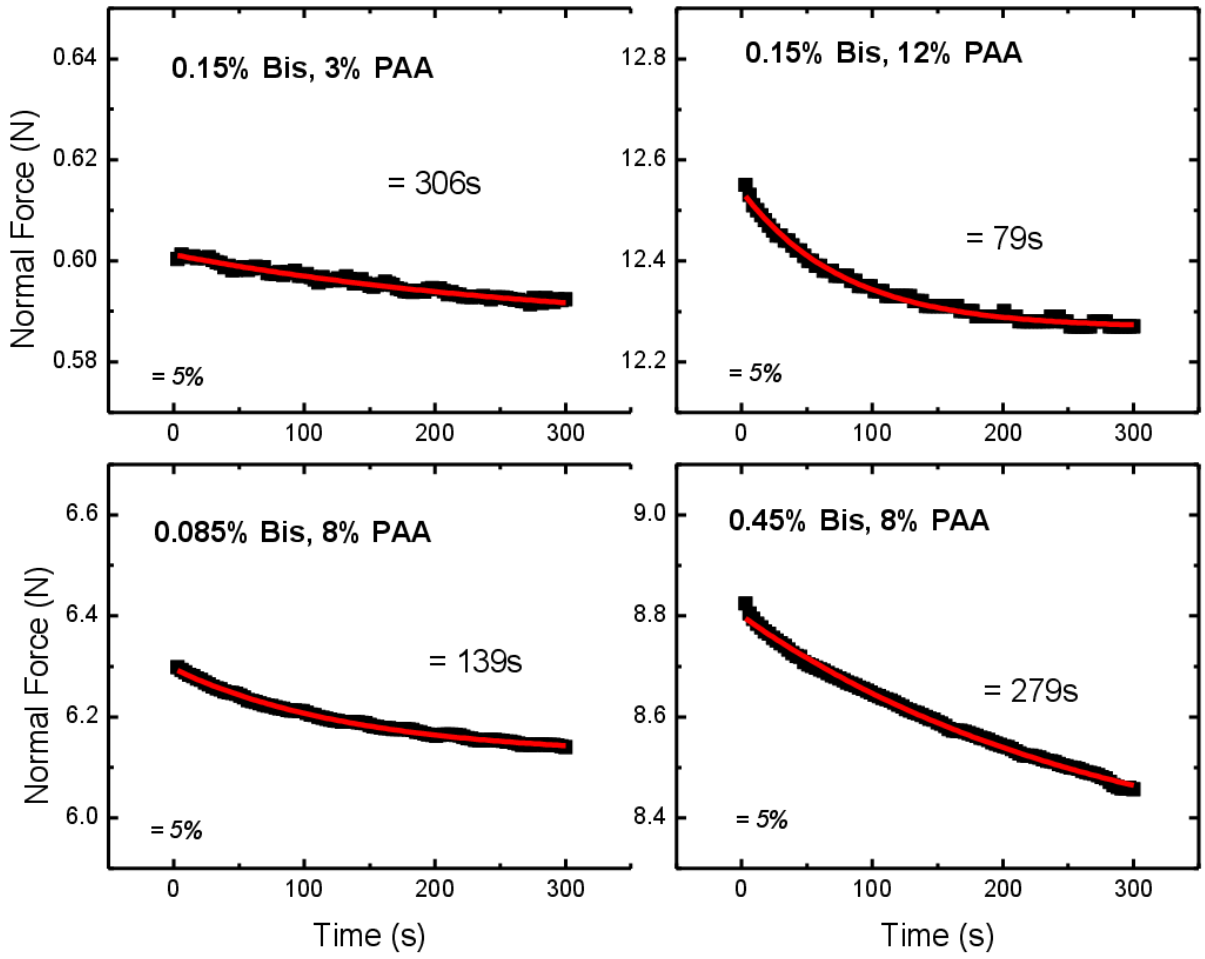
and

$$r_p \approx 2 \left(\frac{\eta}{2} \frac{D(1-2\nu)}{G(1-\nu)} \right)^{\frac{1}{2}}.$$

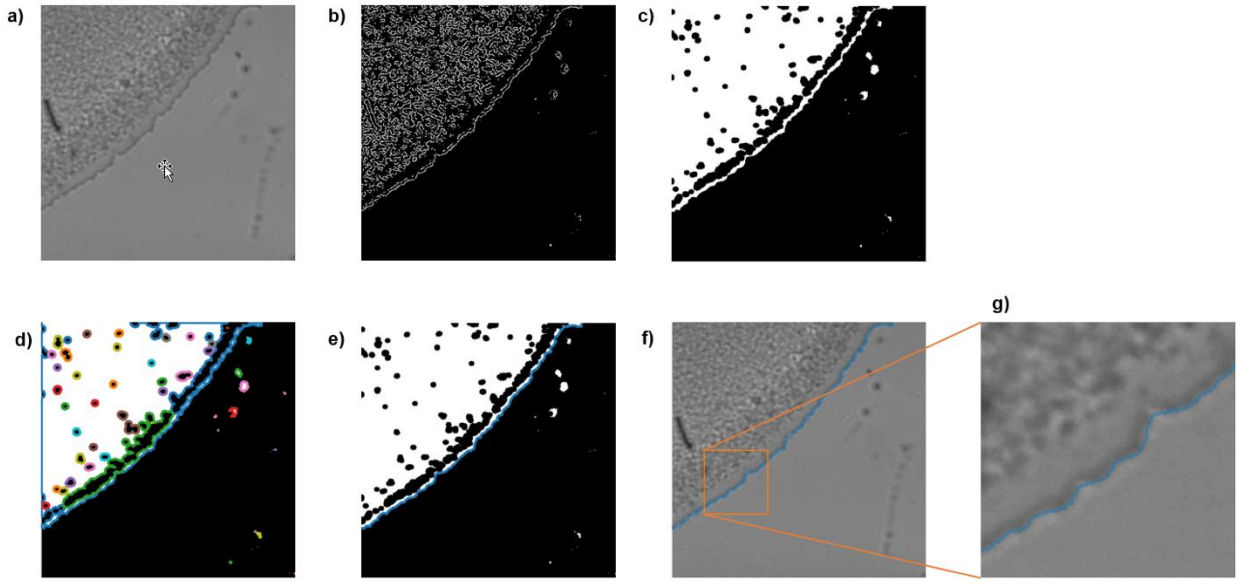
Supp. Table 2.4 summarizes the elastic and transport coefficients measurements from this work. Our estimated Poisson ratios are in the range of 0.43 to 0.49 (Supp. Table 2.4). Assuming the viscosity of the fluid permeating the gel is close to that of water and ignoring the effects of tortuosity within the gel, the estimated mean pore size is in the range of 0.8 to 22 nm (Supp. Table 2.4), consistent with prior measurements. Note that the shear modulus does depend on the confinement and the compression. For small compressions, we may approximate the shear moduli by the uncompressed values.

% PAA	% Bis	G' (kPa)	Poisson Ratio	Pore size (nm)
3	0.15	0.14 ± 0.02	0.43 ± 0.06	22 ± 5.6
12	0.15	12.0 ± 0.93	0.49 ± 0.01	0.85 ± 0.14
8	0.085	(3.5)	0.49 ± 0.01	1.5 ± 0.11
8	0.45	(10)	0.49 ± 0.01	0.9 ± 0.07

Supplementary Table 2.4. Parameters used to estimate effective pore size.



Supplementary Figure 2.9. Normal force relaxation of PAA gels. Representative normal force decay curves from uniaxial compression of different PAA gel compositions. 5% uniaxial compression was applied to each gel and the normal force decay was measured with a parallel plate rheometer equipped with a 20 mm plate. The Poisson ratio ν was then given by $\frac{F_0}{F_\infty} \approx 2(1-\nu)$, where F_0 is the initial jump in force when the compression is applied and F_∞ is the final value of the force after three minutes of observation. We conducted three independent trials per condition to compute a mean and standard deviation.



Supplementary Figure 2.10. Automated biofilm boundary detection algorithm. (a) An unprocessed biofilm image. **(b)** Canny edge detection produces a binary image. **(c)** A circular kernel nine pixels wide smooths the boundaries and closes broken lines. **(d)** The function cv2.findContours identifies the boundaries around contiguous regions, each shown here in a different color. The segment with the longest edge-to-edge distance matches the biofilm boundary but often includes erroneous features around the finite edges of the image. **(e)** Since the boundary consistently intersects the edge of the picture, these erroneous features are easily removed by cutting the biofilm boundary segment at the picture edges. **(f)** The coordinates identified with this method follow the biofilm boundary with high precision. **(g)** Zoomed-in snapshot of boundary from (f).

Supervised biofilm boundary detection

The quantitative metric of biofilm growth used in this study is biofilm boundary velocity. To calculate the boundary velocity from a sequence of time lapse images, image processing is used first to automatically detect the coordinates of the biofilm boundary for each image. Then the detected boundary is manually verified and, if necessary, corrected before being further processed as explained in the Methods section.

The automatic boundary detection and manual correction steps were integrated into a single Python script to increase efficiency and ease of use. Highly detailed edge boundary coordinates can be extracted from many images in this way.

The Python implementation of the OpenCV library [112] is used to implement the steps of the automatic boundary detection algorithm. Using the Anaconda distribution of Python, OpenCV can be installed as `pip install opencv-contrib-python`, and then used in scripts with `import cv2`. Loading images is accomplished with the PIMS package.

The automatic boundary detection algorithm consists of the following steps:

1. **Canny edge detection (`cv2.Canny`)**

   This produces a binary image where edges are shown as white pixels, and the rest of the image is black. Two parameters, a minimum and maximum gradient threshold are the only inputs.

2. **Morphological closing (`cv2.morphologyEx`)**

   This step closes the broken lines often produced by Canny edge detection by first dilating white pixels (i.e. outlining white pixels in white by a specific thickness, or kernel size), which bridges the gaps between nearby regions, and then eroding the white pixels using the same kernel size to obtain the same edge thickness as before.

3. **Closed contour identification (`cv2.findContours`)**

   This function identifies all the contiguous white regions in the image and returns the contour boundary positions for each one. The contour with the longest edge-to-edge distance consistently matches with the biofilm boundary (Supp. Fig. 2.10e).

4. **Contour segmentation**

   This step removes the portions of the closed contour that lie on the image edge, removing erroneous edge features and leaving line segments that extend from edge to edge of the image (Supp. Fig. 2.10e).

5. **Segment selection**

   Finally, the segment with the longest end-to-end distance is displayed as the predicted biofilm boundary.

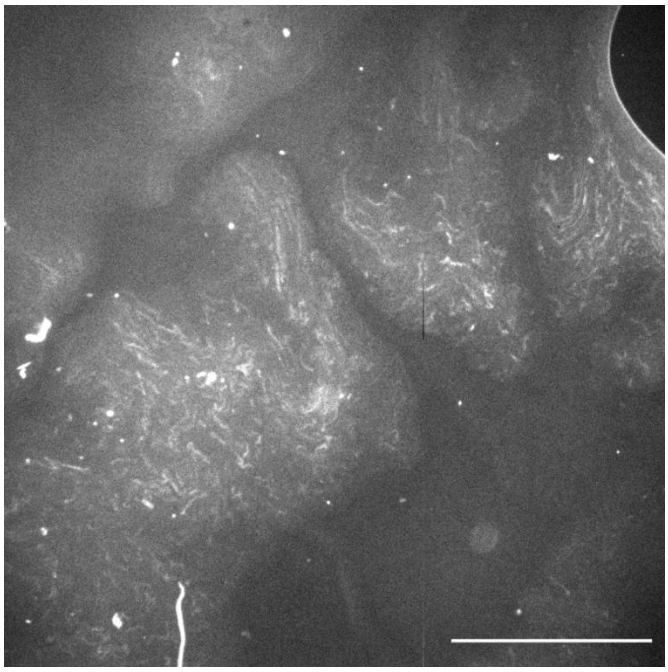
Occasionally, an imperfection in the gel surface can get identified as part of the biofilm boundary. To address this issue, we developed a GUI that allows the user to click on a point on the automatically identified boundary and manually draw a new segment portion that ends on the boundary or the image edge. These manually drawn points replace the erroneously identified points, and the coordinates of the resulting biofilm boundary are saved by the user when all corrections are complete. A demo is shown in Video S1.

Taken together, this semi-automated boundary tracking code significantly reduces time spent in analysis. A typical experimental trial generates 1,000-2,000 images (12 individual colonies in a multi-well plate x 4 locations per colony x 3 frames/hr x 10 hours). Manual tracing of this many boundaries, in ImageJ with a digital pad for example, would require multiple days to complete. Using this code, 1000 individual images can be analyzed and verified for boundary identification in about one hour. The full code is available via Github at https://github.com/masp01/SUBII-Trace, along with sample biofilm images. Any data used in this study is available upon request.

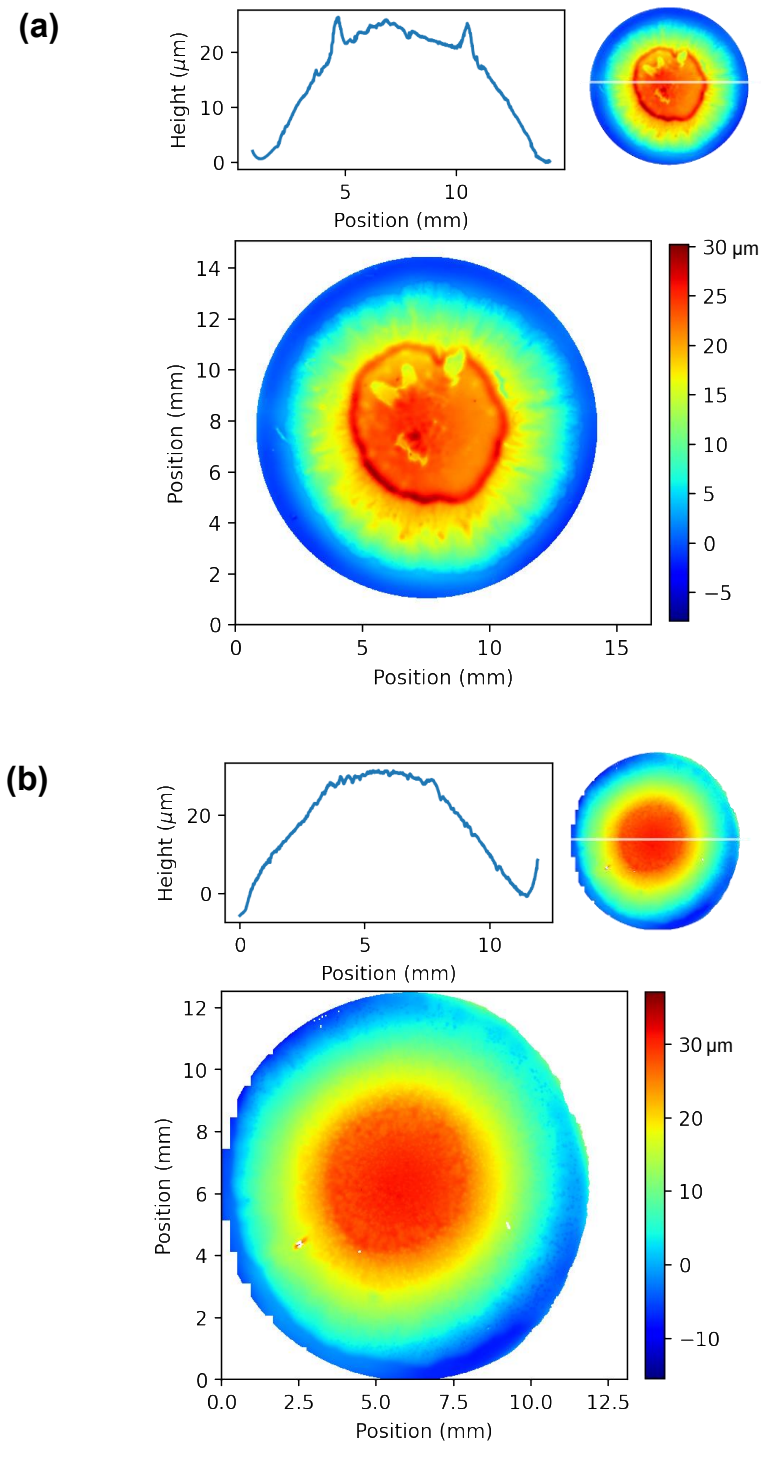
The resolution of this technique depends on the closing kernel radius (step 2). Here, we used 4.5 pixels, or 14.5 µm with our 4x objective, which is larger than an individual cell but much smaller than the length of boundaries (several mm).

Biofilm matrix fluorescent imaging

To determine whether the bacterial colonies are excreting EPS, after 15 to 24 hours of growth, *Serratia marcescens* colonies were stained with Invitrogen Film Tracer SYPRO Ruby Biofilm Matrix Stain (Thermo Fisher F10318) according to the manufacturer's instructions. Supp. Fig. 2.11 shows a representative image of a biofilm with positive matrix staining. Adding the matrix strain slightly disrupts the colony structure, resulting in a diluted mixture as seen in the image.



Supp. Figure 2.11. EPS fluorescent staining indicating biofilm development. The edge of the *Serratia marcescens* colony after 15 hours of growth is present in the upper-left corner. Biofilm matrix stain was applied to the surface of the biofilm, revealing regions of concentrated EPS that indicate biofilm development. Scale bar is 1 mm.


(a)
Height (μm)
20
10
0
5
10
Position (mm)
Position (mm)
14
12
10
8
6
4
2
0
0
5
10
15
Position (mm)
30 μm
25
20
15
10
5
0
−5
(b)
Height (μm)
20
0
0
5
10
Position (mm)
Position (mm)
12
10
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
Position (mm)
30 μm
20
10
0
−10

Supplementary Figure 2.12. Three-dimensional landscape of bacteria colonies. The 3D colony landscape was mapped with a white light interferometer (Bruker CONTOURX-200). Representative interferometer images for *Serratia marcescens* colonies on: **(a)** a soft, 500 Pa gel (3.5% PAA, 0.15% Bis) and **(b)** a stiff, 5 kPa gel (8% PAA, 0.15% Bis).

Captions for Supplementary Videos (available online)

Video S1: Demonstration of the GUI that implements the automated boundary detection algorithm, allowing the user to quickly verify the boundary quality and make manual updates.

Video S2: Representative *Serratia marcescens* biofilm growing on a soft polyacrylamide hydrogel (G' = 0.9 kPa). This video was taken with a 4X objective at 10 minutes/frame for 15 hours. Playback is at 4200 x real-time speed.

Video S3: Representative *Serratia marcescens* biofilm growing on a stiff polyacrylamide hydrogel (G' = 3 kPa). This video was taken with a 4X objective at 10 minutes/frame for 15 hours. Playback is at 4200 x real-time speed.

Video S4: A *Serratia marcescens* biofilm growing on a soft polyacrylamide hydrogel (G’ = 0.9 kPa) with 4.8 µm diameter fluorescent beads embedded in the gel to display substrate displacements. During the first loop, the biofilm is shown in brightfield. During the second loop, only the fluorescent beads are shown. During the third loop, an outline indicating the biofilm boundary is overlaid. This video was taken with a 10X objective at 10 minutes/frame for approximately 1.5 hr. Playback is at 4200 x real-time speed.

3. Phenotypic similarity measures redundancy of genes

This chapter is based on the article "Phenotypic similarity is a measure of functional redundancy within homologous gene families," posted as a preprint to bioRxiv in 2022, and coauthored by myself, Jessica Comstock, Fatmagül Bahar, Isabella Lee, Alison Patteson, and Roy Welch. The manuscript was written by Jessica Comstock and myself, with editorial work by Alison Patteson and Roy Welch. The original experiments upon which my analysis was based were performed by Fatmagül Bahar. Isabella Lee aided me in the development of the analysis code and manual observation of time series data. Manual phenotype analysis was performed by Jessica Comstock.

3.1 Introduction

A reverse genetics approach to characterizing a gene often begins by disrupting or deleting the gene and observing the resulting phenotype. Differences between the mutant and wild-type phenotypes can provide invaluable insights regarding gene function(s), but in practice many single-gene knockouts, even those in genes predicted to be important based on previously studied homologs, yield phenotypes that are relatively minor or indistinguishable from the wild-type organism [113,114]. This robustness to the phenotypic impact of genetic mutation is an important part of an organism's phenotype and has implications for fitness.

Robustness is commonly attributed, at least in part, to functional redundancy, or the tendency for functionally similar genes to compensate for the role of a disrupted gene [115]. Functional redundancy can arise through many mechanisms including duplication and divergence, where reduced selective pressure can cause paralogs to accumulate mutations and take on new,

slightly different functions over time [116,117], 2002). Paralogs that are maintained over long timescales often retain some of their ancestral function in addition to their diverged function [118,119], thus building in redundancy. Repeated gene duplication events can give rise to large gene families wherein genes have a range of biochemically similar but specialized functions. Though many homologs in a gene family may be capable of performing a similar function, due to divergence it is difficult to predict which genes might be able to compensate for the function of others. The most recent duplicates within a gene family are not always capable of being functionally redundant while some older and more diverged paralogs are [120]. Sequence similarity alone is not enough to predict functional redundancy, and the extent to which duplicates contribute to robustness varies across organisms [121]. For these reasons, it is unclear to what extent families of homologs are contributing to the functional redundancy that gives rise to robustness in biological systems.

Many studies attempting to elucidate functional redundancy in the genome involve the creation of single and double knockouts of paralogs to probe for synthetic lethality [122–124]. While this method is effective in assessing functional redundancy in pairs of closely related genes, it is limited in its power to explore larger networks of redundancy, as may exist in expanded gene families. A double mutant that does not show a more significant phenotype than each of the corresponding single mutants could imply either that the genes are non-redundant, or that they are part of a larger redundancy network that has a strong buffering capacity and therefore has decreased fragility in the face of genetic perturbation [125]. In this way, phenotype is often the readout for assessing redundancy and robustness within biological systems. The phenotypic impact of mutation reveals information about robustness, and we can

investigate the mechanisms that lead to robustness by considering gene sequence, so understanding how redundancy affects robustness is a crucial genotype-to-phenotype question.

Any given gene processes the flow of information from precursors, producing outputs that feed into other networks or cellular functions. In a simple case of non-redundancy, a gene produces one protein with a primary function, and when this gene is intact, expresses a wild-type phenotype (Fig. 3.1A). A mutation in this gene would severely impact the fitness of the organism. However, if a given gene is part of a network of structurally similar genes which each have their own primary function but also retain some ancestral function, as in gene families that arise from duplication, the impact of a mutation can be diffused through the other members of its network, producing a relatively minor deviation in phenotype. Redundancy networks (Fig. 3.1B), which we here define as the group of two or more genes whose products can compensate for the loss of function of one another, allow for the rerouting of information through alternative pathways so that the end result has a minimal impact on fitness. As shown in Figure 3.1C, an additional byproduct of this buffering effect is that knocking out one member of a redundancy network should produce a similar phenotype as knocking out any other member of that group, because the entire set of genes is affected no matter which component of the network is disrupted by mutation. In this way, phenotypic similarity may be an important indicator of functional redundancy within homologous gene families and may provide insights into the level of robustness in a genome. Further, because a gene's redundancy network likely overlaps significantly with its family of homologs due to the relationship between protein

structure and function (Fig. 3.1D), we predict that mutations in genes from within the same gene family will be more phenotypically similar.

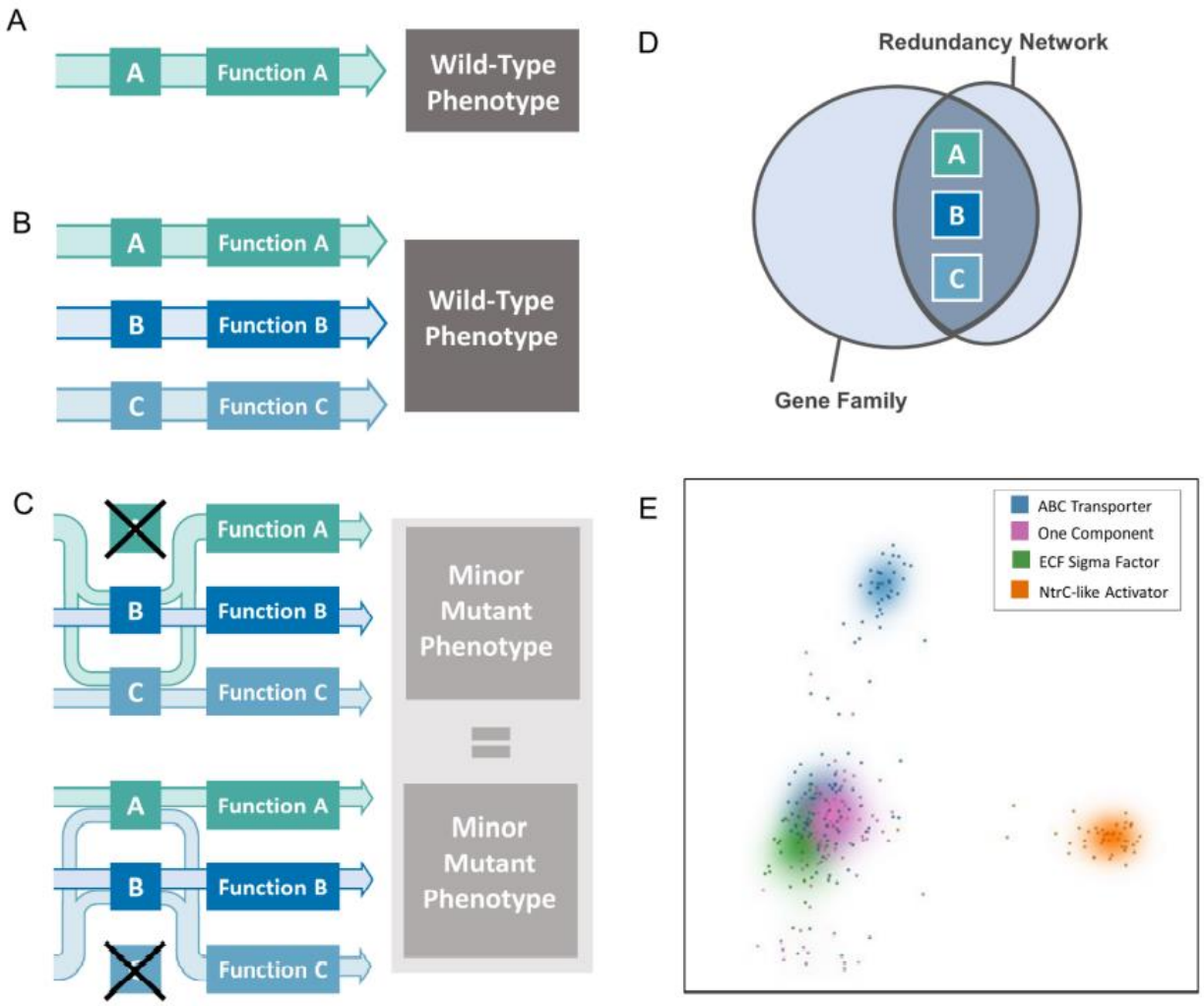


Figure 3.1. Functional redundancy resulting in phenotypic similarity. **(A)** In a pathway with no redundancy, Gene A contributes to Function A. Any mutation that renders Gene A nonfunctional would produce a severe phenotype or lethality if Function A is essential. **(B)** Genes A, B, and C belong to the same redundancy network, meaning each gene can compensate for the loss of function of one member of its network. In the scenario where all three genes are functional and operating optimally, each gene contributes to its primary function (for example, Gene A is responsible for most of the contribution to Function A), producing the wild-type phenotype. **(C)** When a mutation occurs that renders Gene A nonfunctional (top), the input to Gene A gets rerouted through Genes B and C such that Function A can still occur, but in a slightly reduced capacity (indicated by thickness of arrows compared to panel B). Since Genes B and C are processing more input from A, Functions B and C are also affected and operate at a reduced capacity. The slight reduction in function of all three network components produces a phenotype that is relatively minor and may be indistinguishable from wild-type. A mutation in Gene C (bottom) would result in a similar phenomenon, where the input that normally feeds into Gene C is processed by Genes A and B, resulting in overall decreased output from each. In this model, a mutation in one member of a redundancy network affects the

output from all components regardless of which gene contains the mutation (indicated by the similar output arrow size of top and bottom of panel C), and we predict that mutations in members of the same redundancy network will produce similar phenotypes. **(D)** Though not every member of a gene family is functionally redundant, and there may be redundant genes that do not belong to the same gene family, the relationship between structure and function of proteins dictates that genes in the same redundancy network are likely to come from the same gene family. If redundancy networks come primarily from the same gene family, and components of redundancy networks show similar mutant phenotypes, then members of the same gene family would be more likely to produce the same mutant phenotype. **(E)** Similarity of the protein sequences used in this study by multidimensional scaling. Each point represents one gene, with a Gaussian kernel density estimate to guide the eye. Proteins that are more similar in sequence, belonging to the same gene family, cluster together. Each gene family forms a single cluster with the exception of the ABC Transporters, which form two major clusters due to the different subunits. The highly conserved ATP-binding domains [126] separate very distinctly from the periplasmic and substrate-binding domains. We predict that mutations in genes that belong to the same gene family will be more phenotypically similar than they will be to mutant phenotypes in other paralogous gene families. Thus, we expect phenotype to cluster by gene family.

To test this, we phenotypically characterized over 250 single-gene mutations in *Myxococcus xanthus*, a soil bacterium with a large genome containing multiple homologous gene families [127] and examined the relationship between gene family and phenotype. Under nutrient stress, swarming cells of *M. xanthus* undergo development, aggregating into multicellular fruiting bodies wherein populations of cells will differentiate into spores [128] (Fig. 3.2A). Since the ability of *M. xanthus* to form fruiting bodies and sporulate is directly tied to its fitness, it is likely a robust biological process that involves many functionally redundant genes. We generated a library of microscopic time-lapse movies (time series) showing the development of 265 knockout strains of *M. xanthus* belonging to four different gene families (102 ABC transporter genes, 45 NtrC-like activators, 80 One component signal transduction genes, and 38 ECF sigma factors; see references [21,56,129] for previous work on some of these genes in *M. xanthus*). We made qualitative observations of the ways in which resulting phenotypes differed from wild-type and used these observations to inform a novel image processing and phenotypic analysis pipeline that automates quantitative measurements of phenotype that are explicitly defined. Although previous studies have used image processing to extract phenotypic features of aggregate formation [130], this work has applied these tools to

the largest library of time series of which we are aware, necessitating a new pipeline and analysis methods. Finally, we compared the similarity of phenotypes across gene families using principal component analysis (PCA). We found that, just as mutant strains within a gene family cluster by sequence similarity through multidimensional scaling (Fig. 3.1E), they also cluster by gene family in the phenotypic feature space with a statistically significant sharpness (i.e. small cluster size) and separation of clusters, indicating large networks of redundancy within these gene families.

3.2 Results

Manual Characterization of Development Phenotypes

Under starvation, a swarm of *M. xanthus* cells will execute a developmental program during which millions of rod-shaped cells coordinate their movements and self-organize into dome-shaped multicellular aggregates. Some nascent aggregates destabilize and disperse, but most persist and continue to grow; when the persisting aggregates become large enough, cells in the middle of each differentiate to form a cluster of spores, at which point they are considered mature fruiting bodies (Fig. 3.2A). Capturing this process with time-lapse brightfield microscopy results in a time series of grayscale images where initial aggregates appear roughly circular with irregular boundaries, somewhat darker than the background swarm. Later in the time series, dispersing aggregates shrink and disappear, and the persisting ones grow and darken, with boundaries that become stable and clearly defined (Fig. 3.2B). Image features such as these can be leveraged to compare development phenotypes between wild-type and mutant *M. xanthus* strains.

For this study we recorded 24-hour time series for wild-type and a set of 265 single gene knockout mutant strains (Supp. Table 3.4), with an average of three replicates per strain. Due to their important roles in signal transduction, transport, and transcriptional regulation, we predict that genes within these families will be part of redundancy networks to ensure robustness. We compared the mutant phenotypes to our wild-type strain with an emphasis on aggregate composition and dynamics. Wild-type aggregation initiated at 9.2±1.6 hours and formed uniformly dark circular aggregates with stable and clearly delineated boundaries within 24 hours (Fig. 3.2B). Mutant strains that consistently initiated aggregation either before or after wild-type were designated “early” or “late”, respectively. Mutant strains that consistently initiated aggregation at the same time as wild-type but had aggregates that failed to darken and/or form clearly delineated boundaries were designated “immature” (Fig. 3.2C). Mutant strains that initiated aggregation at the same time as wild-type but then all the aggregates dispersed within 24 hours were designated “fall apart” (Fig. 3.2D). Mutant strains that initiated aggregation at the same time as wild-type but then all the aggregates dispersed and then re-aggregated within 24 hours were designated “aggregate-reaggregate” (Fig. 3.2D). Mutant strains that consistently matched all aggregation criteria and were indistinguishable from wild-type were designated “Like Wild-Type” (LWT). Finally, Mutant strains where the replicates displayed different developmental classifications were designated “variable”.

Distribution of Manual Development Phenotypes within each Gene Family

Of the mutant strains characterized in this study, less than 10% failed to initiate aggregation at all, and 62% consistently produced fruiting bodies that were qualitatively comparable to wild-type by the end of the 24-hour window. An additional 20% of mutants were able to initiate aggregation, but aggregates remained immature; some of these strains may have formed mature aggregates if the time series extended longer than 24 hours.

We hypothesize that the relatively high success rate of aggregation in these mutants is due, at least in part, to *M. xanthus* development being a robust phenotype. If redundancy networks are contributing to functional redundancy to produce this robustness, then, according to the hypothesis portrayed in Fig. 3.1, mutants within the same gene family will be more phenotypically similar. As an initial test of our hypotheses, we sorted the mutant strains into their gene families and visualized the proportional representation of our developmental phenotype classifications (Fig. 3.2E). The distribution of some phenotypes did seem to favor specific gene families. For example, LWT strains made up over half of the ABC Transporter family, the early aggregating strains compose nearly half of the ECF sigma factor family, and about one third of the One component family produce variable phenotypes in different replicates.

The manual categorization of development phenotypes presented here serves two purposes. First, it provides support for our hypothesis that *M. xanthus* development is as robust a phenotype as we expected, making it a suitable phenotype for observing the extent of functional redundancy networks in gene families. Second, though we do not claim these data alone provide sufficient evidence for the existence of redundancy networks, as the data show only the most obvious associations between gene family and phenotype, these qualitative

observations do provide information about the various ways in which phenotype can differ during development. This was used to inform a more systematic, quantitative, and multidimensional characterization pipeline to test our remaining hypothesis about phenotypic similarity among families of paralogs: if redundancy networks contribute to robustness, and if those networks are comprised primarily of genes within the same family, then a grouping of mutant strains based on phenotypic features should also group the strains according to gene family.

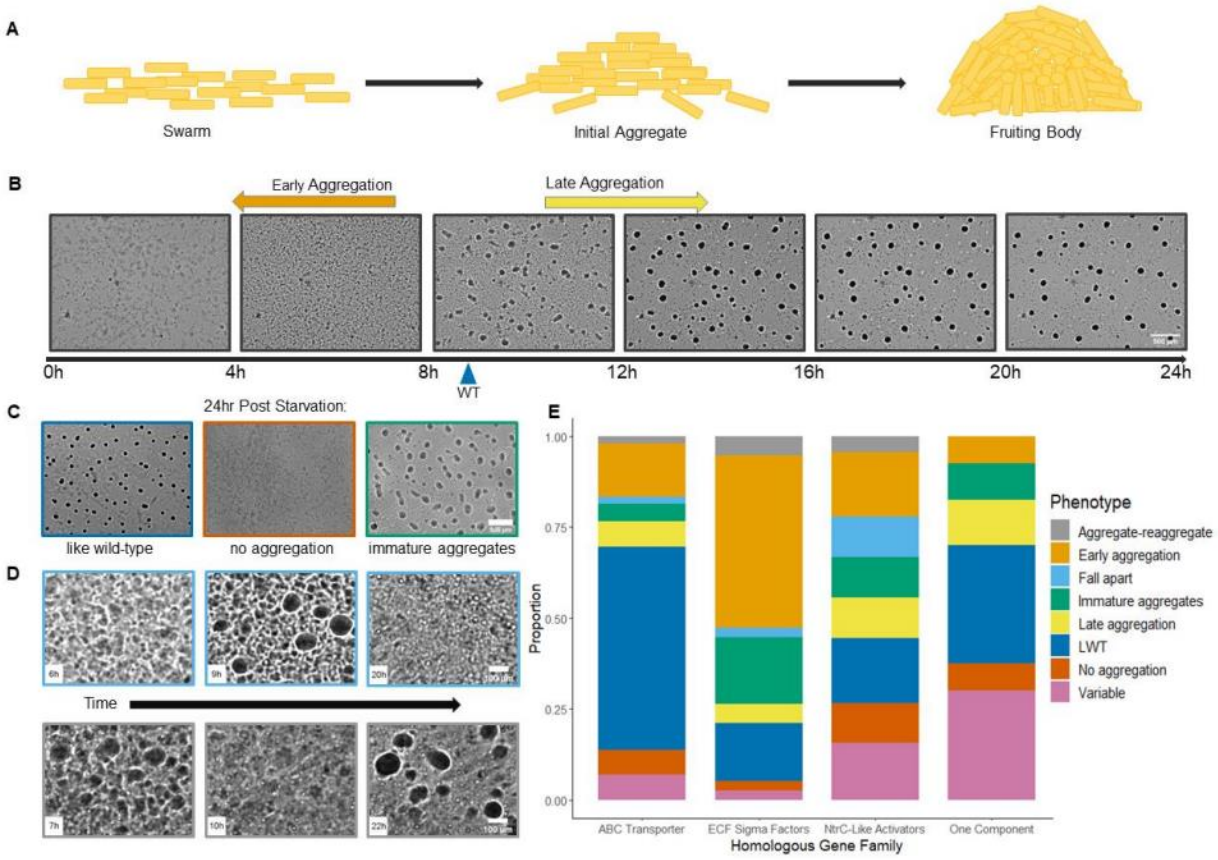


Figure 3.2. Manual categorization of *M. xanthus* development. **(A)** Upon sensing nutrient stress, vegetative *M. xanthus* cells undergo a developmental process that culminates in spore-filled fruiting bodies. **(B)** Wild-type *M. xanthus* cells on TPM agar begin to cluster into early aggregates after 9 hrs of starvation (blue arrow), and as more cells join the premature aggregates over the course of 24 hours, the aggregates mature into fruiting bodies that appear round and dark with conventional brightfield microscopy. Mutant strains that show initial aggregation either before or after the average time for wild-type are assigned the early aggregation (orange arrow) and late

aggregation (yellow arrow) phenotypes, respectively. Scale bar 500 μm. **(C)** Like wild-type (LWT) mutants that produced dark, circular fruiting bodies on a timeline similar to wild-type (left), non-aggregating mutants (center), and mutants that produced immature aggregates (right). Scale bar 500 μm. **(D)** Some mutants formed initial aggregates that eventually shrank and fell apart (top). Other mutants formed initial aggregates that fell apart before re-aggregating into mature fruiting bodies (bottom). Scale bar 100 μm. **(E)** Distribution of development classifications within each gene family.

Automated Characterization of Phenotypes

We developed and implemented an automated image processing pipeline in Python (see Methods, SI). Using it, we were able to identify and track every aggregate in all the time series, recording changes in aggregate number, position, size, shape, and gray value. In total, our pipeline captured the developmental dynamics of more than 150,000 aggregates, both dispersing and persisting. These data were analyzed to determine the timing and position of significant changes in swarm dynamics, such as the initial onset of aggregation, the average aggregate growth rate, and the rate of change in aggregate gray value; these quantitative features serve as an unbiased and more accurate replacement for the manual phenotypes “early”, “late”, “immature”, “LWT”, and “variable”.

We identified 18 quantitative features (Fig. 3.3) to represent and measure the variation observed in the wild-type and mutant strains. For each time series, we calculated a list of these 18 numbers, mapping it to a single point in an 18-dimensional feature space. Distance between points in this feature space is a measure of phenotypic dissimilarity. To reduce the complexity of this data and visualize it, we used principal component analysis (PCA), a deterministic method with no additional input parameters, to reduce the feature space to two dimensions, PC1 and PC2. The distribution of points on a two-dimensional map of PC1 versus PC2 captures the phenotypic features that vary the most across the dataset, while discarding those combinations of features that vary less.

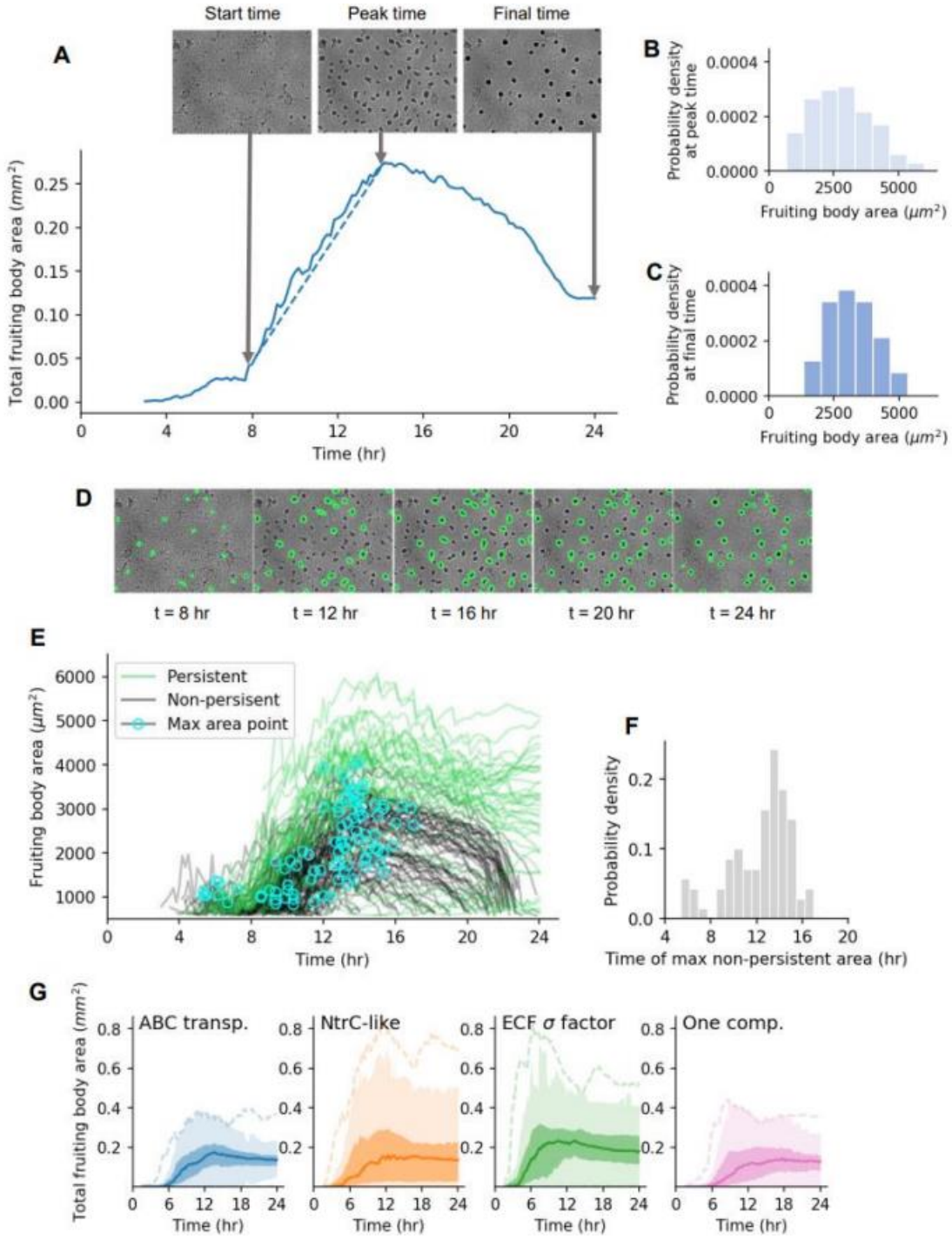


Figure 3.3. Automated quantification of fruiting body formation phenotypes. **(A-C)** Features related to global fruiting body development **(D-F)** Features related to fruiting body fate **(A)** A representative curve showing total fruiting body area over time in a 7.2 mm² field size. Images are shown of aggregation at start time, peak time, and final time (24 hours), all measured as time elapsed since inoculation (t=0). The slope of the dotted line in **(A)** gives the average growth rate, a key phenotypic feature. **(B)** Representative histograms from the same time series of average fruiting body area at peak time and **(C)** final time. The mean and standard deviation of these distributions are key phenotypic features. **(D)** Five representative time lapse images show fruiting body fate, either to persist or

disappear after 24 hours of development. **(E)** Area versus time curves for each identifiable fruiting body in the same time series. For non-persistent fruiting bodies, the point of peak area is marked with a cyan circle. Two key features are the fraction of total identifiable fruiting bodies that persist (in this case, 32%, or 42 of 132), and **(F)** the standard deviation of the time at which non-persistent fruiting bodies peak in size (temporal coherence). Developmental dynamics can distinguish between time series of different homologous groups, as illustrated in **(G)** the curves for median total fruiting body area over time (quartiles bound the shaded regions, and outliers are bounded by the dotted lines). These variations are captured by 18 phenotypic features, with quantitative definitions given in Supp. Table 3.3.

The most significant phenotypic features are revealed by the makeup of the first two principal components, PC1 and PC2 (Table 3.1). These two components together account for 43% of the total variance. The constituent parts of both principal components represent a broad array of many different features, with no single outstanding feature. However, there are significant differences between PC1 and PC2. PC1 primarily represents growth rate, mean and standard deviation in fruiting body area at peak time, and mean and standard deviation in fruiting body area at final time. PC2, while sharing mean and standard deviation in area at peak time with PC1, also represents features involved with timing, including growth time, peak time, and temporal coherence. These developmental features with definitions are illustrated in Fig. 3.3. Although PC2 shares some key features with PC1, its correlations are different. For example, a high value in PC2 indicates high standard deviation in aggregate area and low growth rate, whereas a high value in PC1 indicates high standard deviation in fruiting body area and high growth rate.

PC1 Feature Name	Value
01) Final area mean	0.363
02) Final area std	0.325
03) Growth rate	0.322
04) Peak area mean	0.321
05) Peak area std	0.297
06) Maturation rate	0.279
07) Peak time	-0.256
08) Gray value % change	0.254
09) Max N falloff	-0.221
10) Fraction of evaporators	0.217
11) Lifetime std	0.209
12) Max N	0.205
13) Growth time	-0.205
14) Max mean area falloff	-0.118
15) Start time	-0.106
16) Temporal coherence	-0.077
17) Stability time	-0.062
18) Lifetime mean	-0.045

PC2 Feature Name	Value
07) Peak time	0.426
05) Peak area std	0.375
13) Growth time	0.328
04) Peak area mean	0.322
16) Temporal coherence	0.305
02) Final area std	0.298
17) Stability time	0.229
03) Growth rate	-0.195
15) Start time	0.193
11) Lifetime std	0.186
01) Final area mean	0.172
06) Gray value rate	-0.165
18) Lifetime mean	-0.141
08) Gray value % change	-0.133
10) Fraction of evaporators	-0.132
09) Max N falloff	0.086
14) Max mean area falloff	-0.058
12) Max N	-0.007

Table 3.1. Makeup of PC1 and PC2 by phenotypic feature. Each primary component is a direction or vector in the 18-dimensional phenotype space, with its makeup shared to varying degree by each feature, with either a positive (blue) or negative (red) correlation. PC1 captures the direction of greatest variance in the overall dataset, and PC2 is the direction perpendicular to PC1 that captures the next greatest amount of variance. The features most strongly represented in each primary component are those that have the greatest potential to distinguish time series phenotypically across the dataset. Each feature is numbered according to its prevalence in PC1.

Each of the four homologous gene families used in this study has a different number of genes, and therefore each family is not represented by an equal number of mutant strains. This presents a potential bias towards over-represented gene families if PCA were to be performed on the entire dataset. To address this, we performed the PCA multiple times on random samplings of the time series such that each gene family is always equally represented (see Methods). We found that across all such samplings of the full dataset of over 1000 time series, the gene families always form clusters in distinct parts of two-dimensional phenotype space (Fig. 3.4), with clusters representing a different “typical” developmental phenotype for each gene family. There was significant overlap between the clusters, so that the differences between clusters only became apparent when using a sufficiently large sample size to visualize

an estimate of the probability density. The PCA analysis therefore agrees with the manually derived developmental classifications presented in Fig. 3.2, in that mutant strains from each gene family display a full spectrum of phenotypes, but there are specific phenotypes that each family exhibits with higher frequency.

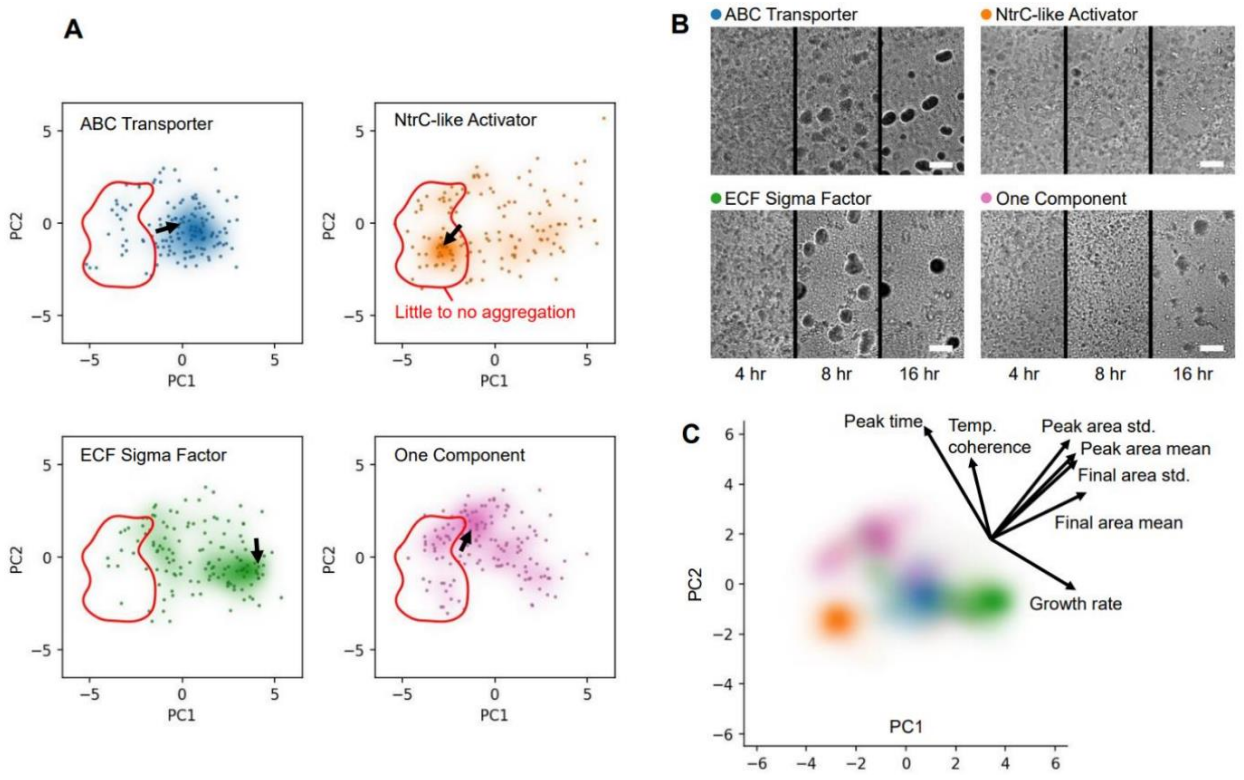


Figure 3.4. PCA reveals typical phenotypic features for each homologous family. **(A)** Each point represents a single time series, placed by phenotype according to values of PC1 and PC2. Units for PC1 and PC2 are arbitrary, but (0,0) represents average behavior. Behind points is displayed an estimation of the probability distribution function, using Gaussian kernel density estimation. Higher probability is plotted with higher opacity, revealing phenotypic clusters in each gene family. Outlined in red is a phenotypic zone containing only time series that exhibit little to no aggregation, a severe phenotype. Outside the red zone are successful fruiting body formation time series. An arrow points to a time series typical of the cluster, shown in: **(B)** The typical phenotype in each gene family cluster, illustrated with three frames from a representative time series, taken at 4 hours, 8 hours, and 16 hours after inoculation. Scale bars 100 μm. **(C)** Only the probability distribution estimates for each gene family are shown, illustrating both separation and overlap in phenotypic behavior. The directions of seven key phenotypic features are shown to indicate the coupled meaning of PC1 and PC2. Values of each feature increase in the direction of each respective arrow, with the length of the arrow indicating how much motion in feature space is caused by a fixed increase in the value of that feature, i.e. how significantly the feature is expressed by the two principal components.

The phenotypic cluster of ABC transporter mutants and ECF sigma factor mutants each show a distinct but consistently successful method of fruiting body formation, whereas One component mutants tend to vary widely phenotypically or form immature aggregates, and a plurality of NtrC-like mutants fail to form fruiting bodies entirely, as shown in the insets in Fig. 3.4B. Expanding on the comparison between the automated and manual phenotyping, we see that the phenotypic PCA clusters are not only consistent with the results of Fig. 3.2, but may also explain why a particular phenotype, such as “early aggregation” for the ECF sigma factor mutants, is expressed more often than for other homologous groups: a set of redundant genes robustly produces a minor mutant phenotype which shares multiple features in their aggregation formation dynamics, among which is an early aggregation time. Variability in phenotype can now also be measured by the spread of replicates in phenotype space, instead of needing to be classed as a separate phenotype unto itself. Representative visualizations and a basic measure of the spread of strain replicates are presented in Supp. Fig. 3.6.

There are two generic features of the way in which phenotype clusters form for each gene family. First, the separation between clusters indicates that mutations in each homologous group affect phenotype in a distinct way. Second, the size of each cluster’s individual peak points to how often each similar phenotype is expressed within the homologous group. Clusters in the PCA output were shown to be both separated and small with high statistical significance (with p-values of 10^{-5} and 0.0083 respectively) when compared to random groupings of time series instead of grouping by homologous group (Supp. Fig. 3.5).

By manually reviewing each time series that fell near a phenotypic cluster, we determined that each represented a coherent overall phenotype with only a small number of outliers.

Significantly, a developmental phenotype that could be considered severe, little to no aggregate formation, was shown to vary throughout the dataset, and the NtrC-like activators' typical phenotype was shown to be a distinct form of failure to aggregate in which there was still bacterial motility, but little to no departure from a uniform swarm layer. Details on each typical phenotype and the metric values that distinguish them are available in the SI.

Our decision to characterize developmental phenotype via aggregate dynamics may have had a negative impact on our ability to differentiate the severe phenotypes that exhibit little to no aggregation. There is an identifiable phenotypic zone in the PCA output that represents little to no aggregation, as outlined in red in Fig. 3.4A. However, even within this zone, different behaviors are still distinguishable, and there is separation between homologous gene families. This is shown by the NtrC-like activators and One component mutants tending to occupy different phenotypic territory within the red outline. Differences in the dynamics of small, transient aggregates still hold significance in identifying typical behaviors across gene families.

Our observation that PCA of phenotypic metrics recapitulates homologous families of mutant strains indicates that phenotypic similarity and sequence identity are positively correlated across a genome, but this correlation does not scale down to fine-grained genetic differences. We observe that, within a homologous gene family, pairwise sequence similarity in this dataset correlates only weakly with phenotypic similarity, and therefore it is not an effective predictor of phenotype between pairs of genes within the same family (Supp. Fig. 3.6), consistent with previous findings [121]. The phenotype clusters for each gene family are populated by replicates of many mutant strains in that family, both the genetically similar and dissimilar. For the ABC transporters, 56% of strains (20 of 36) have replicates within the phenotypic cluster.

This is also true for 24% (8 of 34) of NtrC-like activator strains, 39% (15 of 38) of ECF sigma factor strains, and 64% (16 of 25) of One component strains. Fine-grained genetic similarity is not necessary for redundancy to exist, and redundant gene networks may be quite large.

Taken together, Fig. 3.4 demonstrates that different homologous gene groups are likely to contain redundant gene networks, which each produce a distinct minor mutant phenotype.

3.3 Discussion

Goldman et al proposed that the expansion of the *M. xanthus* genome due primarily to duplication and divergence has led to an enrichment of some gene families, especially those involved in cell signaling and transcriptional regulation, over others [127]. This asymmetry of enrichment is notable because it implies a purpose for the expansion of specific gene families. We propose at least part of that purpose is to create functional redundancy networks that act as buffers to stabilize *M. xanthus* development (i.e. robustness). In this study we confirm that *M. xanthus* development meets the criteria of a robust phenotype, by showing that more than 250 mutant strains with disruptions in genes that are part of four large homologous families display severe developmental phenotypes very infrequently. We then provide support for our hypotheses regarding the existence of redundancy networks by quantifying the phenotypes of these mutant strains and observing that using PCA to map phenotypic feature space also clusters the mutant strains according to the four homologous gene families.

Paralogs within a gene family may share similar molecular mechanisms, but they are expected to have different biological functions. For example, the ABC transporters in *M. xanthus* all perform active transport across membranes, but they are expected to transport different

substrates. This would mean that a disruption of any one ABC transporter would cause a change in developmental phenotype specific to its substrate. There is no obvious reason why mutant strains of the ABC transporters would display similar changes in phenotype unless there is significant functional redundancy between transporters. There is also no obvious reason why the phenotypic similarities would include a plurality of a large homologous gene family unless the functional redundancy is widely distributed.

When a group of functionally redundant genes absorbs the effect of one member's disruption with low overall stress on the system, the impact on phenotype is more subtle. The ABC Transporter and ECF sigma factor gene families exemplify this, as there are very few single gene knockouts that result in severe phenotypes (Fig. 3.2E). The phenotypes of those genes cluster sharply by homologous family in the PCA feature space (Fig. 3.4A), meaning that both gene families display a typical phenotype that is different from the others. Plausible biological explanations for this type of widely distributed functional redundancy can be made. Many ABC Transporters, due to varying homology in periplasmic and substrate-binding domains across the gene family, may be able to transport similar and/or overlapping substrates [131,132], mitigating the effect of many of the mutations in this gene family and most often producing like wild-type phenotypes. Similarly, robustness has been shown to be encoded in transcriptional regulatory networks by alternative pathways [133], and though some studies suggest that alternative sigma factors display minimal crosstalk [134], it is not unprecedented for there to be overlap in the regulation of genes by multiple ECF sigma factors, creating networks of integrated regulation [135–137]. Our data indicate that *M. xanthus* may use such networks of crosstalk among ECF sigma factors to coordinate transcription in response to extracellular

signals, and that this may involve integration from many redundant or parallel pathways, ultimately leading to earlier aggregation initiation time and faster fruiting body growth rate than we see in wild-type for the majority of ECF sigma factor mutants.

In contrast, NtrC-like activator mutants show more severe phenotypes and cluster in a region where strains do not form fruiting bodies (Fig. 3.4). Though it could be argued that a non-aggregating strain indicates a lack of redundancy for the mutated gene, this seems unlikely given that the NtrC-like activators fail to produce aggregates in a way that is distinct from non-aggregating mutants in other gene families (Fig. 3.4A—region outlined in red). This again points to the idea of networks of redundancy, but highlights that there can be a cost to redundancy in some situations. Extensive research has shown that kinase-response regulator pairs tend to be very insulated with limited crosstalk, and that this feature rapidly evolves in newly-duplicated two-component systems [138,139]. NtrC-like activators and other bacterial DNA-binding response regulators have high affinity interactions with their cognate kinases, and crosstalk generally increases noise and decreases the overall response of the system to the incoming signal [140]. The specificity of response regulators for phosphorylation by their cognate kinases is governed primarily by molecular recognition, though these proteins can be very sequence similar, and by maintaining a relatively high abundance of response regulator relative to its cognate kinase within a cell to prevent unwanted phosphorylation [138,141,142]. Taken together, this indicates that mutations to response regulators, like those that we have introduced in the NtrC-like proteins in this study, might lead to a situation where there is a high concentration of phosphorylated kinase in the absence of its highest affinity interaction partner, allowing the cognate kinase to phosphorylate structurally similar non-target response

regulators and inappropriately initiate those signaling cascades. This model would explain why so many of the NtrC-like activator mutations produced severe phenotypes that fail to form fruiting bodies in the same way and highlights that redundancy due to gene duplication can have negative consequences without proper insulation.

Rarely is a typical phenotype expressed a majority of the time within knockouts of homologous group – instead, the typical phenotype represents the plurality behavior. Replicates even of the same strain are seen to straddle multiple zones in the PCA feature space. This is particularly evident in the One component gene family, which was observed to have the greatest variation in phenotype from replicate to replicate (Supp. Fig. 3.6). The same mutant strain, for example, can yield a phenotype indistinguishable from wild-type in one replicate, and make immature aggregates or fail to aggregate entirely in other replicates. This variability might indicate that the impact of mutation on these functional networks increases sensitivity to stochastic fluctuations within the cellular environment that contribute to a tendency toward one phenotypic fate or another.

We do not propose that phenotypic similarity serves as a strong indicator of functional redundancy. There are almost certainly insignificant associations in the PCA feature space. For example, there are a small number of ABC transporter mutants positioned within the NtrC-like activator cluster. We do not propose that these genes are functionally redundant with the majority of NtrC-like activators, but we might suggest that they are less likely to be functionally redundant with those in the ABC transporter cluster. It is also possible that any group of completely unrelated genes could have some degree of functional redundancy, but this represents a background level or lower threshold of observable redundancy. We have shown

that the redundancy we observe is significantly above that background by measuring the phenotypic clustering of random groupings of genes from the various homologous groups. We are sure there are many forms and many degrees of functional redundancy that are not represented by this PCA, but it does reveal a widely distributed functional redundancy above a background threshold.

Extensive progress has been made in recent years in looking at large-scale studies of digenic and even trigenic interactions and redundancies that affect phenotypic robustness and fitness [118,143]. Others have begun to disentangle the relationship between subsets of multigene families and their roles in redundancy [144]. We add to this growing body of literature by exploring functional redundancy in large, homologous gene families of *M. xanthus*. We make no claims about functional redundancy between specific genes in *M. xanthus*. Rather, we seek to define the scale and distributive nature of redundancy networks that include these large gene families, demonstrating that redundancy networks are not necessarily limited to a few of the closest paralogs; they may include dozens or hundreds of genes.

Instead of trying to quantify the direct effect of mutations on fitness by measuring a single variable such as growth, we chose to measure multiple aspects of a complex development process. While our method requires the collection of more data per strain than a synthetic genetic array, for example, it has the ability to detect more subtle phenotypes that may not have strong implications for fitness but can still inform studies of redundancy. Since many single gene disruptions have such subtle phenotypes, and since we propose that extensive redundancy networks protect an organism from the fitness costs of mutation, we chose

phenotypic similarity, rather than overall effect on fitness, to assess the extent of functional redundancy within gene families.

Our results highlight the importance of considering the nature and extent of redundancy when making claims regarding interactions between genotype and phenotype. Gene families can have high degrees of functional entanglement that may mitigate the impact of mutation, so that quantifying even minor deviations in phenotype may allow for the recognition of patterns; if mutations within a redundancy network produce similar phenotypes, then subtle changes in phenotype have the potential to inform annotation. For example, a gene of unknown function displaying a subtle phenotype similar to that of genes of known function could provide evidence that the unknown gene is part of a redundancy network. Our image analysis pipeline can be extended to future studies of *M. xanthus*, even under differing experimental conditions, for automated extraction of phenotypic features. Further, our dataset can be used to probe whether there are patterns in amino acid sequence homology that lead to functional redundancy by comparing the sequences of genes that are located within the family cluster on the PCA to those that are located outside the cluster and are presumably non-redundant. Underscoring all these results is the observation that without a sufficiently large collection of mutants and replicates, functional redundancy does not present itself clearly enough to be recognized.

3.4 Conclusions

This work provides evidence for the existence of large networks of redundant genes as a means by which an organism such as *Myxococcus xanthus* can execute complex multicellular social

behaviors robust to perturbations to gene function. We observe subtle deviations in phenotype, a distinct set for each homologous gene family, that present when knocking out any one gene within these redundancy groups. These subtle deviations are measurable due to the large number of time series included in our full dataset and the quantitative detail of the extracted phenotypic information, which in combination necessitate the automated analysis pipeline we have developed.

3.5 Methods

Strains and Culture Conditions

Myxococcus xanthus strain DK1622 was used as the wild-type for this study. All 265 mutant strains in the ABC Transporter, ECF sigma factor, NtrC-like activator, and One Component Signal Transduction System families (Supp. Table 3.4) were created using plasmid insertion via homologous recombination as previously described [21,145] and modified by Yan et al [56]. Briefly, 400-600bp internal fragments of each gene were PCR amplified and ligated into pCR®2.1-TOPO [Invitrogen]. The plasmids were amplified in *E. coli* before isolation and electroporation into *M. xanthus* DK1622, where the plasmid incorporates into the *M. xanthus* genome via the homologous region on the plasmid. PCR verification was used to confirm the location of each insertion.

Cells were grown overnight in CTTYE (1% Casein Peptone (Remel, San Diego, CA, USA), 0.5% Bacto Yeast Extract (BD Biosciences, Franklin Lakes, NJ, USA), 10 mM Tris (pH 8.0), 1 mM $KH(H_2)PO_4$, 8 mM $MgSO_4$) with vigorous shaking at 32°C. Cultures of mutant strains were supplemented with 40µg/mL kanamycin. Cells were centrifuged to remove the nutrient broth,

washed in TPM buffer (10 mM Tris (pH 7.6), 1 mM $KH(H_2)PO_4$, 8 mM $MgSO_4$), and resuspended to a final concentration of $5x10^9$ cells/mL. For development assays, approximately $2.5x10^7$ cells were spotted onto TPM agar slide complexes, as previously described [146].

Imaging

Development assays for wild-type and mutant strains were carried out on TPM starvation agar slide complexes for 24 hours, with approximately three replicates per strain. Though it can take multiple days for cells within fruiting bodies to fully differentiate into spores, we generated time series of only the first 24 hours of development because wild-type cells show little to no observable change in fruiting body morphology, count, or behavior following this period at the magnification used. Time-lapse grayscale images were captured every 60 seconds under 4x magnification with a Nikon Eclipse E-400 microscope [Nikon Instruments] and SPOT Insight camera. ImageJ was used for processing the .TIFF images into time series for analysis.

Multidimensional scaling of gene sequence dissimilarity

Amino acid sequences for the four homologous families were retrieved from NCBI and imported into the Multiple Sequence Alignment tool in Clustal Omega [147], generating a percent identity matrix for all 265 proteins. This was then converted to a percent dissimilarity matrix and used as the input for the Classical Multidimensional Scaling package in R to generate plotting coordinates in two dimensions. Then Gaussian kernel density estimation was used to plot an estimate of the probability distribution function (plotted with increased opacity to represent higher probability) to guide the eye in identifying sub-clusters of similar genes within each paralogous group.

Manual phenotyping

Manual preliminary phenotyping of the mutant strains in this study was performed using the time series described above. We will refer to mature aggregates as fruiting bodies for simplicity, though we did not test sporulation efficiency in this study. First, strains that failed to produce fruiting bodies at all within 24 hours across all replicates were labeled “no aggregation” mutants. Strains that formed initial aggregates that disassociated completely before the 24-hour mark were labeled as “fall apart”. Some strains, labeled “aggregate-reaggregate”, formed aggregates that initially fell apart, but new aggregates were formed that persisted and looked similar to wild-type by the endpoint of the time series.

To qualitatively determine the start time of aggregation, the time series were observed in sliding windows of 25 minutes to identify the window where initial aggregates were first formed. The average start time of 22 wild-type replicates was used for comparison, and mutant strains that had start times outside of one standard deviation of the mean of wild-type were considered either “early” or “late” aggregation mutants.

Wild-type fruiting bodies at 24 hours appear almost black in color and are roughly circular in brightfield images. Any strains that appeared to have these characteristics and initiated aggregation within the same window as wild-type were classified like-wild-type (LWT). Strains that initiated aggregation at a normal time but didn’t develop aggregates that were as dark in gray value as wild-type were labeled as “immature aggregation” mutants. Finally, mutants that did not display consistent phenotypes across replicates were classified as “variable”. A table of all mutant strains used in the study, as well as their manually assigned phenotype, can be found in Supp. Table 3.4.

Automated phenotyping

Phenotype was automatically quantified for the mutant strains in this study by running 144 individual .TIFF images (ten minutes between each frame over 24 hours of total development) from each time series through a custom Python image processing and analysis pipeline to identify in each frame which pixels could belong to a fruiting body, based on their gray value. The information for the position and geometry of each aggregate was filtered to remove noise and spurious aggregates. This information was then collected for the entire time series to track individual fruiting bodies over time, revealing their fate and dynamics. This detailed data summary for each time series then had a list of eighteen specific numbers extracted from it, each of which captures one overall feature, such as average growth rate or the average size of final fruiting bodies. The values of these eighteen metrics together (a phenotypic vector) constitute the phenotype profile for that time series. The full details of the image processing pipeline and all phenotypic metrics are available in the Supplementary Materials.

A selection of 133 mutant time series were chosen at random from each paralogous group so as not to weigh any paralogous group more than the other. The phenotypic vector for each time series was calculated, and the values of each metric were shifted by a constant amount and scaled by a constant factor so that across the dataset, each metric had a mean of zero and a variance of one. This ensured that one metric would not supersede the others simply due to the magnitude of its units. PCA was performed on this normalized dataset to extract the two combinations of metrics, PC1 and PC2, that captured the most variation across the dataset.

The phenotypic clusters were revealed by plotting each time series as a point in the PC1 vs. PC2 phenotype space and then estimating the probability density for each homologous group via

Gaussian kernel density estimation. Essentially, a Gaussian blur was applied to the points, and areas of greater overlap were colored with higher opacity, as shown in Fig. 3.4. The width of the smoothing kernel was chosen to be the smallest value that could preserve the shape of the probability density for different equally sized subsamples from each homologous group.

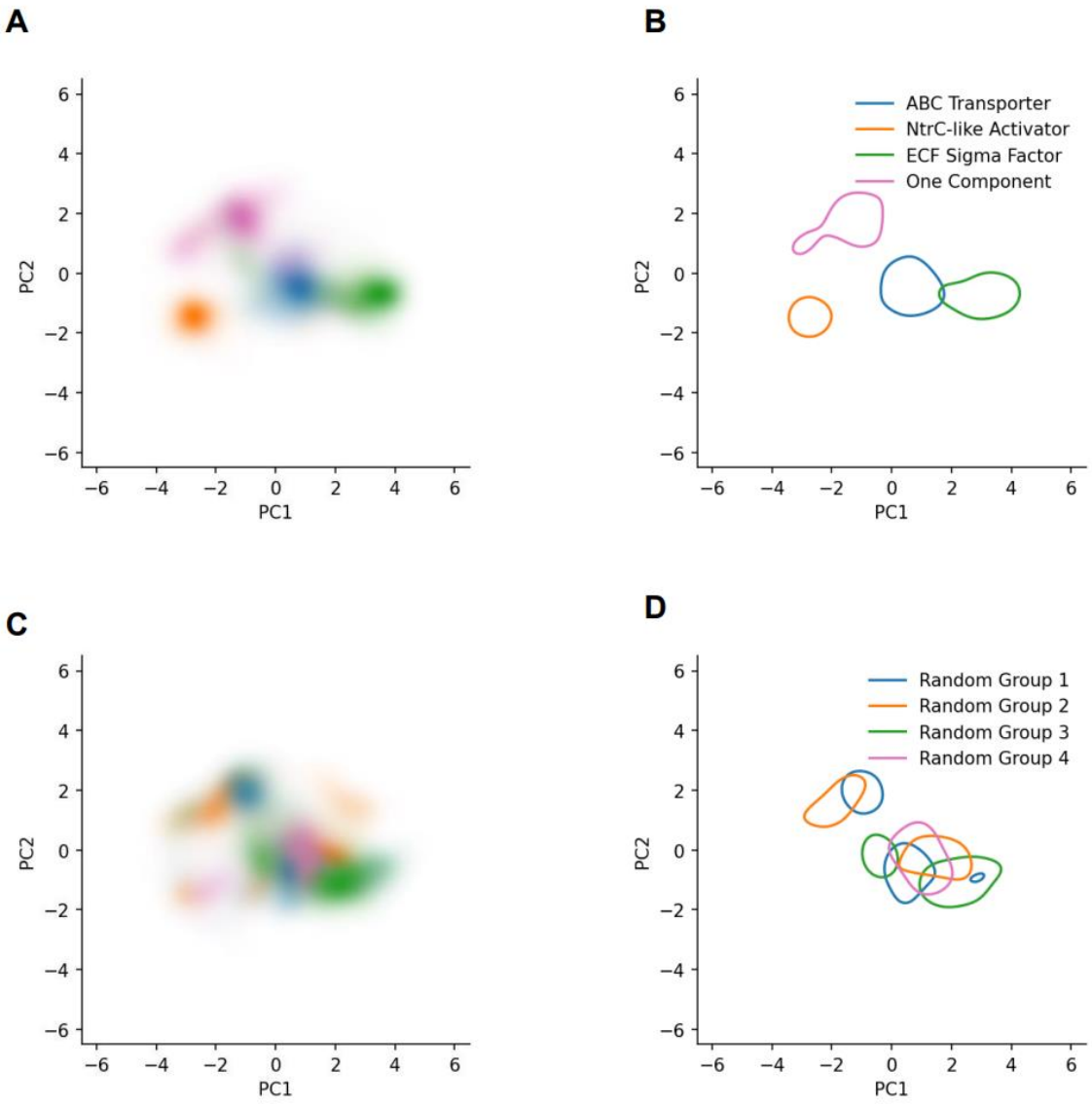
The statistical test used to generate the p-values for average cluster separation and average cluster size was a form of bootstrapping which started with the PC1 and PC2 coordinates of each point shown in the data sample of Fig. 3.4. Each point was randomly reassigned one of four arbitrary families in such a way that replicates of the same strain were all assigned the same family. A new Gaussian kernel density estimation was performed to approximate the probability density of each family in PCA phenotype space. The contour representing 75% of the maximum value of the estimated probability distribution function was then extracted, with cluster separation being the average across all pairings of families of the centroid-to-centroid distance between contours, and cluster size quantified by the average across families of the radius of gyration of each contour, i.e. the root mean square distance of each contour's points from its centroid. Each p-value was calculated as the fraction of the random groupings that had a greater average separation or smaller average size than that of the original data grouped by the actual gene families.

3.6 Supplementary Materials

Statistical testing and controls

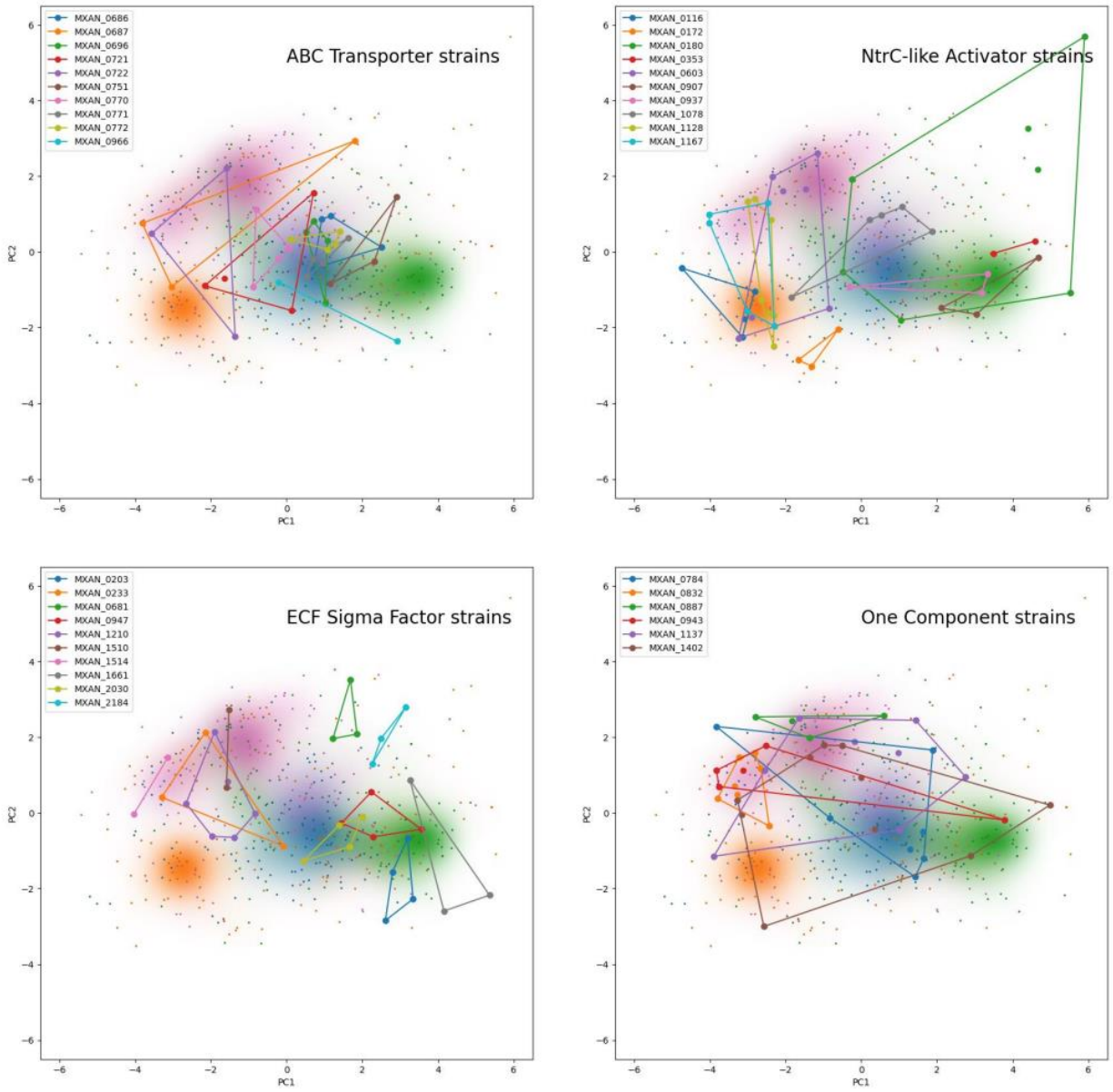
The results presented in Fig. 3.4, namely the separation of phenotypic clusters for each homologous gene family, were tested for statistical significance by recreating the estimated

probability density functions for random groupings instead of the actual gene family and observing the resulting sharpness of and separation between peaks in each grouping's probability density function (see Methods). To illustrate the difference in phenotypic peak between these random groupings and the actual gene family groupings expected to correspond to redundant gene groups, we present the results of a representative random grouping in Supp. Fig. 3.5.



Supplementary Figure 3.5. Phenotypic clusters arise robustly from homologous gene families as compared to random groupings of mutant strains. **(A)** Reproduced from Figure 3.4, each of the four gene families produces a distinct phenotypic cluster when plotting the estimated probability distribution function for that family (using Gaussian kernel density estimation) in phenotype space. **(B)** A contour is shown for each gene family where the estimated probability distribution function is at 75% of its maximum value, and the geometry of that contour is used to quantify the width of the cluster and its separation from other clusters. **(C)** The same data used in Figure 3.4 was regrouped into four random groupings, and the PCA and probability density function estimates were repeated, showing much more incoherent phenotype clusters. **(D)** The corresponding contours for the four random groupings show less separation and less sharpness than phenotypic clusters based on homologous gene groups. Both (C) and (D) come from one representative random grouping, many of which were made to calculate the p-values for cluster separation and sharpness reported in Results.

Replicates of the same strain can vary in phenotype. Several plots showing phenotypic spread for a few representative strains are included in Supp. Fig. 3.6.



Supplementary Figure 3.6. Replicates of the same strain can vary in phenotype. Reproduced for context from Figure 3.4 are the phenotypic scatterplot resulting from the PCA (where each point is a time series, plotted nearby other time series that are phenotypically similar) and the superimposed probability distribution functions for each of the homologous gene families: ABC transporters in blue, NtrC-like activators in orange, ECF sigma factors in green, and One component in pink. Each subplot includes all replicates of a few representative strains, where the replicates of each strain are represented in a single color and drawn with a bounding polygon to aid the eye. Replicate-to-replicate variation is larger or smaller depending on strain and to which homologous family the strain belongs.

A metric for replicate-to-replicate phenotypic spread of a specific strain is the sample standard deviation s generalized to two dimensions

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}((x_i - \bar{x})^2 + (y_i - \bar{y})^2)}$$

where n is the number of replicate points, x_i and y_i are the coordinates of the ith replicate point, and $\bar{x}$ and $\bar{y}$ are the means of each coordinate across the replicate points, i.e. the centroid coordinates. In this case, the x and y coordinates are the value of PC1 and PC2 respectively.

Supp. Table 3.2 summarizes the mean replicate-to-replicate phenotype spread averaged over strain for each gene family using the metric s, with errors given by the standard error of the means.

Gene family	**Mean replicate-to-replicate spread** s
ABC Transporters	1.42 ± 0.12
NtrC-like Activators	1.57 ± 0.14
ECF Sigma Factors	1.31 ± 0.09
One Component	1.94 ± 0.18

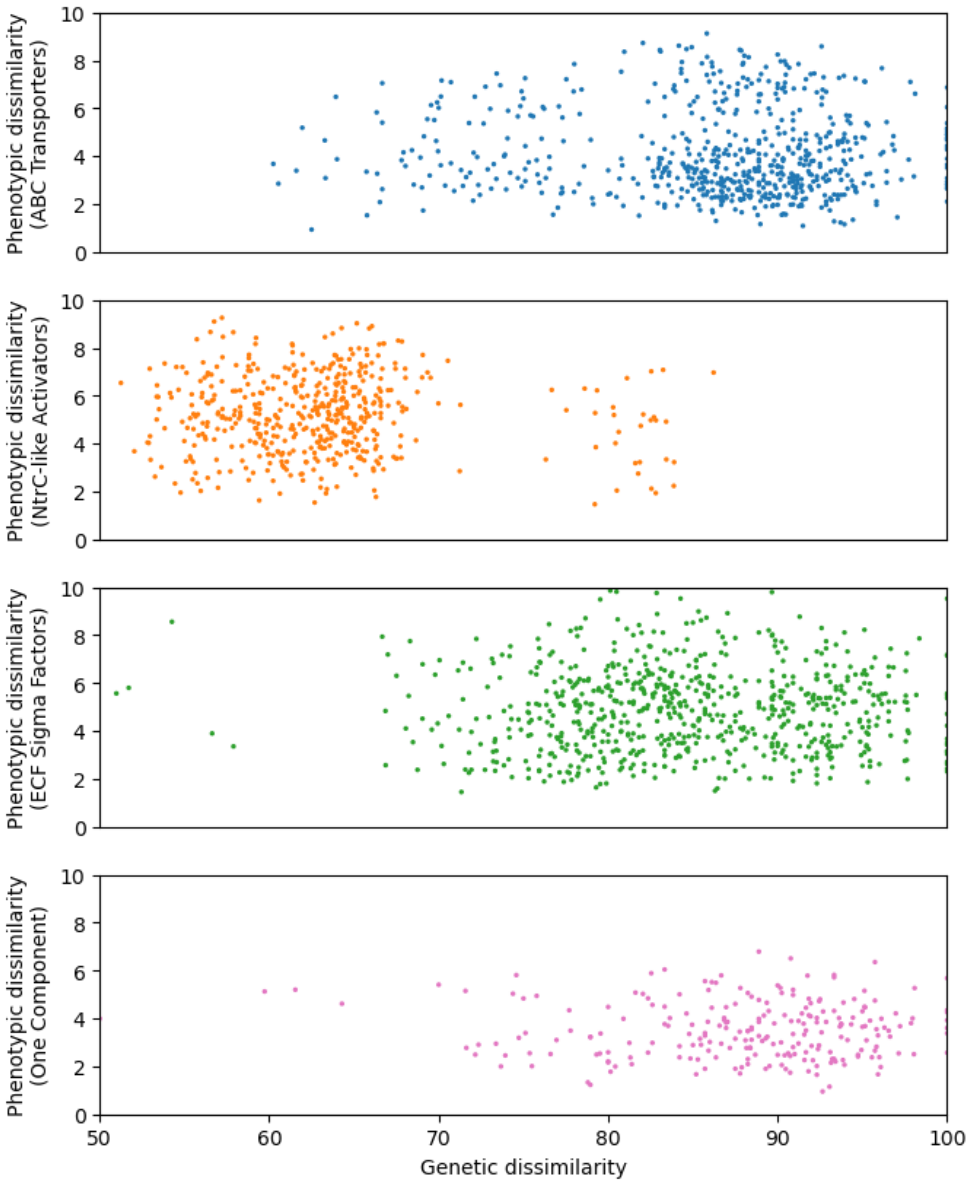
Supplementary Table 3.2. A summary of the average replicate-to-replicate spread for each homologous gene family, with errors given by the standard error of the means. This spread is illustrated for some representative strains in Supp. Fig. 3.6

This indicates a statistically significant difference for replicate-to-replicate phenotype spread between One Component strains and ABC Transporter strains (p = 0.024), and between One Component strains and ECF Sigma Factor strains (p = 0.004) according to a two-sided Welch's t-test.

Correlation of genetic differences with phenotypic differences

When comparing the genetic similarity and phenotypic similarity of the mutant strains used in this study, little correlation was found, as illustrated in Supp. Fig. 3.7.

For each homologous gene group, all unique pairings of different strains were plotted using genetic difference on the x-axis and phenotypic difference on the y-axis. Genetic difference was quantified using Clustal Omega Multiple Sequence Alignment [147], and phenotypic difference was calculated as the Euclidean distance between two 18-feature points, averaging the feature values over all of the replicates for that strain. These plots show that close genetic similarity of two mutant strains is not necessary for and in fact poorly predicts phenotypic similarity. Instead, we infer that phenotypic similarity is roughly equivalent across all mutants in a group of redundant genes.



Supplementary Figure 3.7. Genetic similarity is not an effective predictor of phenotypic similarity within a homologous gene family. For each of the four gene families analyzed, each point represents a unique pairing of two strains. Phenotypic dissimilarity is quantified by Euclidean distance in 18-dimensional feature space, where feature values are represented by averages over all replicates for that strain. Genetic dissimilarity is quantified by comparison of base pairs using Clustal Omega Multiple Sequence Alignment. Within each of the four gene families, phenotypic dissimilarity and genetic dissimilarity to not correlate.

Description of typical phenotype for each homologous gene family

A manual review of time series near the phenotypic cluster for each respective gene family (Fig. 3.4) was performed to describe the typical behavior. This summary is with respect to only time

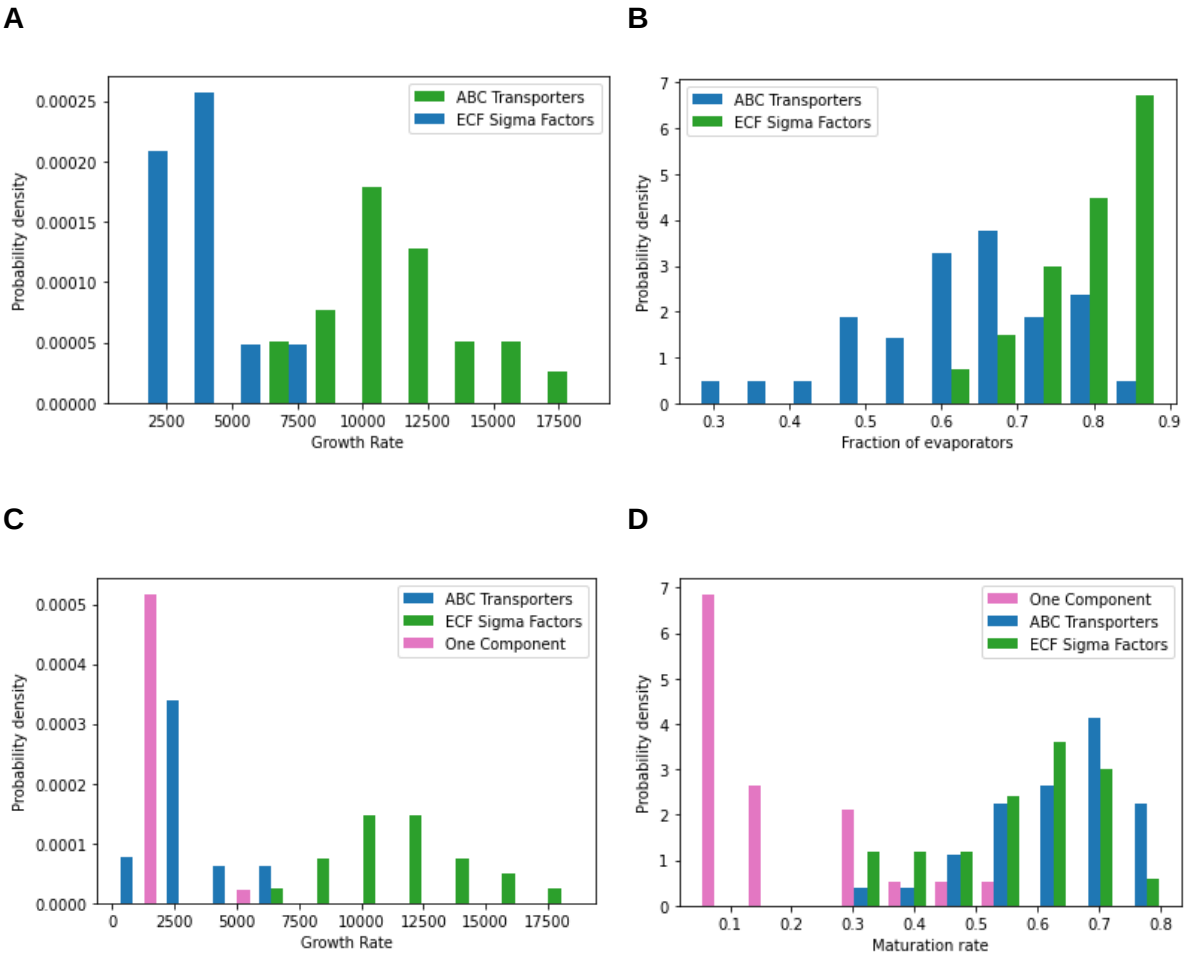
series that fell within the contour of 75% of the maximum of the probability distribution function for each gene family.

Both the ABC transporter and ECF sigma factor mutants showed similar behaviors both with rapid formation and darkening of aggregates. Their differences lie in the number of persistent fruiting bodies and the rates of formation. Fruiting bodies of ECF sigma factor mutants form faster than the those of ABC transporter mutants, captured by growth rate.

The time series in the ECF sigma factor cluster also have a higher fraction of their fruiting bodies fail to persist. So, although they are fast to form, these aggregates have more of a tendency to either be absorbed by their neighboring fruiting bodies or not survive at all.

Nearly all of the time series in the NtrC-like activator cluster fail to form fruiting bodies. While the bacteria do move around and occasionally form aggregates, these never completely darken or stabilize and most evaporate. This tendency to fail to aggregate is most directly captured quantitatively in the small final number of persistent fruiting bodies and in the low growth rate of these time series.

One component mutants near the phenotypic cluster were observed to have the ability to form fruiting bodies but not darken significantly, remaining immature. They also appear to be less circular than the fruiting bodies of ABC transporters or ECF sigma factor mutants. The One component time series also showed signs of struggle when forming – they took much longer than the other successfully aggregating gene families to form their persistent fruiting bodies. The individual phenotypic features that support these observations are included in Supp. Fig. 3.8.



Supplementary Figure 3.8. Quantitative comparison of the behavior in phenotypic clusters for each gene family. **(A)** Histogram of growth rates. ECF sigma factor mutants form aggregates faster than ABC transporter mutants. **(B)** Histogram of fraction of fruiting bodies that fail to persist. ECF sigma factor mutants have more evaporating fruiting bodies than ABC transporter mutants. **(C)** Histogram of growth rates between three gene families with successful fruiting body formation. One component mutants form fruiting bodies the slowest of the three displayed gene families. **(D)** Histogram of maturation rates (i.e. the maximum rate of darkening during fruiting body maturation). One component mutants do not darken or mature at a rate much slower than ABC transporter mutants or ECF sigma factor mutants.

Image processing

Custom Python code was written for this analysis, available on Github

(https://github.com/masp01/SU-myxo-aggregate-tracking). Using the Python implementation

of OpenCV [112], each raw frame is put through the following image processing steps to identify the size, shape, and position of each fruiting body:

1. Non-local means denoising (`cv2.fastNlMeansDenoising`)

   To remove background noise, a 7 pixel wide template window is moved over the image to find regions that visually match (typically uniform, noisy regions). This search is done within a 21 pixel distance of each patch of the image. The gray value of each pixel is replaced with the average gray value of pixels in matching regions, smoothing over noise while keeping boundaries distinct. The smoothing strength was chosen at a constant value of 70 after manually testing parameter values for many images. Template window and search sizes are standard and were not tuned.

2. Adaptive thresholding (`cv2.adaptiveThreshold`)

   To identify locally dark regions that should belong to fruiting bodies, the gray value of each pixel is compared to the average gray value of its neighbors within a block 101 pixels (145 µm) wide. 8-bit pixels (gray value from 0 – 255) that have a gray value at least 20 below this local average are marked white. All other pixels are marked black, creating a binary image. Parameters were chosen after manually testing with many images and are robust enough to be used across the entire dataset.

3. Morphological opening (`cv2.morphologyEx`)

   A circular kernel 5 pixels (7.2 µm) wide is moved over the binary image. Any feature covered entirely by the kernel is removed. This reduces single-pixel noise.

4. Contour identification (`cv2.findContours`)

   Contiguous regions of white pixels are automatically identified in the binary image. A list is compiled of the x,y coordinates of the pixels on the boundary of each such region. This gives both a count of total candidate fruiting bodies and the geometry of their boundary.

At this point, a list of features has been identified, some of which are genuine fruiting bodies, and some of which are noise or spurious aggregates. The contour of each feature is measured for the x,y coordinates of its center, its area A, perimeter P, and average gray value. The circularity $4\pi A/P^2$, is also calculated. It captures the elongation of the fruiting bodies and ranges from 0 (completely flat) to 1 (perfectly circular).

Tracking fruiting bodies and filtering

Once all the frames of a time series have been processed, the Python package Trackpy [148] is used to assign an ID to each feature that tracks it over time. It is at this point that filtering is done to remove spurious features:

1. Minimum area filter

   Features that are smaller than 576 μm² are ignored. This is the smallest fruiting body size that is distinguishable from noise at 4X magnification

2. Maximum gray value filter

   Features with an average gray value above 200 (max 255) are considered too bright to be fruiting bodies and are ignored.

3. Formation time filter

   Features that appear before 100 minutes have elapsed are incidental initial

aggregates, and not genuine fruiting bodies that have assembled over time. In no time series did a new aggregate form in less than 100 minutes. These incidental aggregates are tracked over time and ignored in all frames in which they appear.

4. Category filter

The area dynamics of each remaining fruiting body are considered to see if the fruiting body persists to the end of the time series (persistors) or if it vanishes smoothly (evaporators). Persistors with an average circularity below 0.5 are typically noise and are ignored. Smoothly vanishing is defined as starting with an area less than max area and then decreasing from maximum area by at least 25% by the final frame of the time series. Evaporators with centers within (14.4 μm of the edge of the frame or with an average circularity below 0.5 are considered noise and ignored. Any feature that cannot be categorized as a persistor or evaporator is assumed to be spurious or contain dynamics errors and is ignored.

Feature extraction

The data for each time series is then analyzed to measure the following quantitative features, each a single number summarizing one aspect of the time series. Measurements are taken over a 7.2 mm^2 field size.

Supplementary Table 3.3. Enumeration of all quantitative features used in the automated phenotype analysis

	Feature name	Description	Formula
1	Start time	Elapsed time from inoculation to the beginning of visible aggregation	When at least 10 fruiting bodies grow larger than 1000 μm^2 in area
2	Peak time	Elapsed time from inoculation to the peak of visible aggregation	When total fruiting body area reaches a maximum
3	Stability time	Elapsed time from inoculation to overall aggregate stability	When the number of fruiting bodies changes by less than three per hour (24 hours maximum)

4	Growth time	Duration of initial growth phase	Peak time minus start time
5	Growth rate	Average rate of total area increase during growth phase	Change in total area divided by change in time between start time and peak time
6	Peak average area	The average fruiting body area at peak time	
7	Peak area std	The standard deviation of fruiting body area at peak time	
8	Final average area	The average fruiting body area at the moment 24 hours after inoculation	
9	Final area std	The standard deviation of fruiting body area 24 hours post-inoculation	
10	Gray value % change	Percent difference between minimum and maximum average gray value (only for persistent fruiting bodies)	
11	Maturation rate	Maximum slope of gray value vs. time curve for persistent fruiting bodies	
12	Temporal coherence	How closely in time each evaporating fruiting body reaches its maximum area before starting to shrink	The standard deviation of the distribution of the time of maximum area for evaporators
13	Fraction of evaporators	Total number of evaporators divided by total number of evaporators plus persistors	
14	Maximum number	Total number of fruiting bodies at peak time	
15	Average lifetime	The elapsed time between an evaporator's first and final moment above the minimum area threshold, averaged over all evaporators	
16	Std lifetime	Standard deviation of the elapsed times between each evaporator's first and final moment above the minimum area threshold	
17	Maximum average area falloff		Most negative slope of average area vs. time curve
18	Maximum number falloff		Most negative slope of number vs. time curve

Supplementary Table 3.4. List of strains used in this study, listed by MXAN number, followed by the manual phenotype classification associated with Fig. 3.2, gene family, and citation for creation of the specific strain used.

Strain	**Phenotypic Classification**	**Gene Family**	**Strain Creation**
DK1622	Wild-type	N/A	
MXAN_0035	No aggregation	ABC Transporter	[56]
MXAN_0036	LWT	ABC Transporter	[56]
MXAN_0037	Late aggregation	ABC Transporter	[56]
MXAN_0069	Early aggregation	One Component	This study
MXAN_0079	Variable	One Component	[149]
MXAN_0090	Late aggregation	One Component	This study

MXAN_0107	Early aggregation	ABC Transporter	[56]
MXAN_0108	LWT	ABC Transporter	[56]
MXAN_0116	No aggregation	NtrC-Like Activators	This study
MXAN_0172	Fall apart	NtrC-Like Activators	[149]
MXAN_0180	LWT	NtrC-Like Activators	This study
MXAN_0203	Early aggregation	ECF Sigma Factors	This study
MXAN_0213	Variable	One Component	[149]
MXAN_0214	Variable	One Component	This study
MXAN_0233	Immature aggregates	ECF Sigma Factors	This study
MXAN_0250	Early aggregation	ABC Transporter	[56]
MXAN_0251	LWT	ABC Transporter	[56]
MXAN_0353	Early aggregation	NtrC-Like Activators	This study
MXAN_0387	Late aggregation	One Component	This study
MXAN_0445	Early aggregation	One Component	This study
MXAN_0502	Variable	One Component	[149]
MXAN_0553	LWT	ABC Transporter	[56]
MXAN_0554	LWT	ABC Transporter	[56]
MXAN_0556	LWT	One Component	This study
MXAN_0559	Variable	ABC Transporter	[56]
MXAN_0596	Immature aggregates	ABC Transporter	[56]
MXAN_0597	LWT	ABC Transporter	[56]
MXAN_0603	Late aggregation	NtrC-Like Activators	[149]
MXAN_0622	LWT	ABC Transporter	[56]
MXAN_0627	Early aggregation	One Component	This study
MXAN_0629	Immature aggregates	ABC Transporter	[56]
MXAN_0654	LWT	One Component	This study
MXAN_0665	Variable	One Component	[149]
MXAN_0681	Aggregate-reaggregate	ECF Sigma Factors	[149]
MXAN_0684	Late aggregation	ABC Transporter	[56]
MXAN_0685	No aggregation	ABC Transporter	[56]
MXAN_0686	LWT	ABC Transporter	[56]
MXAN_0687	Late aggregation	ABC Transporter	[56]
MXAN_0696	Early aggregation	ABC Transporter	[56]
MXAN_0707	Variable	One Component	This study
MXAN_0721	LWT	ABC Transporter	[56]
MXAN_0722	LWT	ABC Transporter	[56]
MXAN_0748	LWT	One Component	This study
MXAN_0751	LWT	ABC Transporter	[56]
MXAN_0770	LWT	ABC Transporter	[56]
MXAN_0771	LWT	ABC Transporter	[56]
MXAN_0772	LWT	ABC Transporter	[56]

MXAN_0832	Variable	One Component	This study
MXAN_0887	Immature aggregates	One Component	This study
MXAN_0907	Early aggregation	NtrC-Like Activators	This study
MXAN_0937	LWT	NtrC-Like Activators	This study
MXAN_0943	Variable	One Component	This study
MXAN_0947	LWT	ECF Sigma Factors	This study
MXAN_0966	Early aggregation	ABC Transporter	[56]
MXAN_0967	LWT	ABC Transporter	[56]
MXAN_0968	Late aggregation	ABC Transporter	[56]
MXAN_0995	LWT	ABC Transporter	[56]
MXAN_1078	Immature aggregates	NtrC-Like Activators	[149]
MXAN_1097	LWT	ABC Transporter	[56]
MXAN_1124	LWT	ABC Transporter	[56]
MXAN_1128	Variable	NtrC-Like Activators	[149]
MXAN_1137	Variable	One Component	This study
MXAN_1151	Variable	ABC Transporter	[56]
MXAN_1153	LWT	ABC Transporter	[56]
MXAN_1154	LWT	ABC Transporter	[56]
MXAN_1155	LWT	ABC Transporter	[56]
MXAN_1167	Late aggregation	NtrC-Like Activators	[149]
MXAN_1189	Early aggregation	NtrC-Like Activators	This study
MXAN_1210	Late aggregation	ECF Sigma Factors	This study
MXAN_1245	Immature aggregates	NtrC-Like Activators	[149]
MXAN_1262	LWT	ABC Transporter	[56]
MXAN_1286	Aggregate-reaggregate	ABC Transporter	[56]
MXAN_1319	LWT	ABC Transporter	[56]
MXAN_1320	Early aggregation	ABC Transporter	[56]
MXAN_1321	Early aggregation	ABC Transporter	[56]
MXAN_1345	LWT	NtrC-Like Activators	This study
MXAN_1376	Late aggregation	ABC Transporter	[56]
MXAN_1377	LWT	ABC Transporter	[56]
MXAN_1402	Variable	One Component	This study
MXAN_1510	LWT	ECF Sigma Factors	This study
MXAN_1514	Immature aggregates	ECF Sigma Factors	This study
MXAN_1547	LWT	ABC Transporter	[56]
MXAN_1548	LWT	ABC Transporter	[56]
MXAN_1565	Variable	NtrC-Like Activators	[149]
MXAN_1575	Variable	One Component	This study
MXAN_1597	LWT	ABC Transporter	[56]
MXAN_1598	Variable	ABC Transporter	[56]
MXAN_1604	Variable	ABC Transporter	[56]

MXAN_1605	Early aggregation	ABC Transporter	[56]
MXAN_1654	LWT	One Component	This study
MXAN_1661	Early aggregation	ECF Sigma Factors	This study
MXAN_1667	LWT	One Component	This study
MXAN_1677	Variable	One Component	This study
MXAN_1683	LWT	One Component	This study
MXAN_1695	Immature aggregates	ABC Transporter	[56]
MXAN_1711	No aggregation	One Component	This study
MXAN_1719	LWT	One Component	This study
MXAN_1726	Variable	One Component	This study
MXAN_1746	Variable	One Component	This study
MXAN_1757	Variable	One Component	This study
MXAN_2018	LWT	ABC Transporter	[56]
MXAN_2019	No aggregation	ABC Transporter	[56]
MXAN_2020	Immature aggregates	ABC Transporter	[56]
MXAN_2030	Early aggregation	ECF Sigma Factors	[149]
MXAN_2078	LWT	ABC Transporter	[56]
MXAN_2128	Immature aggregates	One Component	This study
MXAN_2145	Late aggregation	One Component	This study
MXAN_2159	Early aggregation	NtrC-Like Activators	This study
MXAN_2184	Aggregate-reaggregate	ECF Sigma Factors	This study
MXAN_2204	Immature aggregates	ECF Sigma Factors	This study
MXAN_2230	Late aggregation	One Component	This study
MXAN_2234	Immature aggregates	One Component	This study
MXAN_2249	Late aggregation	ABC Transporter	[56]
MXAN_2250	LWT	ABC Transporter	[56]
MXAN_2251	Early aggregation	ABC Transporter	[56]
MXAN_2268	LWT	ABC Transporter	[56]
MXAN_2395	Early aggregation	ECF Sigma Factors	This study
MXAN_2407	LWT	ABC Transporter	[56]
MXAN_2428	LWT	ABC Transporter	[56]
MXAN_2429	LWT	ABC Transporter	[56]
MXAN_2430	LWT	ABC Transporter	[56]
MXAN_2437	LWT	ECF Sigma Factors	This study
MXAN_2500	Early aggregation	ECF Sigma Factors	This study
MXAN_2501	Aggregate-reaggregate	NtrC-Like Activators	This study
MXAN_2516	Immature aggregates	NtrC-Like Activators	This study
MXAN_2654	Early aggregation	ABC Transporter	[56]
MXAN_2711	Variable	One Component	[149]
MXAN_2783	Early aggregation	ABC Transporter	[56]
MXAN_2794	Immature aggregates	One Component	This study

MXAN_2795	LWT	ABC Transporter	[56]
MXAN_2831	LWT	ABC Transporter	[56]
MXAN_2832	LWT	ABC Transporter	[56]
MXAN_2833	LWT	ABC Transporter	[56]
MXAN_2853	Early aggregation	ABC Transporter	[56]
MXAN_2896	LWT	One Component	This study
MXAN_2929	LWT	ECF Sigma Factors	This study
MXAN_2949	Aggregate-reaggregate	ABC Transporter	[56]
MXAN_2951	Early aggregation	ABC Transporter	[56]
MXAN_3095	Early aggregation	NtrC-Like Activators	This study
MXAN_3142	Immature aggregates	One Component	This study
MXAN_3151	Immature aggregates	One Component	This study
MXAN_3208	Immature aggregates	ABC Transporter	[56]
MXAN_3209	Early aggregation	ABC Transporter	[56]
MXAN_3214	Fall apart	NtrC-Like Activators	[149]
MXAN_3240	LWT	One Component	This study
MXAN_3256	No aggregation	ABC Transporter	[56]
MXAN_3257	No aggregation	ABC Transporter	[56]
MXAN_3258	No aggregation	ABC Transporter	[56]
MXAN_3333	Variable	NtrC-Like Activators	This study
MXAN_3339	LWT	ABC Transporter	[56]
MXAN_3381	Aggregate-reaggregate	NtrC-Like Activators	This study
MXAN_3418	LWT	NtrC-Like Activators	This study
MXAN_3426	Early aggregation	ECF Sigma Factors	This study
MXAN_3429	Early aggregation	One Component	This study
MXAN_3443	Immature aggregates	One Component	This study
MXAN_3648	Variable	ABC Transporter	[56]
MXAN_3650	LWT	ABC Transporter	[56]
MXAN_3686	Early aggregation	ECF Sigma Factors	This study
MXAN_3692	No aggregation	NtrC-Like Activators	This study
MXAN_3702	No aggregation	One Component	[149]
MXAN_3711	Immature aggregates	One Component	This study
MXAN_3717	No aggregation	ABC Transporter	[56]
MXAN_3718	Fall apart	ABC Transporter	[56]
MXAN_3773	LWT	ABC Transporter	[56]
MXAN_3811	LWT	NtrC-Like Activators	This study
MXAN_3908	LWT	ABC Transporter	[56]
MXAN_3909	Early aggregation	ABC Transporter	[56]
MXAN_3959	Early aggregation	ECF Sigma Factors	This study
MXAN_3986	LWT	ABC Transporter	[56]
MXAN_4042	Immature aggregates	NtrC-Like Activators	This study

MXAN_4060	LWT	One Component	This study
MXAN_4072	Late aggregation	One Component	This study
MXAN_4110	Late aggregation	One Component	This study
MXAN_4173	LWT	ABC Transporter	[56]
MXAN_4196	No aggregation	NtrC-Like Activators	[149]
MXAN_4199	LWT	ABC Transporter	[56]
MXAN_4240	LWT	NtrC-Like Activators	This study
MXAN_4247	Late aggregation	One Component	This study
MXAN_4252	Late aggregation	NtrC-Like Activators	This study
MXAN_4261	LWT	NtrC-Like Activators	This study
MXAN_4263	LWT	One Component	This study
MXAN_4309	Early aggregation	ECF Sigma Factors	This study
MXAN_4316	Early aggregation	ECF Sigma Factors	This study
MXAN_4339	LWT	NtrC-Like Activators	This study
MXAN_4356	LWT	One Component	This study
MXAN_4471	LWT	One Component	This study
MXAN_4523	LWT	ABC Transporter	[56]
MXAN_4580	Early aggregation	NtrC-Like Activators	This study
MXAN_4622	LWT	ABC Transporter	[56]
MXAN_4662	Immature aggregates	ECF Sigma Factors	This study
MXAN_4665	LWT	ABC Transporter	[56]
MXAN_4716	Fall apart	ABC Transporter	[56]
MXAN_4733	Early aggregation	ECF Sigma Factors	This study
MXAN_4749	LWT	ABC Transporter	[56]
MXAN_4750	LWT	ABC Transporter	[56]
MXAN_4785	Late aggregation	NtrC-Like Activators	This study
MXAN_4790	Variable	ABC Transporter	[56]
MXAN_4899	Fall apart	NtrC-Like Activators	This study
MXAN_4949	Early aggregation	ECF Sigma Factors	This study
MXAN_4977	Early aggregation	NtrC-Like Activators	This study
MXAN_4983	Variable	NtrC-Like Activators	This study
MXAN_4987	LWT	ECF Sigma Factors	This study
MXAN_5029	No aggregation	One Component	This study
MXAN_5041	Immature aggregates	NtrC-Like Activators	This study
MXAN_5048	Late aggregation	NtrC-Like Activators	This study
MXAN_5101	Early aggregation	ECF Sigma Factors	[149]
MXAN_5124	Fall apart	NtrC-Like Activators	[149]
MXAN_5128	LWT	One Component	This study
MXAN_5153	Early aggregation	NtrC-Like Activators	[149]
MXAN_5245	Late aggregation	ECF Sigma Factors	This study
MXAN_5263	No aggregation	ECF Sigma Factors	[149]

MXAN_5271	No aggregation	One Component	This study
MXAN_5276	LWT	ABC Transporter	[56]
MXAN_5305	LWT	One Component	This study
MXAN_5356	Early aggregation	One Component	This study
MXAN_5379	LWT	ABC Transporter	[56]
MXAN_5410	Early aggregation	ECF Sigma Factors	[149]
MXAN_5480	Early aggregation	One Component	This study
MXAN_5492	LWT	One Component	This study
MXAN_5503	LWT	ABC Transporter	[56]
MXAN_5506	Early aggregation	ECF Sigma Factors	This study
MXAN_5545	Variable	One Component	This study
MXAN_5547	LWT	One Component	This study
MXAN_5584	LWT	ABC Transporter	[56]
MXAN_5680	Variable	NtrC-Like Activators	[149]
MXAN_5731	Immature aggregates	ECF Sigma Factors	This study
MXAN_5777	Variable	NtrC-Like Activators	[149]
MXAN_5853	Variable	NtrC-Like Activators	This study
MXAN_5879	Fall apart	NtrC-Like Activators	[149]
MXAN_5894	Variable	One Component	[149]
MXAN_6000	Late aggregation	ABC Transporter	[56]
MXAN_6058	Variable	ECF Sigma Factors	This study
MXAN_6149	LWT	One Component	This study
MXAN_6157	LWT	One Component	This study
MXAN_6161	LWT	One Component	This study
MXAN_6167	Variable	One Component	This study
MXAN_6173	Early aggregation	ECF Sigma Factors	[149]
MXAN_6206	Variable	One Component	This study
MXAN_6251	Variable	One Component	This study
MXAN_6402	LWT	ABC Transporter	[56]
MXAN_6426	No aggregation	NtrC-Like Activators	[149]
MXAN_6461	Fall apart	ECF Sigma Factors	This study
MXAN_6468	Variable	One Component	This study
MXAN_6475	Early aggregation	ABC Transporter	[56]
MXAN_6479	LWT	One Component	This study
MXAN_6486	LWT	One Component	This study
MXAN_6518	Variable	ABC Transporter	[56]
MXAN_6549	Late aggregation	One Component	This study
MXAN_6551	LWT	ABC Transporter	[56]
MXAN_6646	LWT	One Component	This study
MXAN_6653	No aggregation	One Component	This study
MXAN_6759	Immature aggregates	ECF Sigma Factors	This study

MXAN_6833	Variable	One Component	This study
MXAN_6889	No aggregation	One Component	[149]
MXAN_6967	Late aggregation	One Component	This study
MXAN_7072	Variable	One Component	This study
MXAN_7078	LWT	One Component	This study
MXAN_7214	Early aggregation	ECF Sigma Factors	This study
MXAN_7289	Immature aggregates	ECF Sigma Factors	This study
MXAN_7312	LWT	One Component	This study
MXAN_7316	LWT	One Component	This study
MXAN_7322	Late aggregation	One Component	This study
MXAN_7326	LWT	ECF Sigma Factors	This study
MXAN_7440	No aggregation	NtrC-Like Activators	[149]
MXAN_7454	Early aggregation	ECF Sigma Factors	This study

4. Phenotype probability distinguishes near-wild-type from wild-type behavior

This chapter contains work that has not yet been published, done in collaboration with Roy Welch, Alison Patteson, and Eduardo Caro. The manuscript was written by myself and Eduardo Caro. I was responsible for the analysis work, and Eduardo Caro performed the experiments. The image acquisition system presented here employed microscopy hardware previously produced by the Welch lab. I created the software that coordinates and controls the image acquisition and managed the networking of the microscopes, and Eduardo Caro maintained and optimized the microscopes.

4.1 Introduction

Multicellular development is a necessary part of the life cycle of many organisms, requiring extensive coordination and response to stochastic events for a desired phenotype to emerge [150–152]. Networks of intracellular and extracellular signals, gene expression, and the physical principles that control them are highly interwoven and contain redundancies [153,154]. The genotype-phenotype problem is the broad task of understanding this web of phenotypic effect and genetic cause.

We use the bacteria *Myxococcus xanthus* as a model organism that exhibits a rudimentary but robust form of development in the form of fruiting body aggregation, wherein cells in a low nutrient environment spontaneously organize from a homogeneous, flat swarm into a discrete number of mound-shaped aggregates harboring thousands of cells. In nature, these aggregates mature into fruiting bodies, which contain metabolically quiescent myxospores that can reseed

a new, active colony once conditions improve [42]. This example of a collective behavior that achieves a specific end (fruiting bodies) that are tied to communal survival is a promising object of study for the genotype-phenotype problem. A complex regulatory system is necessary for successful fruiting body development, including genes that regulate internal cell behaviors, as well as those that control cell-to-cell signaling, such as the cell-membrane-associated C-signal protein [155]. Using site directed mutagenesis, specific genes in the genome can be disrupted and observed for changes in the process and results of fruiting body formation, i.e., phenotypic changes [156]. The observations can then be contrasted with the genetically undisturbed "wild-type" behavior to intuit the function of the disrupted gene. However, gene expression itself is highly stochastic, particularly in timing. The protein a gene codes for can be produced in short, irregular bursts instead of at a steady rate [150,157]. This stochasticity is a trait that organisms like bacteria rely on for necessary biological processes such as sporulation [158].

Differentiating mutant behavior from wild-type is at the heart of the genotype phenotype problem and biological research in gene annotation, the assignment of function to otherwise unknown genes. Of the over seven thousand protein-coding genes identified in the genome of wild-type *M. xanthus* strain DK1622, nearly 40% do not currently have a predicted function [159]. A key factor needed to establish significance when observing mutant behavior is a boundary defining what may be considered "near-wild-type" behavior. Biological systems have inherent variation, so controlling genome and experimental conditions can only reduce that variation to a baseline level. If genes can only be identified by catastrophic changes in fruiting body formation, the knowledge of the associated gene function is severely limited. The orchestration of many cells to form fruiting bodies requires interplay between many genetic

factors, most of which, on their own, have only a subtle effect or an effect with high variability due to gene redundancy arising from gene duplication and other factors [113]. This causes problems with some conventional analysis methods such as t-tests on one developmental metric at a time. A random variable with high, unknown variance can be falsely selected by a t-test as significant, and when the number of samples is small, low sensitivity means no effect is captured with statistical significance [160]. This necessitates a method both for reliably observing subtle changes in phenotype and quantifying how far the deviation is from wild-type behavior.

Performing studies that embrace and scrutinize stochasticity is necessary for developing a working understanding of complex, biophysical systems. Statistical techniques that are useful in the face of highly stochastic events such as gene expression are a topic of active interdisciplinary research [161], and the inherent ability of living systems to submit to statistical study is an open epistemological question [162]. When a baseline level of variation can be measured for a wild-type system, any differences distinct from that baseline are capable of distinguishing mutant phenotypes. Many observations are needed to establish this baseline with confidence. In this study, we present a high-throughput experimental setup for observing many instances of cellular aggregation and an analysis pipeline that quantifies developmental phenotype in order to distinguish mutant strain behavior from wild-type behavior with statistical significance and assess the developmental impact of single gene mutations. The array of centrally controlled microscopes built to fulfill the unique needs of a high-throughput system for fruiting body aggregation assays, when combined with a novel statistical technique that

combines the effects of multiple quantitative markers of development, demonstrates a statistically significant measurement of the subtle effects of single gene mutations.

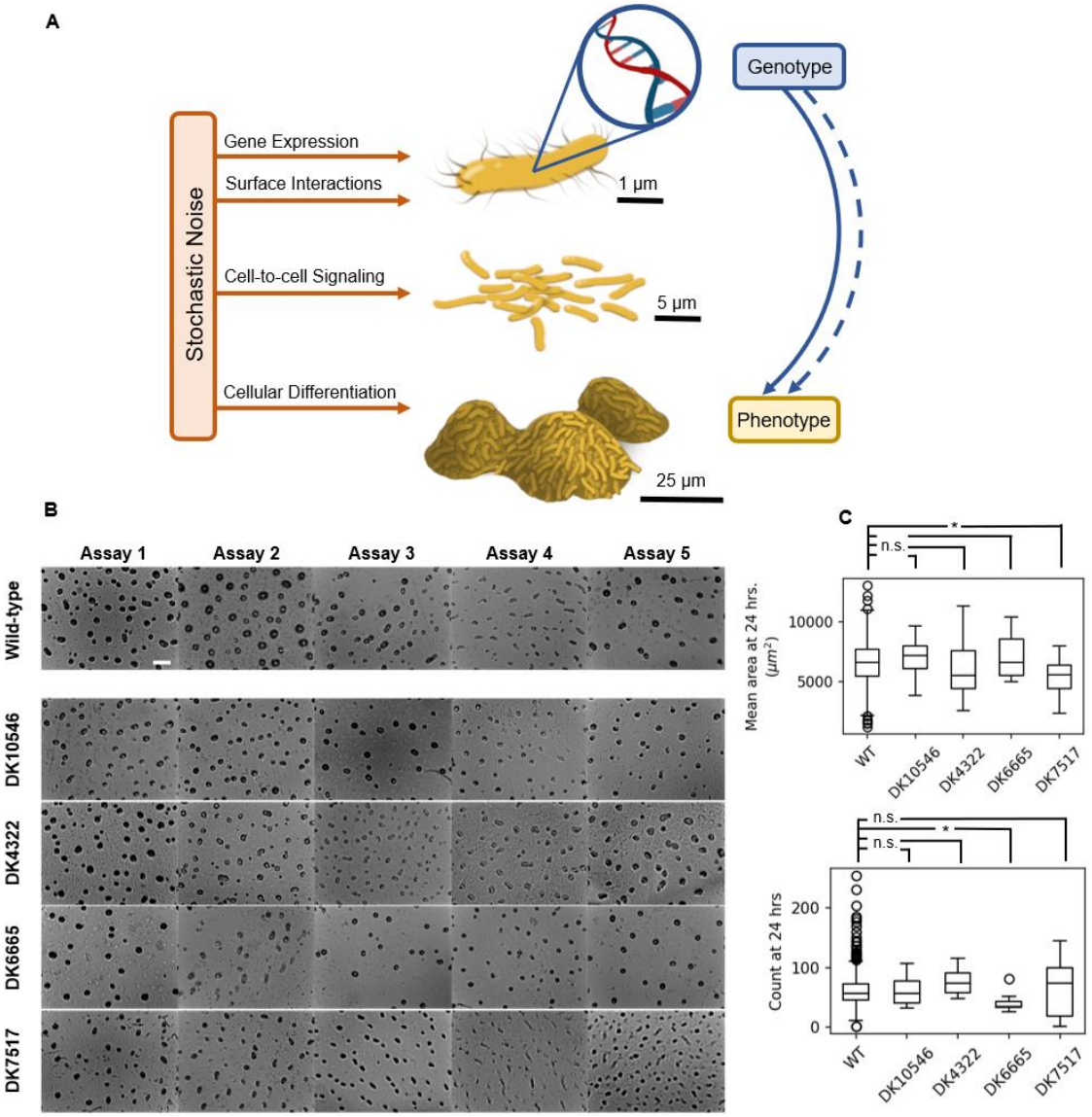


Figure 4.1: Stochasticity is inherent to multicellular behaviors in social bacteria. (**A)** A bacterial colony undergoing fruiting body development is exposed to stochastic noise on multiple scales. At the cellular level, gene expression depends on thermally driven chemical events, and environmental factors such as variations in temperature and humidity introduce further uncertainty. Thus both direct and indirect effects of genotype arrive at a final phenotype, possibly via multiple developmental paths. (**B)** Images of final developmental phenotype for separate aggregation assays at 24 hours post inoculation. Pictured are a range of outcomes from the wild-type *M. xanthus* strain as well as the four mutant strains used in this study. Aggregates are visible as dark spots, seen from above. Scale bar, 250 µm. **(C)** The average final area and final count of wild-type aggregates and those for four mutant strains are reported with boxplots. Although there are some differences in these typical metrics of comparison,

there is considerable overlap between wild-type and each of the mutant strains. N > 500 measurements for wild-type, taken over 25 different days; N = 15 measurements for each mutant strain, taken over 2 different days.

4.2 Results

An accurate measurement of the variation of wild-type behavior requires a large sample size to accommodate the many developmental outcomes observed in *M. xanthus* fruiting body development and estimate their probability. To accomplish this with enough simultaneously running assays to measure the impact of day-to-day variation, we produced custom built microscopes and a microscope control network to collect and organize developmental data. Each fruiting body aggregation assay requires a sealed chamber with sufficient temperature, oxygenation, and humidity for development to occur. Cells are inoculated from liquid culture and sealed in each slide assembly, where the aggregates begin to form after about five hours. After manual setup and focus, automated imaging proceeds for 24 hours, at which point most aggregates are stable. The resulting time series images are organized on a central hub computer from which image processing can begin.

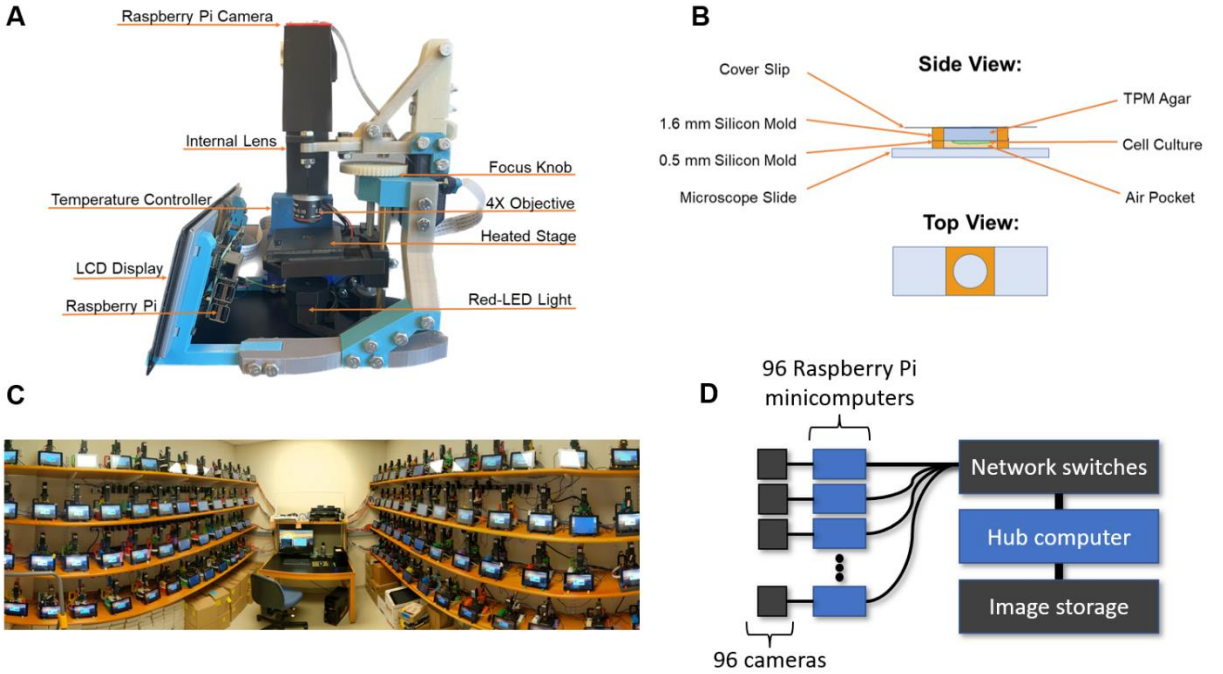


Figure 4.2. High-throughput time series acquisition setup. (A) A single microscope, at 1.2 kg and 24x19x25 cm^3, with 3D-printed armature, 4X objective lens, light source, heated stage, camera, and Raspberry Pi microcomputer. **(B)** Slide assembly for each developmental experiment. Sandwiched between a glass coverslip and a glass slide, two silicone gaskets create a sealed enclosure containing a disk of non-nutritive agarose on which a colony of *M. xanthus* has been inoculated. Aggregate development is imaged over a 24-hour period with one image taken each minute. **(C)** Panoramic photograph of full image acquisition setup including 96 microscopes and central hub computer. **(D)** Basic network architecture for centralized image storage and control of all 96 microscopes.

Using a dataset of over 500 wild-type aggregation time series acquired over 25 separate days,

we quantify the range of developmental phenotype by measuring ten quantitative metrics for

each video. We choose three metrics related to timing: start time, when aggregation begins; peak time, when the area occupied by aggregates reaches its maximum; and stability time, when the number of aggregates becomes stable. We also measure the mean and standard deviation in average aggregate area at two time points: peak time, and 24 hours. We measure the number of identifiable aggregates at both peak time and 24 hours. Finally, we measure the fraction of aggregates that appear and then disperse before 24 hours elapse from inoculation. Myxobacteria development, when observing fruiting body morphogenesis, is typically resolved by 24 hours, especially in wild-type DK1622. After placement onto starvation media, the movement of aggregates, their size, and distinctive morphology has usually been set by this point, even though the internal myxospores are still maturing for an additional 3-5 days. For the purposes of our study, we chose to limit our time lapses to 24 hours to highlight differences in the dynamics of fruiting body aggregation. Phenotypic differences could yet be observed in mutants that take longer than 24 hours using the methods established here. The specific algorithms used to determine each phenotypic metric are detailed in the Supplementary Materials (Supp. Table 4.3). These metrics are extracted with a custom Python image processing algorithm that identifies and measures each aggregate, as described in Methods. Values for these metrics across the wild-type dataset are shown in Figure 4.3A, with the distributions illustrating averages and variation for each metric. Peak time, aggregate count at peak time, and the fraction of aggregates that disperse exhibit bimodal distributions. Long-tailed distributions, such as the standard deviation (σ) of area and aggregate count (both at peak time and after 24 hours) indicate the presence of abnormal phenotypes with extreme values in these metrics.

We next map the wild-type dataset in a visualizable way. Because we use ten phenotypic metrics, each time series may be represented by a point in a 10-dimensional phenotype space, where points closer together are more phenotypically similar than points far apart. To reduce the number of dimensions but retain the structure of our dataset, we use principal component analysis (PCA), to reduce the number of dimensions from ten to two. the resulting metrics from each time series are mapped to a point in a 2D phenotype space. The two dimensions of this space are called PC1 and PC2, the first and second principal components, respectively. PC1 and PC2 are each a single numerical measure that is a mathematical composite of multiple quantitative features, each weighted differently. Principal component analysis guarantees that PC1 and PC2 are the metrics that display the most variation across the wild-type dataset as compared to any other linearly independent combination of the input metrics. Between just PC1 and PC2, the majority of the variance across the full dataset (56%) is accounted for. The distribution of points in this map constitutes the wild-type phenotype profile.

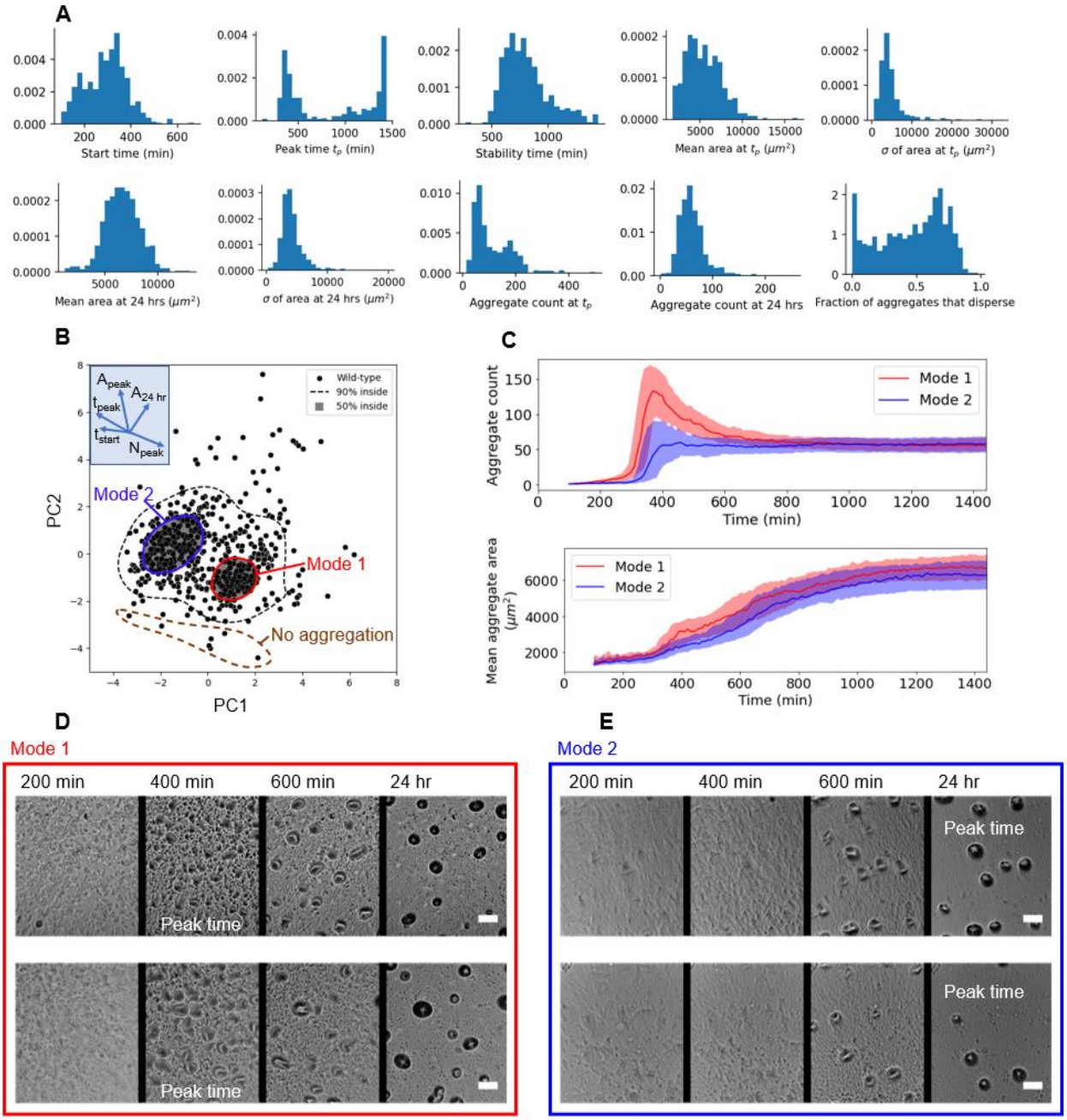


Figure 4.3. Quantitative breadth of wild-type phenotype. (A) Histograms display the range of phenotypic metrics across over 500 wild-type aggregate development time series. Bimodal shapes in peak time (when total aggregate area is maximum), aggregate count at peak time, and fraction of aggregates that disperse reflect the two most common groupings of metrics. Long-tailed distributions, such as standard dev. (σ) of area and aggregate count (both at peak time and after 24 hours) indicate the presence of abnormal phenotypes. All y-axes display probability density. **(B)** By using PCA to combine information from all ten metrics, each wild-type time series is plotted as a single datapoint in a phenotypic feature space. PC1 primarily measures aggregate area, and PC2 correlates with number and timing of aggregates. For example, while moving in the direction of the arrow labelled "N_{peak}," datapoints will have generally higher numbers of aggregates at peak time. Units of PC1 and PC2 are arbitrary, although the origin at (0,0) represents average behavior across the full wild-type dataset. A contour is drawn enclosing 90% of the datapoints, separating typical phenotypes from rare phenotypes. Within typical behavior, two

separate clusters, Mode 1 and Mode 2, contain 50% of the wild-type datapoints. **(C)** Curves displaying the total number of aggregates over time (top) and mean area of aggregates over time (bottom) illustrate the developmental differences and similarities between the two wild-type modes. The central line represents the median at each time point, and the colored bands span the 25th to 75th percentiles at each time point, i.e. half the data about the median. In Mode 1, a larger number of aggregates develop at an earlier time, most of which disperse. The final number of aggregates is comparable for both modes. The rates of increase of mean area are also similar across the two modes. **(D&E)** Two representative time series each for Mode 1 and Mode 2 phenotypes at three relevant time points. Mode 1 displays many, dense aggregates that form early and then disperse. This causes an early peak time. Mode 2 displays aggregates that form later, most of which persist through the 24 hours of development, slowly growing in area and darkening. This causes a late peak time. Scale bar 100 μm.

The unique weighted combination of metrics that make up PC1 and PC2 indicate key metrics that can distinguish behavior. The weights are bounded between -1 and 1, with larger absolute values indicating more strongly weighted metrics. Both PC1 and PC2 contain a mix of all ten metrics, with no one metric standing out in significance over the others, but rather groups of metrics being more significant. In this study, the top weighted metrics of PC1 (with weights given in parentheses) are number of aggregates at peak time (0.47), fraction of aggregates that disperse (0.44), peak time (-0.43), and start time (-0.39). For PC2, the top metrics are mean area at peak time (0.57), standard deviation in area at peak time (0.46), standard deviation in area at 24 hours (0.42), and mean area at 24 hours (0.40). The full list of weights is given in the Supplementary Materials (Supp. Table 4.2). In summary, PC1 is primarily shared between timing and the total number of fruiting bodies that form, in diametrical opposition. That is, when aggregation starts and peaks at an earlier time, the number of fruiting bodies tends to be larger, and vice-versa. PC2 is a variable independent from PC1 that mostly characterizes area. Thus, large aggregates can present in large or small numbers, and do so early or late relative to average wild-type behavior.

We observe two primary modes of aggregate formation, as shown by the two shaded regions in Figure 4.3B. What we term “Mode 1” features aggregates that start forming and peak in total

aggregate area generally sooner than other wild-type assays. Mode 1 aggregates are generally numerous, small, and dark at peak time, but a large fraction of them disappear before 24 hours of development. These aggregates tend to be dynamic and lack a well-defined shape until after peak time (Figure 4.3D). In contrast, Mode 2 aggregation is less mature early on, with either no visible aggregates or aggregates with fewer layers of cells able to block light (Figure 4.3E). These aggregates are more static and form with more well-defined shapes, and more of them tend to persist through the 24 hours of development. Because these aggregates tend to persist once they form, the time of peak total area is very late for Mode 2, when stable aggregates are still growing slowly. Although there are fewer Mode 2 aggregates at peak time than most wild-type assays, the mean number and size of these aggregates at 24 hours is equal to that of Mode 1, as well as wild-type assays in general. Both modes demonstrate more consistently sized aggregates than other wild-type assays, both at peak time and at 24 hours. Histograms of all ten metrics for the two modes are presented in the Supplementary Materials (Supp. Fig. 4.6). Exceptional phenotypes observed in our wild-type dataset include those that produce unusually large fruiting bodies. These occur by a variety of mechanisms, such as large aggregates forming either extremely early with defined shapes from initial formation or extremely late with shapes that only appear visible towards the end of 24 hours (time series in Supplementary Materials, Supp. Fig. 4.7). These abnormal behaviors present at the margins of PCA phenotype space because they represent a confluence of multiple abnormal metrics, revealing more information than standard statistical tests on one metric at a time.

Some rare behaviors observed include failure to aggregate, which occurred in about 2% of wild-type assays, and failure for aggregates to stabilize after 24 hours, which occurred in about 17% of wild-type assays.

We choose four mutant strains to compare with our nominal wild-type strain DK1622, each with 60 to 80 replicates each collected over two to six separate days. These strains were chosen to be developmentally similar to wild-type in order to test the sensitivity of our methods. In preliminary experiments, all four strains produced a set of three replicates that were manually identified as “near-wild-type” and displayed final aggregate size and number that could not be distinguished from wild-type with a Student’s t-test. Three mutant strains contain insertions of simple reporter genes, such as DK10546 producing GFP (See Methods for more details on each strain). Analyzing these strains tests the assumption that introducing reporter genes into a prokaryotic genome will not significantly impact cellular behavior or emergent phenotypes, an important preliminary consideration before their use in other experiments. When more replicates had been analyzed, standard statistical tests distinguish one strain, DK7517, as distinct from wild-type because it produces smaller than average aggregates (Fig. 4.1). Because the distribution of wild-type final mean areas is non-Gaussian as measured by a Shapiro-Wilk normality test, the Kolmogorov-Smirnov test for distinguishing two distributions was chosen as the standard test in favor of a Student’s t-test, which assumes normality of the underlying distributions.

The developmental data for the additional replicates of the mutant strains are projected onto the same PC1, PC2 axes that were defined for the wild-type data. This allows direct comparison and visualization of multiple metrics simultaneously. The typical behavior and variability of each

mutant strain's development is captured by two regions: a contour is drawn that captures 50% of the datapoints, creating an effective median region in PCA space. We then choose a wider contour that captures 90% of datapoints to serve as a boundary for abnormal phenotypes. By comparing the distribution of the mutant strain points to that of wild-type, a p-value can be calculated for the null hypothesis that the mutant datapoints are drawn from the wild-type distribution. This p-value depends on the number of points found inside the 50% contour and the number inside the 90% contour, and was calculated with bootstrapping, as described in Methods. This is a nonparametric, data-driven statistical method that makes no assumptions about the dataset *a priori*, allowing for multi-modal distributions which are likely to arise in living systems. Validation can be confirmed on multiple subsamples to quantify the impact of day-to-day variation on the p-value.

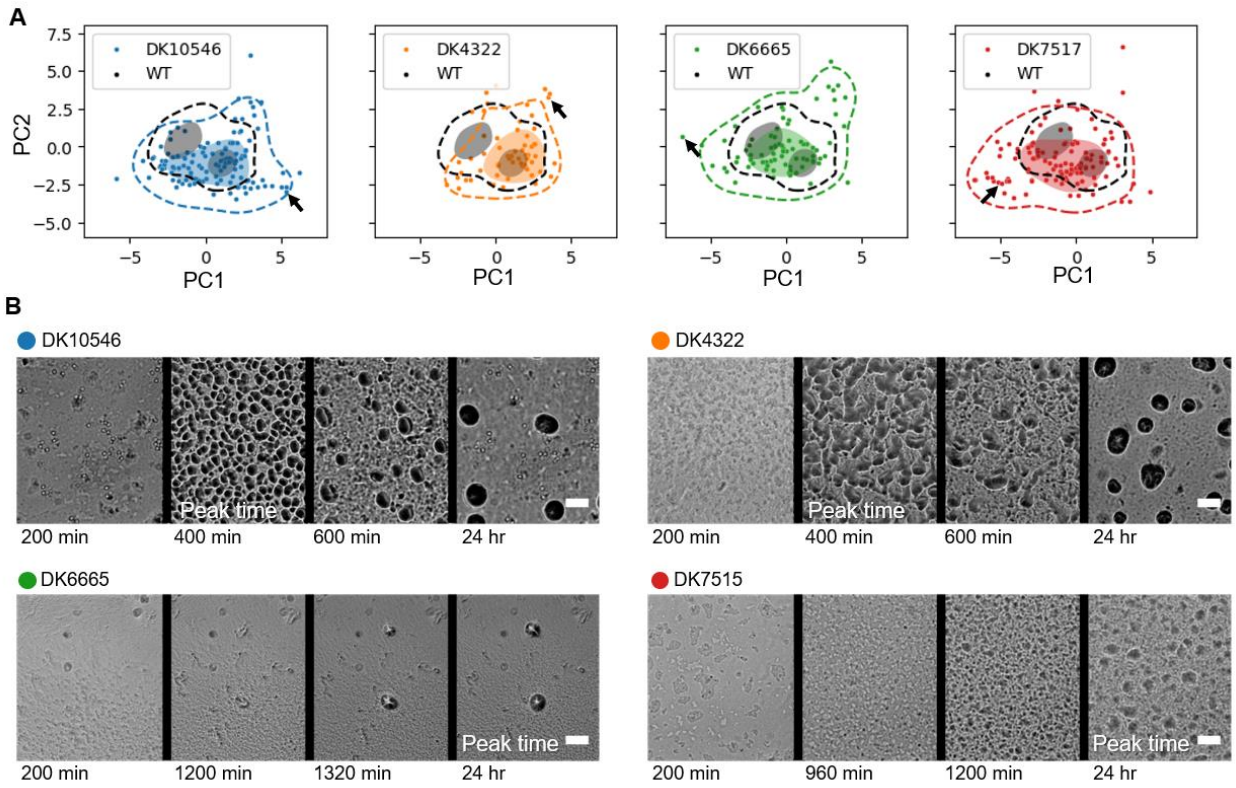


Figure 4.4. Deviation of near-wild-type mutant strains from wild-type behavior. (A) Each mutant development time series is plotted as a single data point in phenotype space, as measured by the collective metrics PC1 and PC2. For each respective strain, dashed contours enclose 90% of data points, and the shaded region(s) enclose 50% of data points. The 90% and 50% contours for wild-type are shown for reference. The deviation of mutant phenotype from wild-type is determined by the departure of the mutant distribution from the wild-type distribution. Statistically significant departures from the wild-type distribution are measured for all four mutant strains, with p-values calculated for subsamples of only 15 replicates each. These p-values are calculated from many random samplings drawn from the wild-type dataset (Methods). Arrows point to time series shown in **(B)** Time series of phenotypes expressed rarely in wild-type are shown at three relevant time points for each mutant strain. Scale bar 100 µm.

All four strains demonstrated subtle yet statistically significant departure from wild-type behavior. Strains DK10546 and DK4322 in particular showed a preference for Mode 1 behavior, with Mode 2 being rarely expressed, unlike in wild-type. Some mutant replicates that exhibited rare behaviors are highlighted in Fig. 4.4B. Replicates of DK10546 displayed more extreme versions of Mode 1 behavior, in which many small aggregates form early on, almost all of which disperse by 24 hours. DK4322 replicates also displayed a more extreme version of Mode 1 behavior in which aggregates at peak time, although distinct, had very irregular shapes. The

final aggregates were slightly larger and more varied in area than typical wild-type assays. The aggregates of some DK6665 replicates formed from sparse, small points that formed late and grew steadily over the course of the 24 hours. This nucleation was seldom expressed in wild-type. This strain also displayed difficulty in the dispersal of the random initial cell clumps that are present at inoculation. In wild-type, these initial clumps nearly always disperse, and final aggregate positions have no correlation with these initial clumps. Finally, the abnormal behavior of DK7517 replicates, which involved late aggregates that never significantly darkened, was also a noticeable deviation from even exceptional wild-type behavior. Among the mutant strains, about 2% failed to aggregate (the same fraction as wild-type), and 30% to 45% of mutant assays failed to stabilize after 24 hours, a significant increase from the 17% observed in wild-type.

4.3 Discussion

Each instance of fruiting body formation is the result of the combined effects of the genetic background, environmental factors both controllable – such as temperature or substrate stiffness – and uncontrollable – such as local pockets of varying initial cell density, or the changes in gene expression that uniquely unfold for that specific population of cells. Genetic changes may be better described by how they affect the odds of a multiplicity of outcomes. In fact, we observe that most mutants behave like wild-type a majority of the time. This is a significant reinterpretation of the meaning of the “phenotype” that results from a given genotype, not as a guaranteed outcome, but as a reshuffling of likely outcomes. This description is appropriate to the physics of living systems, which are tuned through evolution to be poised at the center of a variety of behaviors, ready to adapt to rapid changes either in the

organism or its environment. Techniques in the biostatistics community are consistent with this perspective, such as probabilistic latent variable models [163], which complement the analysis presented in this study.

Essentially, the statistical method reported here characterizes phenotype in terms of abnormal behavior, either locally – i.e. groupings of behavior that fall within the broad scope of wild-type, but are still outside the norm – or globally – i.e. behaviors that are never expressed in the entire wild-type profile. Because measures of mean behavior can fail to capture variations, such as a shifting distribution that happens not to be skewed, the study of abnormalities is a fruitful ground for distinguishing the effects of single-gene mutations, especially when enough replicates can be performed to reliably observe abnormal behavior [164]. This perspective also emphasizes the need to vet exceptional data to ensure that they represent genuine but rare behavior and are not simply abnormal due to a failure of data processing, such as inconsistent imaging conditions. Insofar as possible, the metrics chosen in this study were selected to minimize dependence on imaging setup, and manual vetting was performed on points that fell on the margins of PCA phenotype space.

The methods we report in this work are not unique to bacterial development. A similar analysis could be performed for any stochastic system that can: 1) Have many comparable replicates prepared, 2) Have multiple relevant metrics measured for each replicate. This method can thus be compared to a similar general analysis framework, such as machine learning. Machine learning is powerful in that metrics do not need to be chosen in advance. However, this comes at the cost of the transparency in how categorization is accomplished, and the need for training a model on input data categorized by some other method. It is often true that a system of

interest has several obvious aspects that are amenable to measurement with image processing. In the case where the dynamics of the system are relevant, our method is also attractive to use with time series image data, which in raw form can be multiple gigabytes in size and generally difficult to work with on large scales. The initial choice of phenotypic metrics represents a large simplification of the unprocessed image data that ensures phenotypic relevance is preserved over image acquisition noise. The further reduction of the phenotypic dataset from ten to two dimensions not only produces a phenotypic map that is sufficiently navigable to reveal overall structure and guide investigation of individual datapoints, but also avoids a well-known problem in data science associated with the so-called “curse of dimensionality.” This issue occurs in high-dimensional datasets, where geometry tends to make the distance between neighboring points similar to the distance across the dataset, making “similarity” in terms of distance essentially meaningless [165].

Although 60 replicates or more were analyzed for each mutant strain in this study, a strain that appears like wild-type to the eye can be distinguished from wild-type with fewer replicates. By analyzing the distribution in PCA space of many subsamples of wild-type aggregation, we find that only 15 replicates spread over two days are needed to establish departure from wild-type behavior for each of the mutant strains above. Notably, for this same sample size, standard statistical tests based on individual metrics (average aggregate area after 24 hours shown in Supplementary Materials) can only distinguish one strain, DK7517, as distinct from wild-type, and with less statistical power than the method used in this study. Because the distribution of final mean areas is non-Gaussian as measured by a Shapiro-Wilk normality test, the

Kolmogorov-Smirnov test for distinguishing two distributions was chosen as the standard test in favor of a Student's t-test, which assumes normality of the underlying distributions.

Our results indicate that there are indirect effects on fruiting body formation dynamics in *M. xanthus* due to the use of common reporters. Reporter genes are used as effective markers for successful transfection and are used for quantitative assays, which require them to not obstruct or alter the mechanism of study. GFP, a 28kDa green-fluorescent-protein, allows for the precise visualization of proteins using UV light. Although small enough to diffuse from the cytosol into the nucleus, there are inherent indirect costs to attaching these tags to a molecule of interest. By inserting this extra DNA, another introduction of molecular noise via transcription and translation steps is added during these biochemical reactions [166–168]. These non-target effects can cause cellular differences in expression changes which contribute to the stochastic variation we see in the overall cell population [169].

Another reporter gene, Tn5 lac, is a promoter-less trp-lac fusion that was designed to identify strains that specifically increase beta-galactosidase expression at some point during *M. xanthus* development as developmental markers. Transposons are a diverse class of mobile genetic elements that can promote genetic rearrangements without a requirement for sequence homology [170]. The Tn5 transposon was inserted so lac Z transcription occurred with exogenous promoters and their promoter strength was quantified [171] to identify genes that were expressed during *M. xanthus* fruiting body morphogenesis. By attaching to the promoter, it was assumed that lacZ expression would occur in parallel with gene-specific myxospore development without disrupting gene function. However, Tn5 transposon insertions can promote adjacent deletions [172] which can then disrupt regulatory regions and lead to

changes in phenotype from differences in gene expression [173]. The ability to differentiate these transposon insertion strains from wild-type emphasizes the need to assess the impact of reporter genes, especially in biophysical studies that focus on developmental dynamics, where differences may be easier to observe.

These results point to a method of gene annotation that is sufficiently sensitive to identify the impact of single gene mutations that would otherwise be imperceptible. Each mutant strain becomes associated with a signature distribution in a PCA space that can be generally defined and used by any laboratory. Strains with sufficiently similar distributions can then be said to share a function because they impact development in a demonstrably similar way. If the signature distribution of well-understood genes is reported, newly characterized genes of unknown function can be compared to those benchmark distributions. Notably, these signatures are agnostic of any specific biological model, and are based only on visually observable characteristics. Future work can process a library of single-gene knockout strains with unknown function. Over time, this also develops an overall phenome with respect to fruiting body development that is quantitative, and which creates a common language of comparison for the function of many different genes. As more such experiments are done in this framework, either through high-throughput imaging methods like those described here or by the collective efforts of many researchers, developmental phenotypes that are uncommon will be revealed. These unusual events serve to define a boundary on multicellular behavior and can expose the regulatory mechanisms of fruiting body formation when stretched to their limits by stochastic factors alone. These “exceptions” can teach us much about the “rule.”

Among these sources of behavioral change, we expect that small aspects of experimental protocol will have subtle but measurable effects. Over the course of the experiments carried out for this work, a new protocol variable was confirmed that is not normally controlled for in *M. xanthus* culture, namely the age of the agar plate containing colonies to be harvested for liquid bacterial culture. It is expected that reintroducing bacteria to liquid culture will "reset" their metabolic state regardless of what state they were in before, but our analysis revealed preliminary evidence that a colony will "remember" the age of the plate it was harvested from and produce fewer fruiting bodies if the colony grew on the plate for at least three days (Supplementary Materials, Supp. Fig. 4.8). Although the mechanism of this memory is unknown, the effect has been consistent, and we expect that other such protocol variables exist that have a measurable phenotypic impact.

The sensitivity of the methods presented here may also be used to measure phenotypic response to changes in a variety of environmental variables. In a similar fashion to running replicates of single-gene mutants, running replicates with differing substrates will reveal subtle or overt changes in phenotype. This future work could address the missing environmental information of the genotype-phenotype problem and expand the bounds of "wild-type behavior" as a function of environmental conditions. The contour bounding abnormal wild-type behavior encompasses exactly what has not yet been characterized to have a specific cause, providing both a measure of ignorance of relevant physical and biological mechanisms, and a way to characterize how much knowledge is gained when subregions can be assigned a root cause, and thus separated from wild-type.

4.4 Conclusions

Overall, our work demonstrates that single gene disruptions can produce measurable changes in *M. xanthus* aggregation development, even when introducing reporter genes widely assumed to be benign with respect to impacting cellular behavior. These effects are subtle and emphasize changes in dynamics over changes only in final developmental outcome, but they can be reliably observed in samples of only 15 replicates. These methods provide a pattern for characterizing development by mapping out phenotype in a clearly visualizable way that also incorporates the many different quantitative aspects that can be measured in collective, living systems. These tools can serve as a language of data presentation with applications in gene annotation or investigations of the impact of environmental variables on the genotype-phenotype problem.

4.5 Methods

Imaging setup

The setup can simultaneously collect time series images for 96 experiments using an array of compact and identical microscopes controlled by a central computer. Each of the 96 microscopes is equipped with a single 4X objective lens, a Peltier device that maintains stage and sample temperature, a red-light source, and a camera controlled by a Raspberry Pi, a single-board minicomputer. The 3D-printed armature and assembly hardware serve to keep all components firmly in place and provides a focus knob for higher image quality.

To ensure uniform control of all microscopes and central storage of their time series output, each individual Raspberry Pi unit is networked via ethernet and two 64-port network switches

to a central hub computer. This computer runs Piserver software, which boots each Raspberry Pi from a single operating system image, allowing software to be changed and updated for all Raspberry Pi units simultaneously. Custom software written in Python provides a convenient GUI to control image acquisition from each camera via SSH and organize output in a centralized image storage location.

Cell culture

Long term stock cultures were recovered on nutrient rich CTTYE media agar (1% Casein Peptone (Remel, San Diego, CA, USA), 0.5% Bacto Yeast Extract (BD Biosciences, Franklin Lakes, NJ, USA), 10 mM Tris (pH 8.0), 1 mM $KH(H_2)PO_4$ (pH 7.6), 8 mM $MgSO_4$). Cells were harvested from the plates and used to inoculate broth cultures in CTTYE with vigorous shaking at 32°C and grown to an approximate density of $4x10^8$ cells/mL (100 Klett or 0.7 A_{550}).

Cells were centrifuged to remove the nutrient broth, washed in TPM buffer (10 mM Tris (pH 7.6), 1 mM $KH(H_2)PO_4$, 8 mM $MgSO_4$), and resuspended to a final concentration of $4x10^9$ cells/mL. For the development assay, approximately $4x10^7$ cells (10µL aliquots) were spotted onto a TPM agar slide, a nutrient limited medium, then incubated on the microscope stage at 32°C for 24 hours. TPM slides were prepared as previously described [146].

Table 4.1. The strains used in this study.

Strain	Description
DK1622	The nominal **wild-type** strain was genetically modified from a naturally occurring *M. xanthus* isolate [174]. This was done to establish a stable baseline for fruiting body development assays, as strains isolated directly from soil have a high rate of developmental failure in a laboratory setting.
DK10546	The free expressing GFP labeled strain is used to track motility and cell dynamics during development. Used as an experimental control for fluorescence microscopy, the construct was generated fusing a copy of the *pilA*

	promoter to the coding sequence of GFP (*pilAp*-GFP), that was then re-inserted into the *M. xanthus* chromosome [175]. This study showed an increased likelihood of early aggregation with many dispersing aggregates over wild-type for this strain.
DK6665	The Tn5 Ω6658 sasB7 mutant was generated from a mutation created in suppressor developmental gene *sasB* [176]. Previous work observed no visible phenotypic impact on the mutant as the strain can still proceed through development via other regulatory channels. This study observed initial cell clumps having an unusually high impact on final aggregates due to a lack of dispersal of initial cell clumps relative to wild-type.
DK4322	*The spiA*::Tn5-*lacZ strain,* is a reporter fusion for the developmental gene *spi* with lacZ for β-galactosidase assays. The *spi* gene has been shown to be induced at 2 hrs into development and is developmentally regulated by C signal pathways. In previous work, the transposon insertion was characterized as not interfering with development or affect spore production [171]. This study showed a higher likelihood of irregular aggregate shapes at early times for this strain than wild-type.
DK7517	Generated via a Tn5-LacZ insertion into a TA synthesis gene, as a reporter gene involved in toxin and antitoxin production also for β-galactosidase assays [177] to isolate regulatory mutants. This reporter fusion was shown to be expressed during vegetative growth while peaking during lag phase. In this study, late and immature aggregates were more likely to develop in this strain than wild-type.

Image processing pipeline

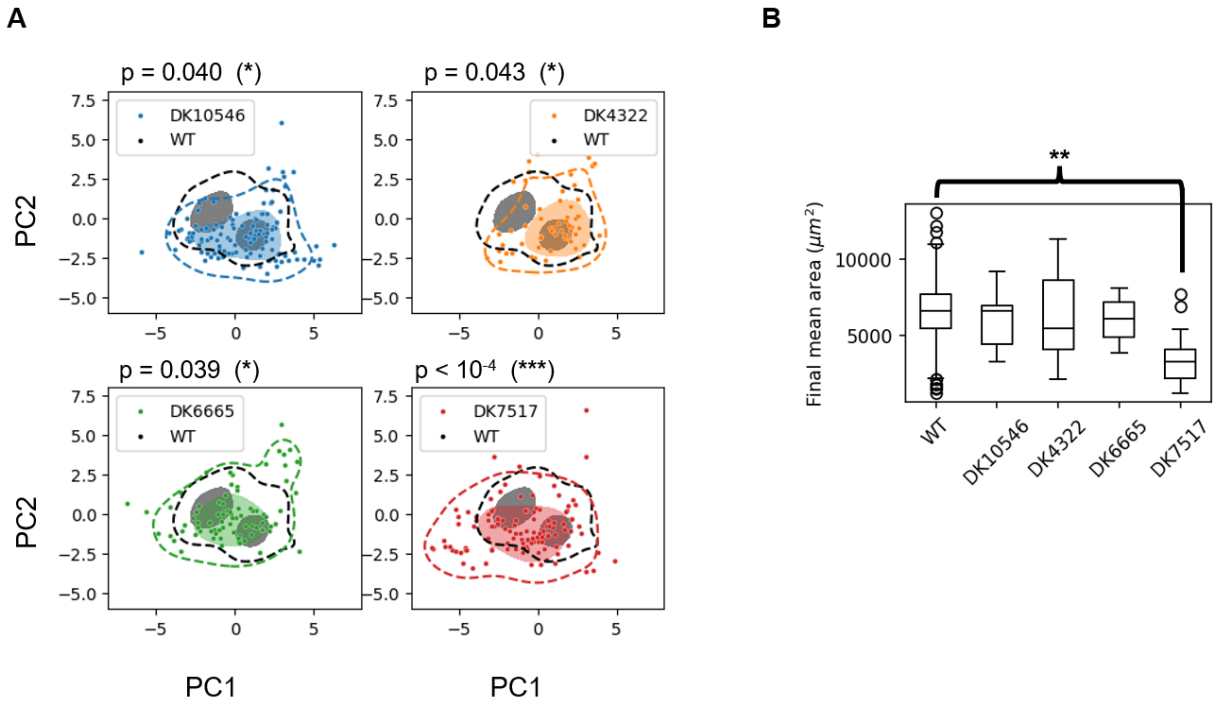
Phenotype was automatically quantified for each fruiting body aggregation assay in this study by running 144 individual .TIFF images (ten minutes between each frame over 24 hours of total development) from each time series through a custom Python image processing and analysis pipeline to identify in each frame which pixels could belong to a fruiting body, based on their gray value. The information for the position and geometry of each aggregate was filtered to remove noise and spurious aggregates. This detailed data summary for each time series then had a list of ten specific numbers extracted from it, each of which captures one overall feature, such as the time at which aggregation began or the average size of final fruiting bodies. The values of these ten metrics together were then used in further analysis. The full details of the

image processing pipeline and all phenotypic metrics are available in the Supplementary Materials (Supp. Table 4.3).

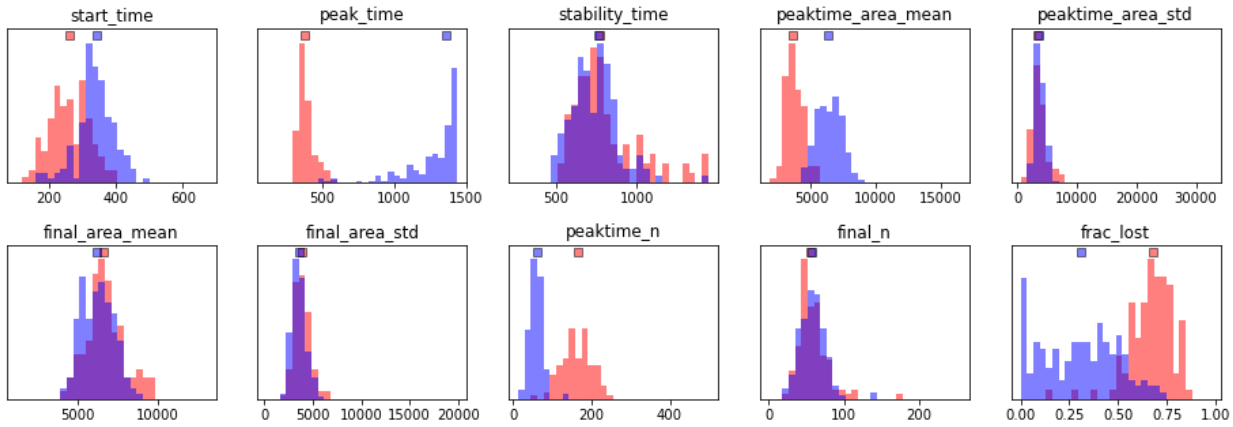
Statistical methods

To calculate p-values that test the null hypothesis of mutant development datapoints in PCA space being drawn from the same distribution as the wild-type development datapoints, we first generate the contours for the wild-type PCA data by starting with Gaussian kernel density estimation (KDE) and using standard root-finding techniques to draw contours from the density estimate that capture 50% and 90% of the PCA datapoints. An appropriate kernel size for the KDE is validated by using 75% of the wild-type dataset, and ensuring that, across many subsamples of the remaining 25% (verification data), the distribution of enclosed points is centered on the appropriate percentage. When this distribution is skewed, it indicates overfitting of the original contour. With these contours drawn, we then use a data-driven statistical technique similar to bootstrapping. Given a sample size N, 10,000 samples of that size are drawn from the wild-type dataset. Each subsample has a characteristic pair of numbers, (n_{50}, n_{90}), which corresponds to the number of points in the sample that fall inside the 50% and 90% contours, respectively. Once the distribution of these pairs for wild-type data is known, n_{50} and n_{90} are calculated for a sample of mutant PCA datapoints of size N. The fraction of wild-type videos that have both n_{50} and n_{90} greater than the mutant sample's values of n_{50} and n_{90} gives the p-value, or the probability that a sample of wild-type data of size N would exhibit the same distribution. Contours for mutant strains are shown for visualization only, and do not figure into the calculation of the p-values.

4.6 Supplementary Materials



Supplementary Figure 4.5. Standard statistical test (Kolmogorov-Smirnov on one metric) compared with p-value calculated from changing distributions in PCA space. (A) Each mutant development time series is plotted as a single data point in phenotype space, as measured by the collective metrics PC1 and PC2. For each respective strain, dashed contours enclose 90% of data points, and the shaded region(s) enclose 50% of data points. The 90% and 50% contours for wild-type is shown for reference. The deviation of mutant phenotype from wild-type is determined by the departure of the mutant distribution from the wild-type distribution. Statistically significant departures from the wild-type distribution are measured for all four mutant strains, with p-values calculated for subsamples of only 15 replicates each. These p-values are calculated from many random samplings drawn from the wild-type dataset. **(B)** The same mutant strains are compared against wild-type using only a metric taken from the last frame of development, final aggregate area. With subsamples of size N=15, a Kolmogorov-Smirnov test can only distinguish between DK7517 and wild-type with statistical significance.



Supplementary Figure 4.6. Comparison of metrics for Mode 1 (red) and Mode 2 (blue) wild-type assays. Modes 1 and 2 are defined by the two regions of the contour bounding 50% of the total wild-type assay data in PCA space, as shown in Fig. 4.3 of the main text. The medians of each histogram are indicated by the position of the square marker at the top of each subplot.

A fruiting body aggregation assay performed in any lab can be compared to the results reported in this work by calculating the values of PC1 and PC2 for that assay. First, scale each metric by subtracting the mean WT value of that metric and dividing the result by the WT standard deviation of that metric, according to the values reported in Table 4.1. This normalizes each metric to have zero mean and unit variance. Then, each resulting scaled metric is multiplied by an appropriate weight, either for PC1 or PC2. The weighted sum of the scaled metrics gives the final value of PC1 or PC2 for that assay.

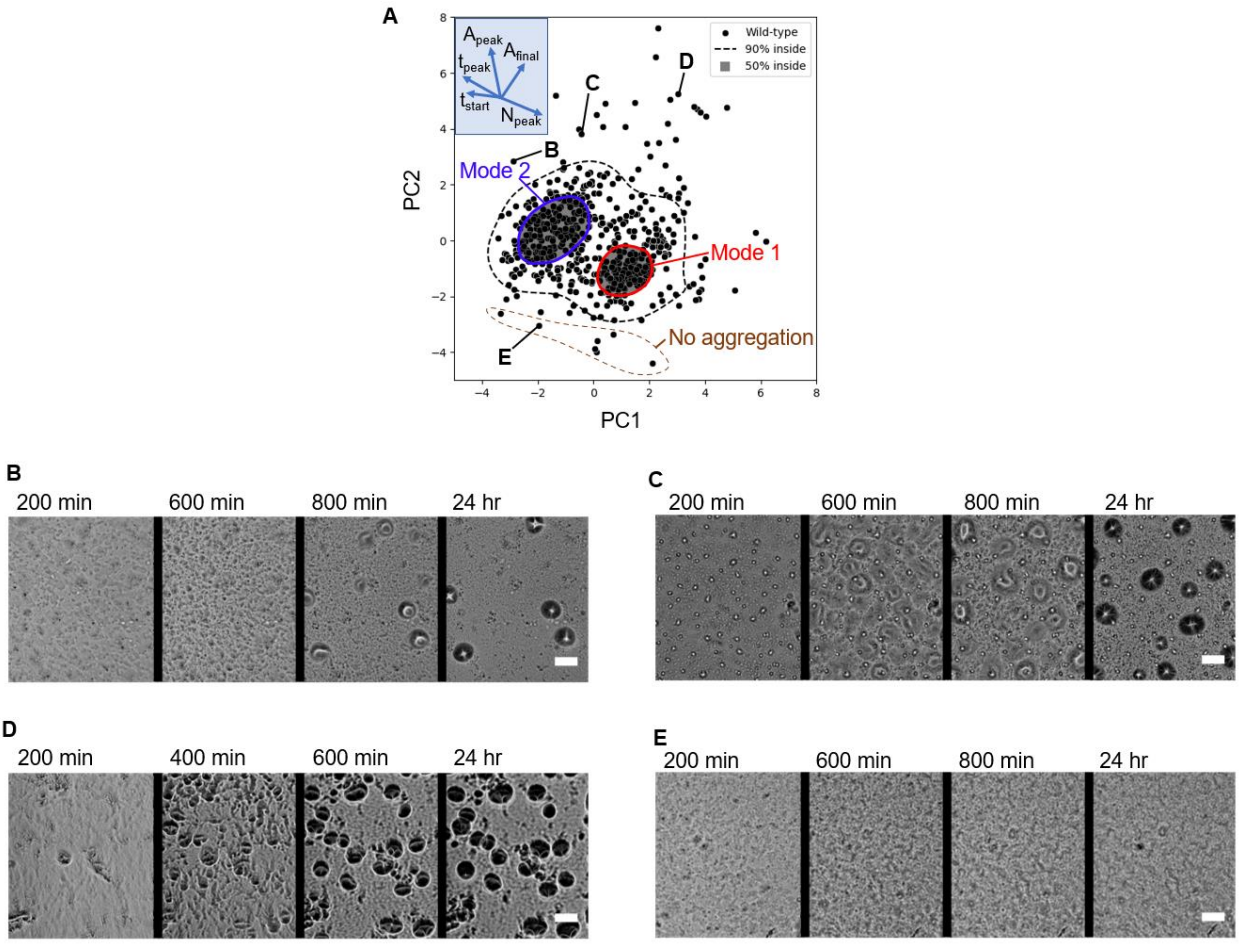
Supplementary Table 4.2. Numerical definition of PC1 and PC2 for reproducibility.

Metric Name	**Mean value (WT dataset)**	**Standard dev. (WT dataset)**	**Weight (PC1)**	**Weight (PC2)**
Start time (min)	291	90.0	-0.387	0.047
Peak time (min)	843	454	-0.430	0.245
Stability time (min)	1008	486	0.183	0.038
Mean area at peak time (μm^2)	5414	2063	-0.126	0.574
Std area at peak time (μm^2)	4670	3525	0.188	0.465
Final mean area (μm^2)	6617	1690	0.255	0.401
Final std area (μm^2)	4244	1972	0.281	0.424
N at peak time*	113	75.6	0.468	-0.198
Final N*	61.7	27.8	0.149	0.062
Fraction lost	0.463	0.253	0.443	-0.083

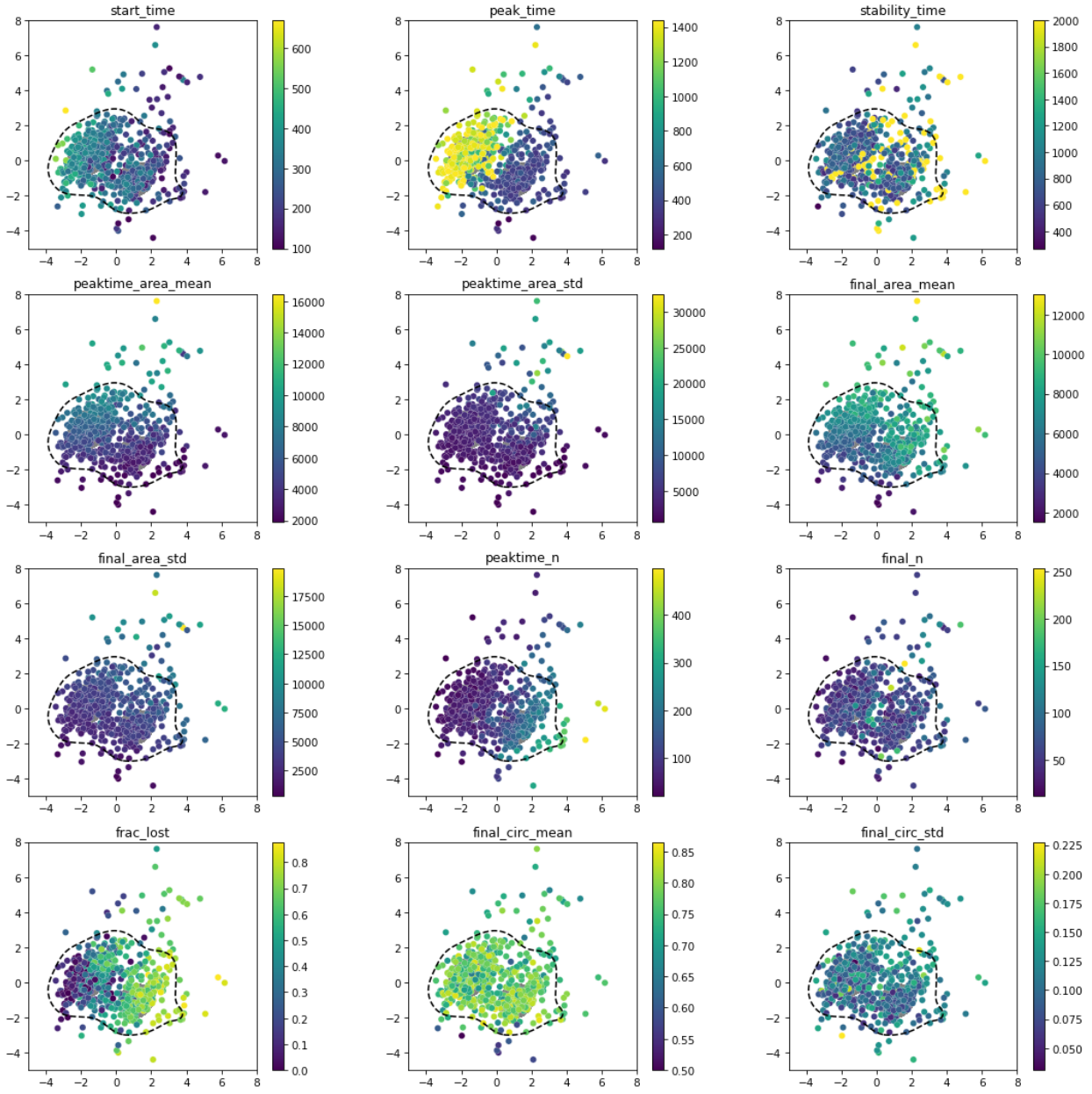
*Number of aggregates is reported in a field size of area 5.0 mm^2. Different field sizes should scale aggregate count appropriately, assuming a constant density of aggregates per mm^2.

Supplementary Table 4.3. Description of each of the ten developmental metrics used to quantify aggregation phenotype.

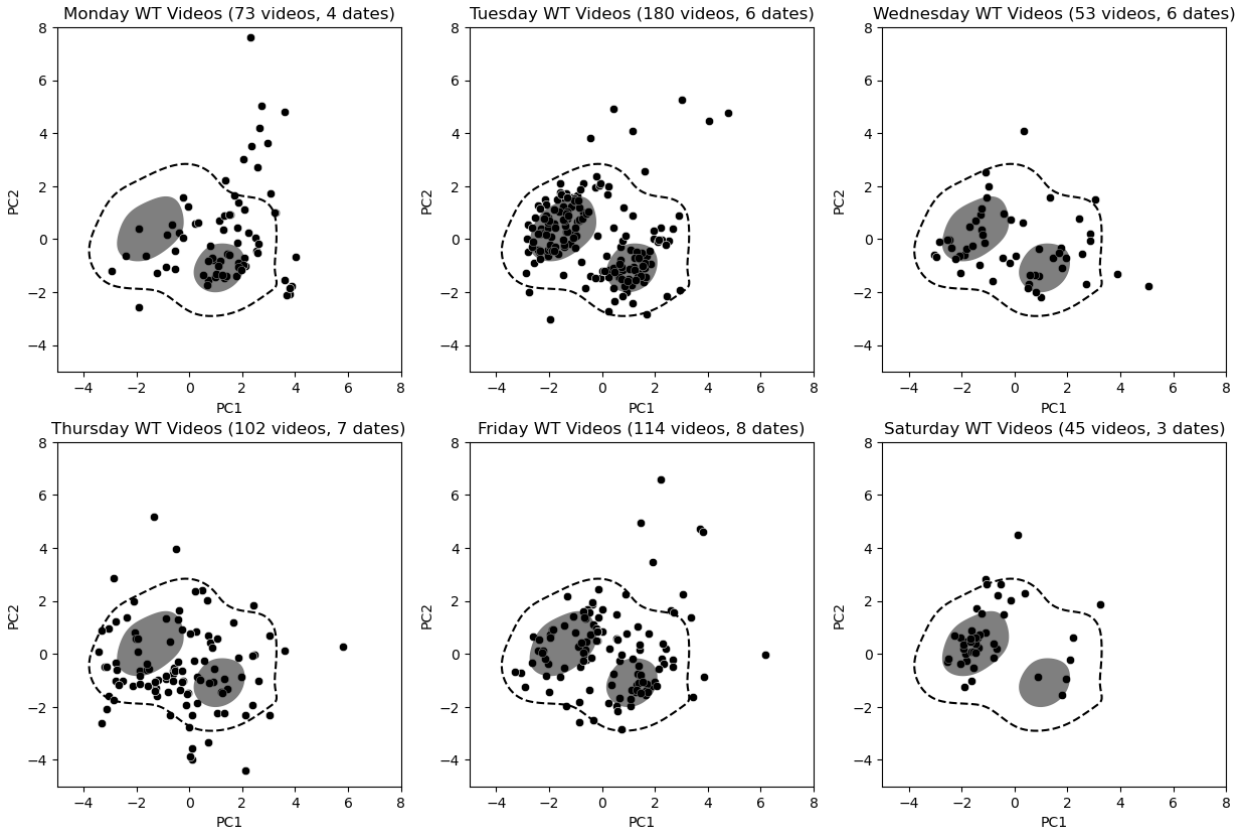
Metric Name	Description	Formula
Start time	The time elapsed between inoculation and the beginning of observable aggregation.	The earliest time at which at least ten aggregates have reached an area of at least 800 μm^2
Peak time	The time elapsed between inoculation and the moment aggregation reaches maximum total area	The time at which the sum of the areas of all aggregates in one frame is at a maximum value across the time series
Stability time	The time elapsed between inoculation and the moment the number of aggregates becomes stable	If stability is achieved within 24 hours, the time at which the rate of change of number of aggregates falls below and stays below 0.5 per minute. Rate of change is calculated with a Savitsky-Golay filter using a window 31 minutes wide. Otherwise, 2000 minutes, to represent a later eventual stability time.
Mean area at peak time	The average area of all aggregates at the moment of peak time	Peak time is evaluated, and the average area of each aggregate in that one frame is calculated
Std area at peak time	The standard deviation in the area of all aggregates at the moment of peak time	Peak time is evaluated, and the sample standard deviation of each aggregate area in that one frame is calculated
Final mean area	The average area of all aggregates 24 hours after inoculation	For the one frame showing 24 hours after inoculation, the average area of each aggregate is calculated
Final std area	The standard deviation in the area of all aggregates 24 hours after inoculation	For the one frame showing 24 hours after inoculation, the sample standard deviation of each aggregate area is calculated
N at peak time	The number of aggregates at peak time	Peak time is evaluated, and all aggregates are counted in the 5.0 mm^2 visible field in that one frame
Final N	The number of aggregates 24 hours after inoculation	All aggregates in the 5.0 mm^2 visible field are counted in the one frame showing 24 hours after inoculation
Fraction lost	The fractional change in number of aggregates from the moment of maximum number to 24 hours after inoculation.	For N_max the maximum number of aggregates (smoothed by averaging the three largest values) and N_24 the number of aggregates 24 hours after inoculation, (N_max – N_final) / N_max



Supplementary Figure 4.7. Abnormal phenotypes expressed in wild-type. (A) Wild-type phenotype profile, showing the location in PCA space of four instances of illustrative, abnormal behavior in wild-type replicates. **(B)** An extreme instance of Mode 2 behavior, especially due to late start time and the small number of final aggregates. Successful aggregates that are less mature than these are expressed seldom in wild-type. **(C&D)** Time series of abnormal behavior show the larger aggregates that form in the region with high PC2 values, but with significant differences in timing and number depending on the value of PC1. (**E)** A representative time series of failed aggregate formation. Bacterial activity is visible, but aggregates of any significant size or darkness do not form after 24 hours. Scale bar 100 µm.



Supplementary Figure 4.8. Variation of metrics across wild-type dataset. This figure illustrates how PC1 and PC2 organize the variation of phenotypic metrics in PCA space. Of the ten metrics used in the PCA, eight display a particular gradient direction. Only two, stability time and final number of aggregates, do not display a clear gradient direction but vary in a more complex pattern. For instance, high stability time (i.e. those time series that do not stabilize by 24 hours) and high final number of aggregates mostly present outside the regions of Mode 1 and Mode 2 (See Fig. 4.3 in the main text). Although not included in the PCA, the mean and standard deviation of aggregate circularity at 24 hours is also reported, not showing any clear correlation with the other metrics.



Supplementary Figure 4.9. Variation of wild-type videos based on date of inoculation. Differences were observed in the distribution of wild-type outcomes depending on the day the time lapse was started. This may represent an effect of the length of time *M. xanthus* bacteria spend in a colony on agar before being transferred to liquid culture and eventually inoculated on the non-nutritive agar used in fruiting body development assays. A shift from Mode 1 to Mode 2 being favored occurs when comparing Monday to Saturday development videos.

5. Conclusion

We have demonstrated that focusing on the mechanical relationship between bacterial colonies and the substrates they colonize reveal novel effects by which bacteria increase the forces they exert for colony expansion.

Future work can use techniques such as RNA sequencing to identify the genes that change expression in response to substrate stiffness. This may reveal new signaling pathways that cause the change in exerted force. Fluorescence microscopy techniques such as FRAP can now be used to track the assembly and disassembly of pili in real time [41] to provide evidence for the involvement of type IV pili in proposed pathways. Other experiments should test the poroelastic mechanism proposed in Chapter 2, for example by introducing tracer particles into the substrate fluid to track its flow. This would allow the measurement of the correlation between substrate displacement and fluid flow. A detailed simulation of the biofilm and substrate using finite element methods could validate whether this mechanism was sufficient to explain the change in force with substrate stiffness. Additional factors that have a physical effect should be screened, such as the production of surfactants by the bacteria that affect local surface tension, the production of the osmolytes that induce fluid flow, and the amount of viscoelasticity in the substrate, expanding beyond the purely elastic substrates used in this work.

We have also developed new tools to approach the genotype-phenotype problem in bacterial multicellular development and demonstrated their efficacy in providing quantitative evidence

for long-standing hypotheses such as the existence of networks of redundant genes, and in distinguishing subtle phenotypic effects.

Future studies can use these tools to intuit the function of genes that have resisted characterization by correlating their phenotypic fingerprint with that of genes with known functions. The established library of single-gene knockouts in *Myxococcus xanthus* can be used to compile the necessary repository of phenotypic fingerprints. These tools will also aid in measuring the impact of the transduction of mechanical signals when fruiting body development is observed in varying environmental conditions, such as on PAA or agar substrates with varying stiffness. There is still much to uncover in the mechanical interactions that connect genotype and phenotype.

Appendix A: Protocols

Preparation of coverslips for polyacrylamide gels

Adapted from Katrina Cruz, Janmey Lab (2017)

1. Solution preparation

a. Prepare a 10% SurfaSil solution in acetone in a small glass beaker (less than 5-10 ml).

2. Preparation of top coverslips (Coverslips that do not stick to gel)

a. Carefully add ~20 large glass coverslips to the beaker containing 10% SurfaSil.

b. Swirl for a few minutes.

c. Using tweezers, move the coverslips to a beaker containing a small amount of acetone, swirl for a few seconds.

d. Using tweezers, move the coverslips to a beaker containing a small amount of methanol, swirl for a few seconds.

e. Air dry the coverslips on kimwipes in the biological safety cabinet.

f. Store siliconized coverslips at room temperature.

h. Can Re-use coverslips (4 or 5 times) and then re-siliconize them

3. Preparation of bottom coverslips (Coverslips that do stick to gel)

a. Place small coverslips in coverslip racks.

b. Cover the coverslips with 0.1 M NaOH and incubate for 3 minutes. Decant and save the NaOH

c. Working in the chemical hood, cover the coverslips with 3-APTMS, using the smallest glass beakers that fit the coverslip racks to conserve 3-APTMS. Incubate for 3 minutes and then decant 3-APTMS for reuse or disposal.

d. Rinse the coverslips once with deionized water. Wash the coverslips with deionized water three times for 10 minutes each wash.

e. Transfer racks w/ coverslips to new clean container

f. Cover coverslips completely with 0.5% glutaraldehyde solution (stored in 4°C fridge) and incubate for 30 minutes in the chemical hood. Decant the glutaraldehyde for reuse.

g. Rinse the coverslips once with deionized water. Wash the coverslips with deionized water three times, 10 minutes each wash.

h. Place coverslips individually on kimwipes to air dry.

i. Store in the desiccator under vacuum. Can store under vacuum for about one month (especially when preparing stiffer, 30 kPa, gels; for softer gels that don't swell as much these coverslips may last longer)

Materials

Surfasil; Thermo Scientific TS42801 or TS-42800

(3-aminopropyl)trimethoxysilane (3-APTMS) 97%; Sigma-Aldritch 281778-500ML

Glutaraldehyde 50%; Fisher Scientific G151 1 *(dilute to 0.5% with distilled water)*

18 mm circular glass coverslips; Fisher Scientific 12-546-P

22 mm circular glass coverslips; Fisher Scientific 12-546-1P

Preparation of phosphate-buffered-saline (PBS)

Adapted from protocolsonline.com

For 1 liter of 1X PBS, prepare as follows:

(1) Start with 800 ml of distilled water:

(2) Add 8 g of NaCl.

(3) Add 0.2 g of KCl.

(4) Add 1.44 g of Na_2HPO_4. (2.72 g of $Na_2HPO_4 x7H_20$)

(5) Add 0.24 g of KH_2PO_4.

(6) Adjust the pH to 7.4 with HCl.

(7) Add distilled water to a total volume of 1 liter.

Dispense the solution into aliquots and sterilize by autoclaving (20 min, 121°C, liquid cycle).

Store at room temperature.

Preparation of TPM Buffer

For 1L of TPM buffer, prepare as follows:

(1) Add ~800mL distilled water to a flask

(2) Add 10mL $MgSO_4$

(3) Add 10mL Tris pH 7.6

(4) Add 1mL KPO_4

(5) Bring up to 1L by adding distilled water

(6) Mix, titrate pH to ~7.6 and autoclave

Store at room temperature.

NOTE: pH of Tris used in CTTYE is 8.0 while for TPM it's 7.6

Polyacrylamide gel synthesis protocol for bacteria

Varying PAA gel formulas **(each for three 200 µL gels)** with 5% extra volume for error

	3.5% PAA (0.15% Bis)
Distilled water	521
Acrylamide	55.1
Bis-acrylamide	47.3
TEMED	1.58
APS	4.73
Total	**630 µL**

	8% PAA (0.15% Bis)
Distilled water	451
Acrylamide	126.0
Bis- acrylamide	47.3
TEMED	1.58
APS	4.73
Total	**630 µL**

If making gels for traction force measurements, instead of distilled water, use a 20:1 dilution of 2 µm diameter Fluoro-Max beads:

1. Prepare an appropriate volume of fully concentrated Fluoro-Max beads in a mini-centrifuge tube

   (For a target of 1000 µL of diluted bead solution, get 50 µL of fully concentrated beads)
2. Centrifuge at 5000 rpm for 1 minute or until the beads concentrate into a pellet
3. Keeping track of the volume, remove the supernatant (a surfactant that is harmful to bacteria)

   (For 50 µL of fully concentrated beads, about 40 µL of supernatant can be removed)
4. Replace the same volume of removed supernatant with distilled water. This produces a fully concentrated bead solution suspended in mostly distilled water instead of surfactant.
5. Add the appropriate volume of distilled water for a 20:1 dilution

   (For a target of 1000 µL of diluted bead solution, add 950 µL of distilled water)
6. Using a pipette tip, gently mix the resulting dilution until homogeneous.

7. Store protected from light, i.e. wrapped in aluminum foil

Gel synthesis

1. Mix the distilled water, acrylamide, and bis-acrylamide in mini-centrifuge tubes
2. Add TEMED to solution, using the fume hood
3. Vortex solution to mix for about ten seconds
4. Lay out glutaraldehyde cover slips on a working surface (i.e. glass board) and have SurfaSil cover slips ready
5. Add APS to solution and vortex for about 5 seconds (do this step and the following ones quickly)
6. Pipette 200 µL gel solution onto the glutaraldehyde coverslips, then gently place SurfaSil cover slips on top with tweezers

   Tip: be careful to simply release the coverslips and not press downwards
7. Let polymerize for about 10 minutes, then rehydrate each gel by pipetting ~20 µL of distilled water around the edge of the gel, capillary forces should suck it into the gel
8. After 20 minutes total of polymerization time, carefully remove the SurfaSil cover slip and set it aside for reuse. Each gel should stay adhered to the glutaraldehyde cover slip.
9. Place gels in a well plate and immerse in PBS (or TPM for *Myxococcus xanthus*)

Gels can be stored immersed in PBS in well plates sealed with parafilm at 4°C for about one week.

Gel washing process

Each polyacrylamide gel should be washed in an appropriate buffer (TPM for *Myxococcus xanthus*, 1X PBS for other species) and then washed again in an appropriate nutrient medium (CTTYE for *Myxococcus xanthus*, LB for other species) before inoculating them with bacteria.

1. Quick wash: Ensure gels are immersed in buffer. Use aspirator to remove buffer from well plates, then immerse gels again with fresh buffer
2. 10-minute wash: Place well plates on the plate shaker for 10 minutes, then use aspirator to remove buffer. Add fresh buffer again
3. Overnight wash: Seal well plates in parafilm and store at 4°C overnight

Repeat these steps the following day, using three washes in nutrient medium.

Now the gels can be used for the sterilization and inoculation protocol.

Materials

- Acrylamide (40% w/v); Fisher Scientific BP1402 1 – in 4°C fridge in aliquots
- Bis-acrylamide (2% w/v); Fisher Scientific BP1404 250 – in 4°C fridge in aliquots
- Ammonium persulfate (APS, 10% w/v); Fisher Scientific 45-000-225 – in -20°C freezer in aliquots, single use
- Tetramethylethylenediamene (TEMED, 99%); Fisher Scientific AC138450500 – in fume hood, must be handled inside fume hood
- SurfaSil treated cover slips – see cover slip protocol for polyacrylamide gels
- Glutaraldehyde treated cover slips – see cover slip protocol for polyacrylamide gels

Inoculating PAA gels with bacteria

Required materials: Active liquid bacteria culture begun the day before (within 18 to 24 hours), PAA gels that have been undergoing an overnight wash in LB medium.

Ideally, all gel drying and sterilization should be performed in a biosafety cabinet.

Inoculation of bacteria can be performed outside the biosafety cabinet to prevent contamination.

Turn on the microscope incubator to bring it up to temperature (37°C) and allow it to equilibrate while you perform the following procedure.

1. Remove LB medium from well plate with an aspirator.
    a. Then tilt the well plate so that any LB medium remaining on the gels pools on one side of the gel surface, and gently remove the medium from each gel surface. The gel surface should have as little visible liquid as possible, but be careful not to scratch the gels too much.
2. Allow gels to dry, exposed to still air, for 20 minutes (i.e. lower the hood if using a biosafety cabinet).
3. Using either UV from the biosafety cabinet or an external UV lamp, expose the gel surfaces to UV sterilization for 20 minutes. The gel surfaces should now have no visible liquid.

4. After thoroughly mixing liquid bacterial culture by repeatedly pipetting up and down, dispense 5 µL in a single, small droplet in the middle of each gel.

Tip: Hold the micropipette above the gel surface, then dispense the 5 µL droplet, which should still be stuck to the pipette tip. Then gently lower the pipette tip until it contacts the gel surface.

5. From each liquid culture droplet, remove 2 µL to flatten the droplets.

The gels are now ready to begin imaging in an incubated chamber heated to 37°C.

Appendix B: Python Code

Script for combining automated boundary detection with manual validation

System-specific dependencies and other information can be found on Github at

https://github.com/masp01/SUBII-Trace

```
import numpy as np
from tkinter import filedialog
from tkinter import messagebox
from tkinter import *
from FindFeature import Boundaries
from ensureDataPath import ensureDataPath
import matplotlib.pyplot as plt
import os

class App:
	def __init__(self):
		# File access
		self.contourFile = 'trial{0}xy{1:02d}_{2:02d}.txt'
		self.paths = ensureDataPath().paths

		self.root = Tk()		# application window
		self.i = 0			# frame index
		self.c = 0			# contour index
		self.j = 0			# segment index
		self.xmin = np.inf		# display limits
		self.xmax = -1			# ''
		self.ymin = np.inf		# ''
		self.ymax = -1			# ''
		self.drawState = False		# is the user drawing points?
		self.drawnLine = None # matplotlib artist of user-drawn line
		self.drawnXs = []		# coordinates of user-drawn line
		self.drawnYs = []		# ''
```

```
                # Calculate boundary predictions
                self.trial = 5 # default value
                self.xy = 28   # ''
                self.b = Boundaries(trial=self.trial, xy=self.xy, edgeMin=50, edgeMax=50,
boost=5)
                self.c = self.b.frames[self.i].c

                # Store user-drawn points
                self.boundaryLine = None
                self.boundaryXs = []
                self.boundaryYs = []
                self.insertionIndices = []

                # Disable some default hotkeys
                if 's' in plt.rcParams['keymap.save']:
                        plt.rcParams['keymap.save'].remove('s')
                if 'f' in plt.rcParams['keymap.fullscreen']:
                        plt.rcParams['keymap.fullscreen'].remove('f')
                if 'left' in plt.rcParams['keymap.back']:
                        plt.rcParams['keymap.back'].remove('left')
                if 'right' in plt.rcParams['keymap.forward']:
                        plt.rcParams['keymap.forward'].remove('right')

                # Create all frames
                TrialControls = Frame(self.root)
                ViewControls = Frame(self.root)
                EdgeControls = Frame(self.root)
                FrameControls = Frame(self.root)
                ContourControls = Frame(self.root)
                SegmentControls = Frame(self.root)

                # Organize frames vertically in order
                TrialControls.pack(padx=5, pady=5)
                ViewControls.pack(padx=5, pady=5)
```

```
        EdgeControls.pack(padx=5, pady=10)
        FrameControls.pack(padx=5)
        ContourControls.pack(padx=5)
        SegmentControls.pack(padx=5)

        # Window title
        self.root.title("")

        # Widgets in TrialControls
        Label(TrialControls, text='Trial').grid(row=0)
        self.trialEntry = Entry(TrialControls, width=5)
        self.trialEntry.insert(0, self.trial)
        self.trialEntry.grid(row=0, column=1)
        Label(TrialControls, text='xy').grid(row=0, column=2)
        self.xyEntry = Entry(TrialControls, width=5)
        self.xyEntry.insert(0, self.xy)
        self.xyEntry.grid(row=0, column=3)
        Button(TrialControls, text="Load", command=self.loadData).grid(row=0, column=4)

        # Widgets in EdgeControls
        Label(EdgeControls, text="edgeMin").grid(row=0)
        self.edgeMinEntry = Entry(EdgeControls, width=5)
        self.edgeMinEntry.insert(0, 50)
        self.edgeMinEntry.grid(row=0, column=1)
        Label(EdgeControls, text="edgeMax").grid(row=1)
        self.edgeMaxEntry = Entry(EdgeControls, width=5)
        self.edgeMaxEntry.insert(0, 50)
        self.edgeMaxEntry.grid(row=1, column=1)
        Button(EdgeControls, text="Redo Edges", command=self.redoEdges).grid(row=0,
column=2)
        Button(EdgeControls, text="Histogram", command=self.makeHistogram).grid(row=1,
column=2)

        # Widgets in ViewControls
        self.viewMode = StringVar()
        self.viewMode.set('img')
```

```
                Radiobutton(ViewControls, text="Image", variable=self.viewMode, value='img',
command=self.showFrame).pack()

                Radiobutton(ViewControls, text="Edges", variable=self.viewMode, value='edges',
command=self.showFrame).pack()

                Radiobutton(ViewControls, text="Fill", variable=self.viewMode, value='fill',
command=self.showFrame).pack()

                self.adaptZoom = IntVar()

                Checkbutton(ViewControls, text="Adapt Zoom", variable=self.adaptZoom,
command=self.showFullFrame).pack()



                # Widgets in FrameControls

                prevFrameButton = Button(FrameControls, text="<", command=lambda:
self.onClickFrame(-1))

                nextFrameButton = Button(FrameControls, text="Next Frame >", command=lambda:
self.onClickFrame(1))

                self.frameText = StringVar()

                self.frameText.set("{0}/{1}".format(self.i + 1, len(self.b.frames)))

                Label(FrameControls, textvariable=self.frameText).pack(side=BOTTOM)

                prevFrameButton.pack(side=LEFT)

                nextFrameButton.pack(side=LEFT)



                # Widgets in ContourControls

                prevContourButton = Button(ContourControls, text="<", command=lambda:
self.onClickContour(-1))

                nextContourButton = Button(ContourControls, text="Next Contour >",
command=lambda: self.onClickContour(1))

                self.contourText = StringVar()

                self.contourText.set("{0}/{1}".format(self.c + 1,
len(self.b.frames[self.i].contours)))

                Label(ContourControls, textvariable=self.contourText).pack(side=BOTTOM)

                prevContourButton.pack(side=LEFT)

                nextContourButton.pack(side=LEFT)



                # Widgets in SegmentControls

                prevSegmentButton = Button(SegmentControls, text="<", command=lambda:
self.onClickSegment(-1))

                nextSegmentButton = Button(SegmentControls, text="Next Segment >",
command=lambda: self.onClickSegment(1))

                self.segmentText = StringVar()
```

```
                self.segmentText.set("{0}/{1}".format(self.j + 1,
len(self.b.frames[self.i].segments)))
                Label(SegmentControls, textvariable=self.segmentText).pack(side=BOTTOM)
                nextSegmentButton.pack(side=RIGHT)
                prevSegmentButton.pack(side=RIGHT)

                self.fig, self.ax = plt.subplots()
                self.fig.canvas.manager.set_window_title('Biofilm Image')
                plt.axis('off')
                plt.tight_layout()
                displayData = getattr(self.b.frames[self.i], self.viewMode.get())
                self.im = self.ax.imshow(displayData, cmap='gray')
                self.showFrame()
                self.fig.canvas.mpl_connect('pick_event', self.onPickDataPoint) # Enable picking
data points
                self.fig.canvas.mpl_connect('button_press_event', self.onClick)
                self.fig.canvas.mpl_connect('button_release_event', self.onRelease)
                self.fig.canvas.mpl_connect('axes_leave_event', self.onLeaveAxes)
                self.fig.canvas.mpl_connect('key_press_event', self.onKey)
                plt.connect('motion_notify_event', self.drawPoints)
                plt.show()

                self.root.mainloop()

        def showFrame(self):
                # Get the coordinates of the predicted boundary
                xs, ys = self.b.frames[self.i].segments[self.j]
                self.boundaryXs = xs
                self.boundaryYs = ys

                # Set the display limits
                padding = 20
                xmin = min(xs) - padding
                self.xmin = min(xmin, self.xmin)
                self.xmin = max(0, self.xmin)
                xmax = max(xs) + padding
```

```
                self.xmax = max(xmax, self.xmax)
                self.xmax = min(self.b.frames[self.i].width - 1, self.xmax)
                ymin = min(ys) - padding
                self.ymin = min(ymin, self.ymin)
                self.ymin = max(0, self.ymin)
                ymax = max(ys) + padding
                self.ymax = max(ymax, self.ymax)
                self.ymax = min([self.b.frames[self.i].height - 1, self.ymax])

                # Display the frame and predicted boundary
                displayData = getattr(self.b.frames[self.i], self.viewMode.get())
                self.im.set_data(displayData)
                if self.boundaryLine is not None:
                        self.boundaryLine.remove()
                        self.boundaryLine = None
                self.boundaryLine, = self.ax.plot(xs, ys, linewidth=4, color='#1f77b460',
picker=5) # tolerance of 5 pixels for clicking data points
                if self.adaptZoom.get():
                        plt.xlim(self.xmin, self.xmax)
                        plt.ylim(self.ymax, self.ymin)
                self.fig.canvas.draw() # This line is necessary to update the figure


        def showFullFrame(self):
                # updates zoom level appropriately when changing
                # the Adapt Zoom checkbox
                if not self.adaptZoom.get():
                        self.ax.clear()
                        displayData = getattr(self.b.frames[self.i], self.viewMode.get())
                        self.im = self.ax.imshow(displayData, cmap='gray')
                        plt.axis('off')
                        plt.tight_layout()
                self.showFrame()


        def rollback(self, prevTrial, prevXy):
                self.trial = prevTrial
```

```
                self.xy = prevXy
                self.trialEntry.delete(0, END)
                self.trialEntry.insert(0, '{}'.format(self.trial))
                self.xyEntry.delete(0, END)
                self.xyEntry.insert(0, '{}'.format(self.xy))

        def loadData(self):
                prevTrial = self.trial
                prevXy = self.xy
                # Check that the given trial and xy are valid numbers
                try:
                        self.trial = int(self.trialEntry.get())
                        self.xy = int(self.xyEntry.get())
                        # Check that images exist for the given trial and xy
                        try:
                                self.redoEdges(showFrame=False)
                        except IOError:
                                print("No images found for this trial and xy.")
                                self.rollback(prevTrial, prevXy)
                                self.redoEdges(showFrame=False)
                except ValueError:
                        self.rollback(prevTrial, prevXy)

                # Update image
                displayData = getattr(self.b.frames[self.i], self.viewMode.get())
                self.im = self.ax.imshow(displayData, cmap='gray')

                # Print a warning if contours have already been traced for this image
                if os.path.exists(self.paths.contours[self.trial].format(self.xy) +
self.contourFile.format(self.trial, self.xy, self.i+1)):
                        print("This contour has already been traced. Saving will overwrite
previous data.")

                # Update frame label
                self.frameText.set("{0}/{1}".format(self.i + 1, len(self.b.frames)))
```

```
                # Reset display limits
                self.xmin, self.ymin = np.inf, np.inf
                self.xmax, self.ymax = -1, -1

                # Reset to check the first frame
                self.i = 0
                self.showFrame()

        def makeHistogram(self):
                # Display a histogram of the gray values of the current image
                # Most useful when zoomed into the biofilm boundary,
                # to estimate edgeMin and edgeMax values
                xlims = self.ax.get_xlim()
                ylims = self.ax.get_ylim()
                img = self.im.get_array()
                grayValues = img[int(min(ylims)):int(max(ylims)),
int(min(xlims)):int(max(xlims))].flatten()
                self.histFig = plt.figure('Gray Value Histogram')
                plt.hist(grayValues)
                self.histFig.canvas.draw()
                plt.show()

        def redoEdges(self, showFrame=True):
                self.b = Boundaries(trial=self.trial, xy=self.xy,
edgeMin=int(self.edgeMinEntry.get()), edgeMax=int(self.edgeMaxEntry.get()), boost=5)

                # Reset to check the first contour and segment
                self.c = self.b.frames[self.i].c
                self.j = 0

                # Update labels
                self.contourText.set("{0}/{1}".format(self.c + 1,
len(self.b.frames[self.i].contours)))
                self.segmentText.set("{0}/{1}".format(self.j + 1,
len(self.b.frames[self.i].segments)))

                if showFrame:
```

```
                        self.showFrame()


        def onClick(self, event):
                if self.fig.canvas.manager.toolbar.mode == '':
                        self.drawState = True
                        self.drawPoints(event)


        def onLeaveAxes(self, event):
                if self.fig.canvas.manager.toolbar.mode != '':
                        return
                if self.drawState:
                        x = event.xdata
                        y = event.ydata
                        self.drawPoints(event)
                        self.drawState = False
                        if self.drawnLine is not None:
                                self.drawnLine.remove()
                                self.drawnLine = None
                        self.fig.canvas.draw()
                        if len(self.insertionIndices) == 1: # a new insertion was drawn from a
boundary point to the edge of the image
                                i = self.insertionIndices[0]


                                # determine which end of the boundary to change
                                # and add the drawn points to the end of the boundary
                                if i < int(np.floor(len(self.boundaryXs)/2)):
                                        keptBoundaryXs = self.boundaryXs[i:]
                                        keptBoundaryYs = self.boundaryYs[i:]
                                        self.boundaryXs = np.concatenate((self.drawnXs[::-1],
keptBoundaryXs))
                                        self.boundaryYs = np.concatenate((self.drawnYs[::-1],
keptBoundaryYs))
                                else:
                                        keptBoundaryXs = self.boundaryXs[:i+1]
                                        keptBoundaryYs = self.boundaryYs[:i+1]
                                        self.boundaryXs = np.concatenate((keptBoundaryXs,
self.drawnXs))
```

```
                                        self.boundaryYs = np.concatenate((keptBoundaryYs,
self.drawnYs))


                                # redraw the boundary

                                self.boundaryLine.remove()

                                self.boundaryLine, = self.ax.plot(self.boundaryXs,
self.boundaryYs, linewidth=4, color='#1f77b460', picker=5)

                                self.fig.canvas.draw()


        def onRelease(self, event):

                if self.fig.canvas.manager.toolbar.mode != '':

                        return

                self.drawPoints(event)

                self.drawState = False

                self.boundaryLine.pick(event) # run the picker

                if self.drawnLine is not None:

                        self.drawnLine.remove()

                        self.drawnLine = None

                #self.ax.plot(self.drawnXs, self.drawnYs, linewidth=4, color='#e4211c60')

                self.fig.canvas.draw()

                if len(self.insertionIndices) == 2: # a new insertion was drawn that touched the
boundary at two points

                        firstBookend = np.array(range(min(self.insertionIndices) + 1))

                        secondBookend = np.array(range(max(self.insertionIndices),
len(self.boundaryXs)))


                        # check if points were drawn in the opposite direction of the boundary
points

                        defaultDist = np.sqrt((self.drawnXs[0] - self.boundaryXs[firstBookend[-
1]])**2 + (self.drawnYs[0] - self.boundaryYs[firstBookend[-1]])**2)

                        flippedDist = np.sqrt((self.drawnXs[0] -
self.boundaryXs[secondBookend[0]])**2 + (self.drawnYs[0] -
self.boundaryYs[secondBookend[0]])**2)

                        if flippedDist < defaultDist:

                                self.drawnXs = np.array(self.drawnXs)[::-1]

                                self.drawnYs = np.array(self.drawnYs)[::-1]


                        # insert the drawn points inbetween the remainder of the boundary
```

```
                        self.boundaryXs = np.concatenate((self.boundaryXs[firstBookend],
self.drawnXs, self.boundaryXs[secondBookend]))

                        self.boundaryYs = np.concatenate((self.boundaryYs[firstBookend],
self.drawnYs, self.boundaryYs[secondBookend]))

                        self.boundaryLine.remove()

                        self.boundaryLine, = self.ax.plot(self.boundaryXs, self.boundaryYs,
linewidth=4, color='#1f77b460', picker=5)

                        self.fig.canvas.draw()


                # Reset drawn points

                self.insertionIndices = []

                self.drawnXs = []

                self.drawnYs = []


        def drawPoints(self, event):

                if self.drawState:

                        self.drawnXs.append(event.xdata)

                        self.drawnYs.append(event.ydata)

                        if self.drawnLine is not None:

                                self.ax.lines.remove(self.drawnLine)

                                self.drawnLine = None

                        self.drawnLine, = self.ax.plot(self.drawnXs, self.drawnYs, linewidth=4,
color='#e4211c30')

                        self.fig.canvas.draw()


        def onKey(self, event):

                print(event.key)

                if event.key == 's':

                        print('Got save event')

                        filename = self.paths.contours[self.trial].format(self.xy) +
self.contourFile.format(self.trial, self.xy, self.i+1)

                        if not os.path.exists(self.paths.contours[self.trial].format(self.xy)):

                                os.makedirs(self.paths.contours[self.trial].format(self.xy))

                        with open(filename, 'w') as txt_file:

                                for i in range(len(self.boundaryXs)):

                                        txt_file.write(str(self.boundaryXs[i]) + '\t' +
str(self.boundaryYs[i]) + '\n')
```

```
                elif event.key == 'right':
                        self.onClickFrame(1)
                elif event.key == 'left':
                        self.onClickFrame(-1)
                elif event.key == 'f':
                        self.onClickContour(1)


        def onPickDataPoint(self, event):
                line = event.artist
                xs = line.get_xdata()
                ys = line.get_ydata()
                indices = np.array(event.ind)
                ind = indices[int(np.floor(len(indices)/2))] # choose the index of the center
point
                self.insertionIndices.append(ind)
                #points = tuple(zip(xs[ind], ys[ind]))
                #print('onpick points: {0}'.format(points))


        def onClickFrame(self, step):
                # Move one frame
                self.i += step
                self.i %= len(self.b.frames) # loop back to the first frame


                # Reset to check the first contour and segment
                self.c = self.b.frames[self.i].c
                self.b.frames[self.i].segments =
self.b.frames[self.i].splitSegments(self.b.frames[self.i].contours[self.c])
                self.j = 0


                # Update labels
                self.frameText.set("{0}/{1}".format(self.i + 1, len(self.b.frames)))
                self.contourText.set("{0}/{1}".format(self.c + 1,
len(self.b.frames[self.i].contours)))
                self.segmentText.set("{0}/{1}".format(self.j + 1,
len(self.b.frames[self.i].segments)))


                # Reset the display limits when on the first frame
```

```
                if self.i == 0:
                        self.xmin, self.ymin = np.inf, np.inf
                        self.xmax, self.ymax = -1, -1

                # Print a warning if contours have already been traced for this image
                if os.path.exists(self.paths.contours[self.trial].format(self.xy) +
self.contourFile.format(self.trial, self.xy, self.i+1)):
                        print("This contour has already been traced. Saving will overwrite
previous data.")

                # Show the frame and boundary
                self.showFrame()

        def onClickContour(self, step):
                # Move one contour
                self.c += step
                self.c %= len(self.b.frames[self.i].contours) # loop back to the first contour

                # Find the first contour that touches the image edges and split it into segments
                contourRange = np.array(range(len(self.b.frames[self.i].contours)))
                if step == 1:
                        contourRange = np.concatenate((contourRange[self.c:],
contourRange[:self.c]))
                else:
                        contourRange = np.concatenate((contourRange[self.c::-1],
contourRange[self.c+1:][::-1]))
                for thisC in contourRange:
                        self.b.frames[self.i].segments =
self.b.frames[self.i].splitSegments(self.b.frames[self.i].contours[thisC])
                        if len(self.b.frames[self.i].segments) > 0:
                                self.c = thisC
                                break

                # Warn the user if no such contour is found
                if len(self.b.frames[self.i].segments) == 0:
                        print("No contours found that touch the edges")
```

```
                    self.b.frames[self.i].segments =
[self.b.frames[self.i].contours[self.c]] # store contour coordinates anyway


                # Reset to check the first segment

                self.j = 0


                # Update labels

                self.contourText.set("{0}/{1}".format(self.c + 1,
len(self.b.frames[self.i].contours)))

                self.segmentText.set("{0}/{1}".format(self.j + 1,
len(self.b.frames[self.i].segments)))


                # Reset the display limits

                #self.xmin, self.ymin = np.inf, np.inf

                #self.xmax, self.ymax = -1, -1


                # Show the frame and boundary

                self.showFrame()


        def onClickSegment(self, step):

                # Move one segment

                self.j += step

                self.j %= len(self.b.frames[self.i].segments) # loop back to the first segment


                # Update segment label

                self.segmentText.set("{0}/{1}".format(self.j + 1,
len(self.b.frames[self.i].segments)))


                # Show the frame and boundary

                self.showFrame()


app = App()
```

Image processing and other tools for detection of *M. xanthus* aggregates

```
import numpy as np
import cv2
import os
import re
import pandas as pd
import time
import multiprocessing as mp
import matplotlib.pyplot as plt
import trackpy as tp
from data_paths import data_paths
from scipy.signal import savgol_filter
from scipy.spatial import ConvexHull, convex_hull_plot_2d
from scipy.optimize import minimize_scalar
from matplotlib.path import Path
import paramiko # pip install paramiko
from stat import S_ISDIR, S_ISREG
from datetime import datetime

def get_img(video_i, t):
    # Time t measured in minutes since inoculation
    dp = data_paths(video_i)
    img_path = dp.img_name.format(t+1) # Images start at #0001
    img = cv2.imread(img_path, 0)
    return img

# particles is a list of trackpy id numbers to include in the overlay,
# defaulting to all
def get_overlay_img(video_i, t, particles=None):
    dp = data_paths(video_i)
    img = get_img(video_i, t)
    img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGBA)
    if not os.path.exists(dp.contour_points_path.format(t)):
        return img
```

```
    contours = np.load(dp.contour_points_path.format(t), allow_pickle=True)
    if isinstance(particles, int):
        particles = [particles]
    df = get_df(video_i)
    # Remove spurious early aggregates
    early_p = np.unique(df[df.t < 100].particle.values)
    df = df[[p not in early_p for p in df.particle]]
    for i,contour in enumerate(contours):
        # Color based on particle id #
        if df is not None:
            p = df[(df.contour == i) & (df.t == t)].particle
            if len(p) == 0:
                continue
            else:
                p = int(p)
            c = np.array(plt.cm.tab10(p%10))*255
        else:
            c = np.array(plt.cm.tab10(i%10))*255
        if (particles is None) or (p in particles):
            img = cv2.drawContours(img, contours, i, c, 5)
    return img

def load_registry():
    registry = data_paths(0).registry
    return registry

def save_registry(r):
    r.to_csv(data_paths(0).registry_path)

def add_rows_to_registry(new_rows_df):
    # Ensure added rows have the appropriate columns
    r = load_registry()
    column_conditions = [col in new_rows_df.columns for col in r.columns]
    if not all(column_conditions):
        print('New rows added to the registry must have all of the following columns:')
```

```
        print(r.columns.values)
        return
    extra_columns = [col for col in new_rows_df.columns if col not in r.columns]
    if len(extra_columns) != 0:
        print('Columns not recognized:')
        print(extra_columns)
        return

    # Ensure no row corresponds to a video that is already in the registry
    failed_check = False
    for i in new_rows_df.index.values:
        run_num = new_rows_df.loc[i, 'run']
        scope_num = new_rows_df.loc[i, 'scope']
        registry_slice = r[(r.run == run_num) & (r.scope == scope_num)]
        if len(registry_slice) != 0:
            print('Run {} scope {} already exists in the registry.'.format(run_num,
scope_num))
            failed_check = True
    if failed_check:
        return

    # Append the new rows to the end of the registry
    new_r = pd.concat((r, new_rows_df))

    # Reindex the new rows
    new_r.index = np.arange(len(new_r))

    # Ensure no old videos were reindexed somehow
    if not all(new_r.loc[r.index, 'img_folder'] == r.img_folder):
        print('Reindexing failure. Try again or update registry manually.')

    # Save updated registry
    new_indices = [i for i in new_r.index.values if i not in r.index.values]
    print('New video indices added: {}'.format(new_indices))
    print('Saving new registry.')
```

```
    save_regsitry(new_r)

def update_processed_column():
    r = load_registry()
    n_processed_old = sum(r.processed)

    df_filenames = os.listdir(data_paths(0).video_df_folder)
    processed = [data_paths(i).videoname in df_filenames for i in r.index]

    n_difference = sum(processed) - n_processed_old
    print('{} new videos processed'.format(n_difference))

    r.processed = processed
    save_regsitry(r)
    print('Registry saved')
    return r

def ensure_final_slash(dir_path):
    if dir_path[-1] != '/':
        dir_path = dir_path + '/'
    return dir_path

# from a given search location, recursively
# search for all folders containing .tif files
def img_folders(search_path='.'):
    img_folders = []
    for root, dirs, files in os.walk(search_path):
        if any([file.endswith('.tif') for file in files]):
            img_folder = os.path.abspath(root)
            img_folders.append(img_folder)
    return img_folders

# Given a video index, get the dataframe with aggregate information,
# assuming the video has been processed
def get_df(video_i):
```

```
    dp = data_paths(video_i)
    if not os.path.exists(dp.video_df_path):
        print('File {} does not exist'.format(dp.video_df_path))
        return None
    return pd.read_csv(dp.video_df_path)

##############################

##############################
# IMAGE PROCESSING FUNCTIONS #
##############################

##############################

# extract x,y coordinates from a cv2 contour
def get_coordinates(contour):
    xs = [contour[i][0][0] for i in range(len(contour))]
    ys = [contour[i][0][1] for i in range(len(contour))]
    return xs,ys

# Given a video dataframe, categorize the aggregates
def add_category_col(df):
        # Identify early (spurious) aggregates
        early_p = np.unique(df[df.t < 100].particle.values)

        # Identify persistent FBs
        persist_p = []
        for p,df_slice in df.groupby('particle'):
            if p in early_p:
                continue
            avgCirc = np.mean(df_slice.circularity)
            if avgCirc > 0.5:
                persist_p.append(p)

        # Identify evaporators
```

```
        evap_p=[]
        for p,df_slice in df.groupby('particle'):
            if p in early_p:
                continue
            a = df_slice.area.values
            xmin = min(df_slice.x.values)
            xmax = max(df_slice.x.values)
            ymin = min(df_slice.y.values)
            ymax = max(df_slice.y.values)
            notNearEdges = (xmin > 50) and (xmax < 2542) and (ymin > 50) and (ymax < 1894)
            avgCirc = np.mean(df_slice.circularity)
            if (a[-1] < 0.75*max(a)) and notNearEdges and (avgCirc > 0.5) and (a[0] < max(a))
and (p not in persist_p):
                evap_p.append(p)

        # Add category column
        category_col = ['other']*len(df)
        for i,this_row in enumerate(df.iloc):
            if this_row.particle in early_p:
                category_col[i] = 'early'
            elif this_row.particle in persist_p:
                category_col[i] = 'persistent'
            elif this_row.particle in evap_p:
                category_col[i] = 'evaporates'
        df['category'] = category_col

        return df

def find_contours(img):
    # Denoise
    denoised = cv2.fastNlMeansDenoising(img, None, 70, 9, 25)

    ########################
    # A) Create hull mask #
    ########################
```

```
    # A1) Low-pass adaptive thresholding -> binary image of dark regions
    thresh_dark = 255 - cv2.adaptiveThreshold(denoised, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 121, 20)

    # A2) Identify contours, keeping those that aren't contained in a larger contour
    contours,h = cv2.findContours(thresh_dark,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    try:
        parent_info = np.array([i[-1] for i in h[0]])
    except TypeError:
        print('Could not process contour hierarchy')
        return None, None
    top_level_contours = [contours[i] for i in range(len(contours)) if parent_info[i] == -1]

    # A3) Remove small contours, to prep for morphological closing
    # Keep convex hulls of contours
    area_filtered_hulls = np.zeros(np.shape(img), np.uint8)
    thresh_dark_filtered = np.zeros(np.shape(img), np.uint8)
    for contour in top_level_contours:
        if cv2.contourArea(contour) > 1200: # Changed from 480 since previous algorithm
            convex_hull = cv2.convexHull(contour)
            area_filtered_hulls = cv2.drawContours(area_filtered_hulls, [convex_hull], -1,
255, -1)
            thresh_dark_filtered = cv2.drawContours(thresh_dark_filtered, [contour], -1, 255,
-1)

    # A4) Morphological closing
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (21,21))
    closed = cv2.morphologyEx(area_filtered_hulls, cv2.MORPH_CLOSE, kernel)

    # A5) Find contours of closed regions,
    # keeping the convex hulls
    final_contours,h = cv2.findContours(closed,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
    hull_mask = np.zeros(np.shape(img), np.uint8)
    for contour in final_contours:
        convex_hull = cv2.convexHull(contour)
```

```
        hull_mask = cv2.drawContours(hull_mask, [convex_hull], -1, 255, -1)


    ######################
    # B) Find highlights #
    ######################


    # B1) High-pass adaptive thresholding -> binary image of light regions
    thresh_light = 255 - cv2.adaptiveThreshold(255-denoised, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 31, 13)


    # B2) Keep only highlights that are inside aggregates by
    # filtering the high-pass binary image with the hull mask
    interior_highlights = cv2.bitwise_and(hull_mask, thresh_light)


    # B3) Dilate the interior highlights
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (17,17))
    interior_highlights = cv2.morphologyEx(interior_highlights, cv2.MORPH_DILATE, kernel)


    ########################################
    # C) Combine highlights and dark regions #
    ########################################


    # C1) Combine interior highlights with low-pass binary image
    comb = cv2.bitwise_or(interior_highlights, thresh_dark_filtered)


    # C2) Morphologically close the combined aggregates
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15,15))
    agg_mask = cv2.morphologyEx(comb, cv2.MORPH_CLOSE, kernel)


    # C3) Remove small aggregates using the hull mask
    agg_mask = cv2.bitwise_and(agg_mask, hull_mask)


    ####################################
    # D) Obtain final aggregate contours #
    ####################################
```

```
    all_contours,h = cv2.findContours(agg_mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    if len(all_contours) == 0:
        # Error handling if no contours were found
        return [], thresh_dark
    else:
        parent_info = np.array([i[-1] for i in h[0]])
        # Keep contours that aren't contained in another contour (like dark spots)
        contours = [all_contours[i] for i in range(len(all_contours)) if parent_info[i] == -1]
        return contours, thresh_dark

def find_contours_dark_only(img, denoising_strength=70, kernel_size=5, threshold_offset=20):
    # Nonlinear means denoising (blur that maintains edges)
    denoised = cv2.fastNlMeansDenoising(img, None, denoising_strength, 7, 21)

    # Adaptive thresholding (makes binary image)\
thresh1=cv2.adaptiveThreshold(denoised,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,101,th
reshold_offset)

    # Morphological opening (denoises binary image)
    kernel =cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(kernel_size,kernel_size))
    opened=cv2.morphologyEx(255-thresh1,cv2.MORPH_OPEN,kernel)

    # Find contours
    contours,h = cv2.findContours(opened,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    # Minimum area filter (ADDED TO MATCH NEW ALGORITHM)
    contours = [contour for contour in contours if cv2.contourArea(contour) > 1200]

    return contours

def getCenterOfMassCoords(contour, axis):
    return np.mean([contour[i][0][axis] for i in range(len(contour))])

def getAvgGray_dark_only(img, contours, contour_i):
    mask = np.zeros(img.shape, np.uint8)
```

```
    cv2.drawContours(mask, contours, contour_i, 255, -1)
    return cv2.mean(img, mask)[0]


def getAvgGray(img, thresh_dark, contours, contour_i):
    mask = np.zeros(img.shape, np.uint8)
    mask = cv2.drawContours(mask, contours, contour_i, 255, -1)
    mask = cv2.bitwise_and(mask, thresh_dark) # Only count gray value of pixels that aren't
included in a highlight
    return cv2.mean(img, mask)[0]


# Process the frame at time t (minutes from inoculation)
# of the video with registry index number video_i
#
# Saves the contour points and returns a dataframe for the processed image
#
# If dark_only = True, this frame will be processed with the
# old aggregate finding algorithm that assumes aggregates do not have
# bright highlights
def process_frame(video_i, t, dark_only=False):
    # Prep output locations
    dp = data_paths(video_i)

    # Find image file for the chosen video and timepoint
    img = get_img(video_i, t)
    if img is None:
        print('Video {} has no image at time {}'.format(video_i, t))
        return

    # Process image to find aggregate contours
    # and regions where aggregates are dark
    if dark_only:
        contours = find_contours_dark_only(img)
    else:
        contours,thresh_dark = find_contours(img)
    if contours is None:
```

```
        print('Contour hierarchy could not be processed for video {} at time t={}
mins.'.format(video_i, t))
        return pd.DataFrame(columns=['t', 'x', 'y', 'area', 'perim', 'circularity'])


    # Extract data from contours
    all_ts = [t]*len(contours)
    all_cxs = [getCenterOfMassCoords(contour, 0) for contour in contours]
    all_cys = [getCenterOfMassCoords(contour, 1) for contour in contours]
    all_areas = [cv2.contourArea(contour) for contour in contours]
    all_perims = [cv2.arcLength(contour, True) for contour in contours]
    if dark_only:
        all_avggray = [getAvgGray_dark_only(img, contours, contour_i) for contour_i in
range(len(contours))]
    else:
        all_avggray = [getAvgGray(img, thresh_dark, contours, contour_i) for contour_i in
range(len(contours))]


    # Save contour points
    np.save(dp.contour_points_path.format(t), np.array(contours, dtype=object))


    # Make a dataframe for output
    frame_df = pd.DataFrame({
            't':all_ts,
            'x':all_cxs,
            'y':all_cys,
            'area':all_areas,
            'perim':all_perims,
            'grayValue':all_avggray
            })
    # Remove small aggregates
    frame_df = frame_df[frame_df.area > 480]
    # Calculate circularity column
    frame_df['circularity'] = 4*np.pi*frame_df.area.values / (frame_df.perim.values)**2
    # Add a column for contour number
    frame_df['contour'] = frame_df.index.values
```

```
    return frame_df

# Process an entire video
#
# If dark_only = True, this video will be processed with the
# old aggregate finding algorithm that assumes aggregates do not have
# bright highlights
def process_video(video_i, dark_only=False, t_range=None):
    # Default t_range if not specified
    if t_range is None:
        t_range = np.arange(0,1440+10,10)

    # Process frames in parallel (also saves contour points)
    worker_pool = mp.Pool()
    frame_dfs = worker_pool.starmap(process_frame, [(video_i,t,dark_only) for t in t_range])
    frame_dfs = [f for f in frame_dfs if f is not None]
    video_df = pd.concat(frame_dfs)
    video_df.index = np.arange(len(video_df))

    # Add tracking information
    tp.quiet() # Suppress unnecessary messages
    tracked_df = tp.link_df(video_df, 30, memory=70, t_column='t')

    # TODO: Add evaporator/persistor categories (may require handling mergers/separations?)

    # Save dataframe
    dp = data_paths(video_i)
    tracked_df.to_csv(dp.video_df_path, float_format='%.3f', index=False)
    print('Saved video dataframe to ' + dp.video_df_path)

# Reprocess selected frames of a video
#
# If dark_only = True, these frames will be processed with the
# old aggregate finding algorithm that assumes aggregates do not have
# bright highlights
```

```
def reprocess_video(video_i, dark_only=False, t_range=None):
    # Default t_range if not specified
    if t_range is None:
        t_range = np.arange(0,1440+10,10)

    # Load previous dataframe
    video_df = get_df(video_i)

    # Process frames in parallel (also saves contour points),
    # updating only the affected rows of the video dataframe
    worker_pool = mp.Pool()
    frame_dfs = worker_pool.starmap(process_frame, [(video_i,t,dark_only) for t in t_range])
    # Trim out non-existent frames
    processed_frames_i = [i for i in range(len(frame_dfs)) if frame_dfs[i] is not None]
    frame_dfs = np.array(frame_dfs)[processed_frames_i]
    t_range = np.array(t_range)[processed_frames_i]
    for i in range(len(t_range)):
        video_df = video_df.drop(index=video_df[video_df.t == t_range[i]].index.values)
        video_df = pd.concat((video_df, frame_dfs[i]))
        video_df.index = np.arange(len(video_df))
    video_df = video_df.sort_values('t')
    video_df.index = np.arange(len(video_df))

    # Add tracking information
    tp.quiet() # Suppress unnecessary messages
    tracked_df = tp.link_df(video_df, 30, memory=70, t_column='t')

    # TODO: Add evaporator/persistor categories (may require handling mergers/separations?)

    # Save dataframe
    dp = data_paths(video_i)
    tracked_df.to_csv(dp.video_df_path, float_format='%.3f', index=False)
    print('Updated video dataframe ' + dp.video_df_path)

######################
```

```
######################
# ANALYSIS FUNCTIONS #
######################

######################

def accept_video_i(func):
        # If the user specifies a video index,
        # load the video df
        def wrapper(identifier, *args, **kwargs):
            if isinstance(identifier, int) or isinstance(identifier, np.int64):
                df = get_df(identifier)
                return func(df, *args, **kwargs)
            else:
                # otherwise assume the df was passed
                return func(identifier, *args, **kwargs)
        return wrapper

########################
# From video dataframe #
########################

@accept_video_i
def get_start_time(df, n_threshold=10, area_threshold=800):
    # Return time when the number of fruiting bodies
    # above a certain size reaches a certain threshold
    def n_are_large(areas, threshold):
        return sum(np.array(areas) > threshold)

    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    n_large = df_slice.groupby('t')['area'].agg(n_are_large, area_threshold)
```

```
    times_when_n_large_exceeds = n_large.index[n_large >= n_threshold]
    if len(times_when_n_large_exceeds) == 0:
        return 0
    return min(times_when_n_large_exceeds)


@accept_video_i
def get_peak_time(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]


    total_areas = df_slice.groupby('t')['area'].agg(np.sum)
    if len(total_areas) == 0:
        return 0
    times_at_max = total_areas.index[total_areas == max(total_areas)]
    return min(times_at_max)


@accept_video_i
def get_stability_time(df, window=31, threshold=0.5):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]


    n = df_slice.groupby('t')['area'].agg(len)
    if len(n) < window:
        return 2000
    n_deriv = savgol_filter(n.values, window, 3, deriv=1)
    all_stable_after_i = np.where([np.all(abs(n_deriv[i:]) <= threshold) for i in
range(len(n_deriv))])[0]
    if len(all_stable_after_i) == 0:
        return 2000 # A hypothetical eventual stability time?
    else:
        min_stable_after_i = min(all_stable_after_i)
        return n.index.values[min_stable_after_i]
```

```
@accept_video_i
def get_peaktime_area_mean(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    peak_time = get_peak_time(df)
    return np.mean(df_slice[df_slice.t == peak_time].area.values)

@accept_video_i
def get_peaktime_area_std(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    peak_time = get_peak_time(df)
    return np.std(df_slice[df_slice.t == peak_time].area.values)

@accept_video_i
def get_final_area_mean(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    mean_areas = df_slice.groupby('t')['area'].agg(np.mean)
    return mean_areas.values[-1]

@accept_video_i
def get_final_area_std(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    std_areas = df_slice.groupby('t')['area'].agg(np.std)
    return std_areas.values[-1]
```

```
@accept_video_i
def get_peaktime_n(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]
    if len(df_slice) == 0:
        return 0

    peak_time = get_peak_time(df)
    return len(df_slice[df_slice.t == peak_time])

@accept_video_i
def get_final_n(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]
    if len(df_slice) == 0:
        return 0

    return len(df_slice[df_slice.t == max(df_slice.t.values)])

@accept_video_i
def get_frac_lost(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]
    if len(df_slice) < 4:
        return 0

    ns = df_slice.groupby('t')['area'].agg(len) # count could use any column
    n_sort = np.sort(ns.values) # sort ascending
    n_max = np.mean(n_sort[-3:]) # smooth max n with three largest values
    n_final = ns.values[-1]
```

```
    if n_final >= n_max:
        return 0
    else:
        return (n_max - n_final)/n_max

@accept_video_i
def get_final_circ_mean(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    mean_circs = df_slice.groupby('t')['circularity'].agg(np.mean)
    return mean_circs.values[-1]

@accept_video_i
def get_final_circ_std(df):
    # Ignore aggregates that formed at 100 minutes after inoculation or earlier
    early_p = np.unique(df[df.t < 100].particle.values)
    df_slice = df[[p not in early_p for p in df.particle]]

    std_circs = df_slice.groupby('t')['circularity'].agg(np.std)
    return std_circs.values[-1]

########################
# Metrics from images #
########################

def get_final_comp_size(i):
    dp = data_paths(0)
    cs_df = pd.read_csv(dp.img_metrics_compSize, index_col=0)
    return np.mean(cs_df.loc[i].comp_size.values[-10])

def get_max_comp_size(i):
    dp = data_paths(0)
    cs_df = pd.read_csv(dp.img_metrics_compSize, index_col=0)
```

```
    return max(cs_df.loc[i].comp_size.values)

def get_t_max_comp_size(i):
    dp = data_paths(0)
    cs_df = pd.read_csv(dp.img_metrics_compSize, index_col=0)
    this_df = cs_df.loc[i]
    max_slice = this_df[this_df.comp_size == max(this_df.comp_size)]
    if len(max_slice) == 0:
        return 0
    else:
        return min(max_slice.t.values)

def get_final_std_grayValue(i):
    dp = data_paths(0)
    sg_df = pd.read_csv(dp.img_metrics_std_grayValue, index_col=0)
    return np.mean(sg_df.loc[i].std_grayValue.values[-10])

def get_max_std_grayValue(i):
    dp = data_paths(0)
    sg_df = pd.read_csv(dp.img_metrics_std_grayValue, index_col=0)
    return max(sg_df.loc[i].std_grayValue.values)

######################

######################
# PLOTTING FUNCTIONS #
######################

######################

# Plot many images together as a mosaic,
# specifying the video index to show
# and the time to show.
# Optionally, the video index and time can be
# displayed on each image
```

```
def frame_mosaic(video_indices, ts=None, show_i=True, show_t=False, dpi=75):
    ncols = 6
    nrows = int(np.ceil(len(video_indices)/ncols))
    height,width = np.shape(get_img(0,0))
    fig,all_ax = plt.subplots(nrows,ncols,figsize=(20,20*height/width*nrows/ncols*1.025),
dpi=dpi)
    fig.subplots_adjust(hspace=0, wspace=0)

    # Default time
    if ts is None:
        ts = [1440]*len(video_indices)
    # Allow user to pick one timepoint for all images
    if isinstance(ts, int):
        ts = [ts]*len(video_indices)

    for i,ax in enumerate(all_ax.flatten()):
        if i < len(video_indices):
            img = get_img(video_indices[i], ts[i])
            ax.imshow(img, cmap='gray')
            label = ''
            if show_i:
                label = str(video_indices[i])
            if show_t:
                label += ', {} min'.format(ts[i])
            ax.annotate(label, (0,250), color='white')
        ax.axis('off')

# Helper function for scatterplots:
# Plot the polygon that bounds the x,y points
# to emphasize the spread, especially alongside
# other scatterplots
def plotHull(xs, ys, ax, *args, **kwargs):
    if args == ():
        args = 'o-' # default plot marker
    points = np.column_stack((xs, ys))
```

```
    if len(points) > 2:
        hull = ConvexHull(points)
        vertices_i = hull.vertices.copy()
        vertices_i = np.append(vertices_i, hull.vertices[0])
        ax.plot(np.array(xs)[vertices_i], np.array(ys)[vertices_i], *args, **kwargs)
    else:
        ax.plot(xs, ys, *args, **kwargs)

def get_points_inside(c, xs, ys):
    points_data = np.column_stack((xs,ys))
    all_points_inside_i = []
    for clevel in c.allsegs:
        n_points = 0
        for segment in clevel:
            cpath = Path(segment)
            points_inside_i = np.arange(len(xs))[cpath.contains_points(points_data)]
            all_points_inside_i.extend(points_inside_i)
    return np.array(xs)[all_points_inside_i], np.array(ys)[all_points_inside_i]

# Helper function for finding contour
# that contains a given fraction of
# scatterplot points
def get_frac_points_inside(c, xs, ys):
    points_data = np.column_stack((xs,ys))
    frac_points = []
    for clevel in c.allsegs:
        n_points = 0
        for segment in clevel:
            cpath = Path(segment)
            points_inside = sum(cpath.contains_points(points_data))
            n_points += points_inside
        frac_points.append(n_points / len(points_data))
    return frac_points

# Helper function for finding contour
```

```
# that contains a given fraction of
# scatterplot points
def incl_cost(ax, cloud, xs, ys, q, frac):
    c = ax.contour(cloud.T, [np.quantile(cloud.flatten(), q)], extent=(-10, 10, -10, 10))
    points_data = np.column_stack((xs,ys))
    for clevel in c.allsegs:
        n_points = 0
        for segment in clevel:
            cpath = Path(segment)
            points_inside = sum(cpath.contains_points(points_data))
            n_points += points_inside
    return (n_points/len(points_data) - frac)**2

# Gaussian kernel density of 2D scatterplot
def densityMap(xs, ys, extent=(-10,10), nBins=500, kernelWidth=101):
    heatmap, xedges, yedges = np.histogram2d(xs, ys, bins=np.linspace(*extent, nBins))
    density = cv2.GaussianBlur(heatmap, (kernelWidth, kernelWidth), 0)
    return density

# Returns xs and ys of desired contour.
# NOTE: There may be multiple polygons, so plot this as
#
# polys = get_frac_contour(ax, xs, ys, frac)
# for poly in polys:
#     plot(*poly)
def get_frac_contour(ax, xs, ys, frac, extent=(-10,10), nBins=500, kernelWidth=101,
min_q=None, **kwargs):
    # Create density map of scatterplot points
    cloud = densityMap(xs, ys, extent, nBins, kernelWidth)

    # Solve for contour
    if min_q is None:
        min_q = sum(cloud.flatten() > 0)/len(cloud.flatten()) + 0.01
    sol = minimize_scalar(lambda q: incl_cost(ax, cloud, xs, ys, q, frac), bounds=(min_q, 1),
method='bounded')
```

```
    # Extract contour
    c = ax.contour(cloud.T, [np.quantile(cloud.flatten(), sol.x)], extent=(*extent, *extent))

    # Extract coordinates
    polys = [(cdata.T[0], cdata.T[1]) for cdata in c.allsegs[0]]
    return polys

########################

########################
# HUB ACCESS FUNCTIONS #
########################

########################

def get_remote_scope_folders(sftp, remote_run_folder):
    scope_folders = []
    for entry in sftp.listdir_attr(remote_run_folder):
        mode = entry.st_mode
        if S_ISDIR(mode):
            if entry.filename.startswith('Scope'):
                scope_folders.append(entry.filename)
    return scope_folders

def get_t(filename):
    t = int(re.findall('_(\\d+).tif$', filename)[0]) - 1
    return t

def cpMovieFromHub(remote_scope_folder, local_dst_folder, remote_server='as-welchlab-
nat.syr.edu', username='', password=''):
    # Open SSH connection
    ssh = paramiko.SSHClient()
    ssh.load_host_keys(os.path.expanduser('~/.ssh/known_hosts'))
    ssh.connect(remote_server, username=username, password=password, allow_agent=False)
    sftp = ssh.open_sftp()
```

```
    # Get all image filenames in this remote scope folder
    img_filenames = sftp.listdir(remote_scope_folder)

    # If there are less than 100 images, warn the user
    if len(img_filenames) < 100:
        print('There are less than 100 images in {}'.format(remote_scope_folder))

    # Get subset of images with a 10 minute increment
    imgs_to_copy = [os.path.join(remote_scope_folder, f) for f in img_filenames if get_t(f)%10
== 0]

    # Copy those images to the destination folder
    for img_path in imgs_to_copy:
        img_filename = os.path.basename(img_path)
        sftp.get(img_path, os.path.join(local_dst_folder, img_filename))

    # Close SSH connection
    sftp.close()
    ssh.close()


def findRunFolder(sftp, run_num):
    dp = data_paths()
    run_folder = None
    for search_path in dp.hub_run_folder_locations:
        try:
            for entry in sftp.listdir_attr(search_path):
                mode = entry.st_mode
                if S_ISDIR(mode):
                    if 'Run{:04d}'.format(run_num) in entry.filename:
                        run_folder = os.path.join(search_path, entry.filename)
        except:
            continue
    if run_folder is None:
        print('No folder found for run {}'.format(run_num))
```

```
    return run_folder

# Use this function to copy every tenth image from all videos in a given run
# Note that it returns a dataframe that should be manually validated
# and then added to the video registry with add_rows_to_registry()
def cpRunFromHub(run_num, local_dst_folder=None):
    dp = data_paths()
    # Default local destination directory
    if local_dst_folder is None:
        local_dst_folder = dp.new_img_parent

    # Get network hub info for remote connection
    remote_server = dp.hub_ip
    username = dp.hub_user
    password = dp.hub_pw

    # Open SSH connection
    ssh = paramiko.SSHClient()
    ssh.load_host_keys(os.path.expanduser('~/.ssh/known_hosts'))
    ssh.connect(remote_server, username=username, password=password, allow_agent=False)
    sftp = ssh.open_sftp()

    # Find images on the network hub
    remote_run_folder = findRunFolder(sftp, run_num)
    if remote_run_folder is None:
        return
    print('Copying videos from Run {}: {}'.format(run_num, remote_run_folder))
    scope_folders = get_remote_scope_folders(sftp, remote_run_folder)

    # Prep run folder in local destination
    new_run_folder = os.path.join(local_dst_folder, os.path.basename(remote_run_folder))
    if not os.path.exists(new_run_folder):
        os.mkdir(new_run_folder)

    # Get date when run was started
```

```
    m_time = sftp.stat(os.path.join(remote_run_folder, 'scopeInfo.txt')).st_mtime

    timestamp = datetime.fromtimestamp(m_time)

    date = "{dt.month}/{dt.day}/{dt.year}".format(dt = timestamp)


    # Prepare new rows to be added to the registry

    sftp.get(os.path.join(remote_run_folder, 'scopeInfo.txt'), os.path.join(new_run_folder,
'scopeInfo.txt'))

    scopeInfo = pd.read_csv(os.path.join(new_run_folder, 'scopeInfo.txt'), sep='\t')

    new_rows = pd.DataFrame({'run':run_num, 'scope':scopeInfo.scope,
'strain':scopeInfo.strain})

    new_rows['condition'] = '1%_TPM_agar'

    new_rows['date'] = date

    new_rows['error'] = 0

    new_rows['processed'] = False

    new_rows['img_folder'] = '' # to be populated


    # Copy images from the network hub with a time increment of 10 minutes

    for i,scope_folder in enumerate(scope_folders):

        print('Copying from folder {} of {}: {}'.format(i+1, len(scope_folders),
scope_folder), flush=True, end='\r')


        # Get all image filenames in this remote scope folder

        img_filenames = sftp.listdir(os.path.join(remote_run_folder, scope_folder))


        # If there are less than 100 images, skip this folder

        if len(img_filenames) < 100:

            continue


        # Prep scope folder in local destination

        new_scope_folder = os.path.join(new_run_folder, scope_folder)

        if not os.path.exists(new_scope_folder):

            os.mkdir(new_scope_folder)


        # Get subset of images with a 10 minute increment

        imgs_to_copy = [os.path.join(remote_run_folder, scope_folder, f) for f in
img_filenames if get_t(f)%10 == 0]
```

```
    # Copy those images to the destination folder
    for img_path in imgs_to_copy:
        img_filename = os.path.basename(img_path)
        sftp.get(img_path, os.path.join(new_scope_folder, img_filename))

    # Include local destination in rows to be added to registry
    scope_num = int(scope_folder[5:]) # Scope folder named as 'Scope44'
    new_row_i = new_rows[new_rows.scope == scope_num].index.values[0]
    new_rows.loc[i, 'img_folder'] = new_scope_folder

# Close SSH connection
sftp.close()
ssh.close()

# Return rows to be added to registry
return new_rows
```

Tools for PCA analysis

```
"""
# Calculate a new PCA from scratch:
#       run PCA_tools
#       sample = registry[...conditions...]
#       p = PCA_tools().new(sample) # Loads default metrics and scales them unsupervised
#       p.computePCA()

# Save data:
#       p.save() # Prompts user for savepath
#               or
#       p.save('/user/home/myxo-tracking/PCA/tmp/')

# Load previously calculated data:
#       data = PCA_tools().load() # Prompts user for loadpath
#               or
#       data = PCA_tools().load('/user/home/3d-scope-agg-tracking/PCA/PCA_all/')
#
# Loading data also makes plotting functions available

# If computing new default metrics:
#       p = PCA_tools()
#       p.new(filter_errors=False)
#       p.computeMetrics()
#
#       then place metrics_df.csv in the default parentPath
"""

import numpy as np
import agg_tools as tools
import pandas as pd
import os
from datetime import datetime
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib import colors
from matplotlib.patches import Patch
import cv2
from scipy.optimize import minimize_scalar
from matplotlib.path import Path

registry = tools.load_registry() # for convenience when using `run PCA_tools`

class PCA_tools:
	def __init__(self):
		# Filenames for all ouput
		# Keys are properties of the class
		self.output_filenames = {
			'registry_sample':'included_videos_unfiltered.csv',
			'filtered_registry_sample':'included_videos_filtered.csv',
			'metrics_df':'metrics_df.csv',
			'scaled_metrics_df':'scaled_metrics_df.csv',
			'principalDf':'PCA.csv',
			'pc_variance_ratios':'PC_variance.txt',
			'pc_metrics':'PC_metrics.txt',
			'nan_indices':'nan_indices.txt',
			'metrics_datetime':'when_metrics_calculated.txt'
			}

		self.metric_functions = {
			'start_time':tools.get_start_time,
			'peak_time':tools.get_peak_time,
			'stability_time':tools.get_stability_time,
			'peaktime_area_mean':tools.get_peaktime_area_mean,
			'peaktime_area_std':tools.get_peaktime_area_std,
			'final_area_mean':tools.get_final_area_mean,
			'final_area_std':tools.get_final_area_std,
```

```
                'peaktime_n':tools.get_peaktime_n,
                'final_n':tools.get_final_n,
                'frac_lost':tools.get_frac_lost
                }

        # Initialize
        self.metric_names = list(self.metric_functions.keys())

        self.parentPath = self.ensureFinalSlash(tools.data_paths(0).pca_folder) # Directory for output folders, should also contain metrics_df.csv for default pre-calculated metrics
        self.savepath = None

        self.registry_sample = None
        self.filtered_registry_sample = None
        self.metrics_df = None
        self.scaled_metrics_df = None
        self.principalDf = None
        self.nan_indices = None
        self.n_components = None
        self.metrics_datetime = None

        self.registry = tools.load_registry()

    # Prepares scaled metrics from registry sample
    # ready to run PCA
    def new(self, registry_sample=None, filter_errors=True, parentPath=None):
        # Default to using entire registry
        if registry_sample is None:
            registry_sample = self.registry

        # Clean up registry sample
        self.registry_sample, self.filtered_registry_sample = self.prepRegistrySample(registry_sample, filter_errors)

        # Load default metrics
```

```
                self.metrics_df = self.loadDefaultMetrics(parentPath)
                if self.metrics_df is None:
                        # Check if some default metrics are missing
                        return self

                # Scale metrics (unsupervised)
                self.scaleMetrics() # Populates self.scaled_metrics_df and self.nan_indices

                return self

        def save(self, savepath=None):
                # Prompt user for savepath if one is not given or previously specified
                if savepath is None:
                        if self.savepath is None:
                                savepath = self.makeNewDir(self.parentPath)
                        else:
                                savepath = self.savepath

                # Prompt user if anything might be overwritten
                proceed, savepath = self.checkSaveConflicts(savepath)
                if not proceed:
                        return

                # Save each output file
                self.registry_sample.to_csv(savepath + self.output_filenames['registry_sample'])
                self.filtered_registry_sample.to_csv(savepath +
self.output_filenames['filtered_registry_sample'])
                self.appendColumns(self.metrics_df).to_csv(savepath +
self.output_filenames['metrics_df'])
                self.appendColumns(self.scaled_metrics_df).to_csv(savepath +
self.output_filenames['scaled_metrics_df'])
                self.principalDf.to_csv(savepath + self.output_filenames['principalDf'])
                np.savetxt(savepath + self.output_filenames['pc_variance_ratios'],
self.pc_variance_ratios)
                np.savetxt(savepath + self.output_filenames['pc_metrics'], self.pc_metrics)
                np.savetxt(savepath + self.output_filenames['nan_indices'], self.nan_indices,
fmt='%d')
```

```
                with open(savepath + self.output_filenames['metrics_datetime'], 'w') as f:
                        f.write(self.metrics_datetime)

                print('Output data saved to ' + savepath)
                self.savepath = savepath

        # Loads data saved at loadpath
        def load(self, loadpath=None):
                # Defaults to user prompt for loadpath
                if loadpath is None:
                        loadpath = self.pickLoadpath(self.parentPath)

                # Check for each file in self.output_filenames
                for variable_name in self.output_filenames.keys():
                        output_filename = self.output_filenames[variable_name]
                        if os.path.exists(loadpath + output_filename):
                                # Load in dataframes
                                if output_filename.endswith('.csv'):
                                        df = pd.read_csv(loadpath + output_filename, index_col=0)
                                        if variable_name != 'principalDf':
                                                # Remove non-numeric columns that help human
readability
                                                df = df.drop('strain', axis=1)
                                        # Store data in memory
                                        setattr(self, variable_name, df)
                                # Load in metrics_datetime
                                elif variable_name == 'metrics_datetime':
                                        with open(loadpath + output_filename, 'r') as f:
                                                self.metrics_datetime = f.readline()[:-1]
                                # Load in numeric txt files
                                else:
                                        if os.stat(loadpath).st_size != 0: # suppress an annoying
warning about an empty file
                                                data = np.loadtxt(loadpath + output_filename)
                                        else:
                                                data = []
```

```
                                # Store data in memory
                                setattr(self, variable_name, data)
                        else:
                                print(loadpath + output_filename + ' missing')


                # Load the number of primary components
                self.n_components = sum(['pc' in colName for colName in
self.principalDf.columns])
                return self


        ##################
        ##################
        ###            ###
        ###  PCA CORE ###
        ###            ###
        ##################
        ##################


        def computeMetric(self, video_i):
                # Calculate all metric values for this video
                metric_vector = [self.metric_functions[metric_name](video_i) for metric_name in
self.metric_names]
                return metric_vector


        def computeMetrics(self, video_indices=None):
                # Default to indices in the filtered_registry_sample
                if video_indices is None:
                        video_indices = self.filtered_registry_sample.index


                # Calculate and store metrics for each video
                metric_vectors = []
                for i in video_indices:
                        # Indicate which metric is being calculated (in case of warnings/errors)
                        videoname = tools.data_paths(i).videoname
                        print('{}: {}'.format(i, videoname))
```

```
                        # Calculate all metric values for this video
                        metric_vector = self.computeMetric(i)

                        # Store metric values for this video
                        metric_vectors.append(metric_vector)

                # Keep date and time for when metrics were calculated
                self.metrics_datetime = datetime.now().strftime("%Y-%m-%d, %H:%M:%S")

                # Compile all metric values into a dataframe by video index
                self.metrics_df = pd.DataFrame(data=metric_vectors, columns=self.metric_names)
                self.metrics_df.index = video_indices

                return self.metrics_df

        # Filter out metrics that are nan
        # Scale remaining metrics unsupervised,
        #       i.e. mean -> 0, variance -> 1 for the given sample of metrics
        # Returns:
        # - dataframe of scaled metrics
        # - list of video indices with nan metrics
        def scaleMetrics(self, metrics_df=None):
                # Default to self.metrics_df
                if metrics_df is None:
                        metrics_df = self.metrics_df

                # Remove vectors that have missing values
                metrics_df = self.filterNan(metrics_df, store_nan_indices=True)

                # Scale each metric to have mean 0 and variance 1
                scaled_metric_vectors = StandardScaler().fit_transform(metrics_df)

                # Compile all scaled metric values into a dataframe by video index
                self.scaled_metrics_df = pd.DataFrame(data=scaled_metric_vectors,
columns=self.metric_names)
```

```
                self.scaled_metrics_df.index = metrics_df.index

                return self.scaled_metrics_df, self.nan_indices

        def computePCA(self, input_data=None, n_components=6):
                self.n_components = n_components

                # Default to using the scaled metric vectors as input
                if input_data is None:
                        if self.scaled_metrics_df is None:
                                self.scaleMetrics()
                        input_data = self.scaled_metrics_df

                # Report on how many videos of each strain are included in the PCA
                print(self.filtered_registry_sample.strain.value_counts())

                # Perform PCA
                pca = PCA(n_components=n_components)
                pc = pca.fit_transform(input_data)
                pca_out = pca.fit(input_data)

                # Save output:
                # - principal component values for each video
                # - explained variance of each PC
                # - metric makeup of each PC
                self.principalDf = pd.DataFrame(data = pc, columns = ['pc{}'.format(i+1) for i
in range(n_components)])
                self.principalDf.index = input_data.index
                self.principalDf = self.appendColumns(self.principalDf)

                self.pc_variance_ratios = pca_out.explained_variance_ratio_
                self.pc_metrics = pca_out.components_
                #self.covariance = pca_out.get_covariance()

                # Report on explained variance
```

```
            print('Explained variance:')
            print(self.pc_variance_ratios)
            print('{}% of total variance explained by the top {} primary
components'.format(int(100*sum(self.pc_variance_ratios)), n_components))


        ############################
        ############################
        ###                      ###
        ###  PLOTTING FUNCTIONS ###
        ###                      ###
        ############################
        ############################


        # Scatterplot of all PCA points
        def plotPCA(self, x_axis='pc1', y_axis='pc2', data=None, fig=None, ax=None):
            # Make new plot window if one isn't supplied
            if (fig is None) or (ax is None):
                self.fig,self.ax = plt.subplots()
            else:
                self.fig = fig
                self.ax = ax

            # Use full principalDf if a slice isn't supplied
            if data is None:
                data = self.principalDf

            # Plot PCA output
            self.fig.canvas.manager.set_window_title("PCA Feature Space")
            self.ax.set_aspect('equal')
            for i in data.index:
                homolog_group = data.loc[i,'mutant']
                x,y = data.loc[i, x_axis], data.loc[i, y_axis]
                self.ax.scatter(x,y, color=self.h_colors[homolog_group])

            self.fig.tight_layout()
```

```
		plt.show()


	def densityMap(self, xs, ys, extent=(-6.5,6.5), nBins=20, kernelWidth=5):
		heatmap, xedges, yedges = np.histogram2d(xs, ys, bins=np.linspace(*extent, nBins))
		density = cv2.GaussianBlur(heatmap, (kernelWidth, kernelWidth), 0)
		return density


	def plotDensity(self, mutant=None, scatter=True, scattersize=2, x_axis='pc1', y_axis='pc2', data=None, fadeExp=2, darkenFactor=0.75, scatter_alpha=0.6, extent=(-6.5, 6.5), nBins=20, kernelWidth=5, colorbar=True, fig=None, ax=None):
		# Make new plot window if one isn't supplied
		if (fig is None) or (ax is None):
			self.fig,self.ax = plt.subplots()
		else:
			self.fig = fig
			self.ax = ax

		# Use full principalDf if a slice isn't supplied
		if data is None:
			data = self.principalDf

		# Filter by mutant if supplied
		if mutant is not None:
			data = data[data.mutant == mutant]
			scatter_color = self.darken(self.h_colors[mutant], darkenFactor)
			fade_map = self.fade(self.h_colors[mutant], fadeExp)
		else:
			scatter_color = self.darken('C0', darkenFactor)
			fade_map = self.fade('C0', fadeExp)

		# Extract data
		xs,ys = data[x_axis], data[y_axis]

		# Make scatterplot, if desired
		if scatter:
```

```
                        self.ax.scatter(xs, ys, scattersize, color=scatter_color,
alpha=scatter_alpha)


                # Plot density

                self.ax.set_xlim(*extent)

                self.ax.set_ylim(*extent)

                im = self.ax.imshow(self.densityMap(xs, ys, extent=extent, nBins=nBins,
kernelWidth=kernelWidth).T, extent=(*extent,*extent), origin='lower', cmap=fade_map)

                if colorbar:

                        self.fig.colorbar(im)


        def plotPCmakeup(self, top_n_components=4, top_n_metrics=10, width_ratios=[1,5],
fig=None, ax=None):

                # Make new plot window if one isn't supplied

                if (fig is None) or (ax is None):

                        self.fig,self.ax = plt.subplots(1,2, figsize=(8,4),
gridspec_kw={'width_ratios': width_ratios})

                else:

                        self.fig = fig

                        self.ax = ax


                # Calculate PCA if it has not been done yet

                if self.principalDf is None:

                        self.computePCA()


                ############################

                # Explained variance plot #

                ############################

                # Colors for explained variance bars

                bar_colors = [plt.cm.Blues(i) for i in np.linspace(0.25, 1, self.n_components)]


                total_explained_variance = sum(self.pc_variance_ratios)

                # Bar for "other"

                self.ax[0].bar(0, 1-total_explained_variance, bottom=total_explained_variance,
fill=None, edgecolor='lightgray', hatch='//')

                # Stacked bars the explained variance of each PC

                for i in range(self.n_components):
```

```
                    self.ax[0].bar(0, self.pc_variance_ratios[i],
bottom=sum(self.pc_variance_ratios[:i]), color=bar_colors[i], edgecolor='black')


                # Add text to label the variance bars
                i=0
                for bar in self.ax[0].patches:
                        if i > 0:
                                self.ax[0].text(
                                        # Center text
                                        bar.get_x() + bar.get_width() / 2,
                                        # Offset text vertically
                                        bar.get_height() + bar.get_y()-0.055,
                                        'PC{}'.format(i),
                                        ha='center',
                                        color='w',
                                        weight='bold',
                                        size=12
                                )
                        i += 1


                # Style axes
                self.ax[0].axes.xaxis.set_visible(False)
                self.ax[0].set_xlim(-1,1)
                self.ax[0].set_yticks(np.arange(0, 1.1, 0.1))
                self.ax[0].set_ylabel('Cumulative variance')
                self.ax[0].set_title('Variance explained')


                ########################################
                # Primary component breakdown by metric #
                ########################################
                # Choose 18 distinct colors, with the 10 darkest colors the top metrics of PC1,
                # and the other 8 pastel
                init_colors = np.concatenate((plt.cm.tab10.colors, plt.cm.Pastel2.colors))[:18]
                pc1_sort = np.argsort(np.abs(self.pc_metrics[0]))[::-1]
                metric_colors = np.array([None]*18)
```

```
                for i in range(18):
                        metric_colors[pc1_sort[i]] = init_colors[i]


                for i in range(top_n_components):
                        pci_sort = np.argsort(np.abs(self.pc_metrics[i]))[::-1] # Sort
descending
                        pci_sort_comp = self.pc_metrics[i][pci_sort]
                        pci_sort_names = self.metric_names[pci_sort]
                        bar_norm = sum(np.abs(pci_sort_comp))
                        self.ax[1].bar('PC{}'.format(i+1),
sum(np.abs(pci_sort_comp)[top_n_metrics:])/bar_norm, fill=None, edgecolor='lightgray',
hatch='//')
                        for j in range(top_n_metrics):
                                bar,= self.ax[1].bar('PC{}'.format(i+1),
np.abs(pci_sort_comp[j])/bar_norm, bottom=sum(np.abs(pci_sort_comp)[(j+1):])/bar_norm,
color=metric_colors[pci_sort][j], edgecolor='black')
                                if pci_sort_comp[j] < 0:
                                        self.ax[1].scatter(bar.get_x()+bar.get_width()/8,
bar.get_y()+bar.get_height()/2, color='black', zorder=3)
                self.ax[1].set_ylim(0, 1.1)
                self.ax[1].axes.yaxis.set_visible(False)


                # Shrink current axis by 20%
                box = self.ax[1].get_position()
                self.ax[1].set_position([box.x0, box.y0, box.width * 0.8, box.height])


                # Put a legend to the right of the current axis
                legend_elements = [Patch(facecolor=metric_colors[pc1_sort][i],
edgecolor='black', label=self.metric_names[pc1_sort][i]) for i in range(18)]
                self.ax[1].legend(loc='center left', bbox_to_anchor=(1, 0.5),
handles=legend_elements)
                self.ax[1].set_title('Top metrics of Principal Components\n(A dot indicates the
negative direction)')


        # Graphical helper functions
        def fade(self, color, exponent=1):
                if isinstance(color, str):
                        color = colors.to_rgba(color)
                if len(color) == 3: # no alpha in specified color
```

```
                newColors = [(*color, a) for a in np.linspace(0,1,256)**exponent]
        elif len(color) == 4:
                r,g,b,a = color
                newColors = [(r,g,b,alpha) for alpha in np.linspace(0,1,256)**exponent]
        return colors.ListedColormap(newColors)

    def darken(self, color, factor=0.5):
        if isinstance(color, str):
                color = colors.to_rgba(color)
        if len(color) == 3: # no alpha in specified color
                r,g,b = color
                a = 1
        elif len(color) == 4:
                r,g,b,a = color
        return (r*factor, g*factor, b*factor, a)

    ############################
    ############################
    ###                      ###
    ###  HELPER FUNCTIONS    ###
    ###                      ###
    ############################
    ############################

    def loadDefaultMetrics(self, parentPath=None):
        # Prepare directory that contains precalculated metric vectors
        if parentPath is None:
                parentPath = self.parentPath
        parentPath = self.ensureFinalSlash(parentPath)

        # Ensure metrics_df.csv is in the parentPath
        default_metrics_path = parentPath + self.output_filenames['metrics_df']
        if not os.path.exists(default_metrics_path):
                print('Precalculated metrics at ' + default_metrics_path + ' do not
exist')
```

```
                return


        # Load default precalculated metric vectors
        self.metrics_df = pd.read_csv(default_metrics_path, index_col=0)
        self.metrics_df = self.metrics_df[self.metric_names]
        print('Loaded precalculated metrics')


        # Get datetime string for when these metrics were calculated
        timestamp = datetime.fromtimestamp(os.path.getmtime(default_metrics_path))
        self.metrics_datetime = timestamp.strftime("%Y-%m-%d, %H:%M:%S")


        # Restrict to videos for the given registry sample
        try:
                self.metrics_df =
self.metrics_df.loc[self.filtered_registry_sample.index]
        except KeyError:
                # The registry sample contains indices for videos with unprocessed
metrics
                print('\n***New metrics must be calculated***')
                print('\tp = PCA_tools()')
                print('\tp.new(filter_errors=False)')
                print('\tp.computeMetrics()')
                print('\n\tafter saving output, place metrics_df.csv in the default
parentPath: {}'.format(parentPath))
                return


        return self.metrics_df


    # Add strain and videoname columns to a dataframe
    # Requires df.index to be registry index values
    def appendColumns(self, df):
        new_df = df.copy()
        strains = [self.registry.loc[i,'strain'] for i in new_df.index]
        runs = [self.registry.loc[i,'run'] for i in new_df.index]
        scopes = [self.registry.loc[i,'scope'] for i in new_df.index]
        if 'strain' not in new_df.columns:
```

```
                        new_df.insert(0, 'strain', strains)
                if 'scope' not in new_df.columns:
                        new_df.insert(0, 'scope', scopes)
                if 'run' not in new_df.columns:
                        new_df.insert(0, 'run', runs)
                return new_df


        # Prints a summary of processing errors
        def errorSummary(self, print_summary=True):
                errorString = ''
                for i in self.registry_sample[self.registry_sample.error == 1].index.values:
                        run = registry.loc[i, 'run']
                        scope = registry.loc[i, 'scope']
                        errorString += 'Video {} (r{}s{}) had large intial clumps\n'.format(i,
run, scope)


                if self.nan_indices is not None:
                        errorString += '\n\n{} videos removed due to missing metric values
(indices in self.nan_indices)'.format(len(self.nan_indices))


                if print_summary:
                        print(errorString)
                else:
                        return errorString


        # Returns whether to proceed with saving and where
        # based on whether conflicts exist
        # and if so what the user specifies
        def checkSaveConflicts(self, savepath):
                # Prompt user if anything will be overwritten
                conflictingFiles = [f for f in self.output_filenames.values() if
os.path.exists(savepath + f)]
                if len(conflictingFiles) == 0:
                        proceed = True
                else:
                        print('The following files will be overwritten in {}:'.format(savepath))
```

```
                        for f in conflictingFiles:
                                print('\t' + f)
                        overwriteAnswer = '?'
                        while overwriteAnswer not in ['y', 'n', '']:
                                overwriteAnswer = input('Do you want to overwrite these files?
y/[n] ')
                        # Give user the option to save in a new directory
                        if overwriteAnswer != 'y':
                                newDirAnswer = '?'
                                while newDirAnswer not in ['y', 'n', '']:
                                        newDirAnswer = input('Do you want to save in a new
directory? [y]/n ')
                                if newDirAnswer != 'n':
                                        savepath = self.makeNewDir(self.parentPath)
                                        proceed = True
                                # Otherwise abort save
                                else:
                                        proceed = False
                        else:
                                proceed = True

                return proceed, savepath


        def filterNan(self, df, store_nan_indices=False, return_nan_indices=False):
                # Remove vectors that have missing values
                filtered_df = df.dropna()
                kept_indices = filtered_df.index
                if len(df) != len(filtered_df):
                        print('{} rows of {} removed due to missing values'.format(len(df)-
len(filtered_df), len(df)))

                # Store indices of which vectors were nan, if desired
                if store_nan_indices:
                        self.nan_indices = [i for i in df.index if i not in kept_indices]

                # Return indices of which vectors were nan, if desired
```

```
                # Otherwise, just return the filtered dataframe
                if return_nan_indices:
                        return nan_indices, filtered_df
                else:
                        return filtered_df

        def makeNewDir(self, parentPath=None):
                # Prepare parent directory for new output folder
                if parentPath is None:
                        parentPath = self.parentPath
                parentPath = self.ensureFinalSlash(parentPath)

                # Make folder for new output
                newDirSuccess = False
                while not newDirSuccess:
                        # Prompt user for new output folder name
                        newDirName = input('New output folder name? ')
                        # Make the new folder if possible
                        if not os.path.exists(parentPath + newDirName):
                                try:
                                        os.mkdir(parentPath + newDirName)
                                        newDirSuccess = True
                                        print('Folder ' + parentPath + newDirName + ' created')
                                except:
                                        print('Folder ' + parentPath + newDirName + ' could not be
created')
                        else:
                                # Allow an empty, preexisting folder to be used
                                if len(os.listdir(parentPath + newDirName)) == 0:
                                        newDirSuccess = True
                                else:
                                        print('Folder ' + parentPath + newDirName + ' already
exists and is not empty')

                return self.ensureFinalSlash(parentPath + newDirName)
```

```
	def pickLoadpath(self, parentPath=None):
		# Get a parentPath containing the available loadpath folders
		if parentPath is None:
			parentPath = self.parentPath
		parentPath = self.ensureFinalSlash(parentPath)

		# Get all folders in the parentPath
		allContents = os.listdir(parentPath)
		allFolders = [f for f in allContents if os.path.isdir(parentPath + f)]
		if len(allFolders) == 0:
			print('No folders in directory ' + parentPath)
			return

		# Prompt user to pick a folder
		print('Available folders:')
		for f in allFolders:
			print('\t'+f)
		folderPick = '?'
		while not os.path.exists(parentPath + folderPick):
			folderPick = input('')

		return self.ensureFinalSlash(parentPath + folderPick)

	def prepRegistrySample(self, registry_sample=None, filter_errors=True):
		# Default to using all videos if no registry sample is given
		if registry_sample is None:
			registry_sample = self.registry

		# Ignore videos that are unprocessed
		# If desired, also filter out videos with non-fatal errors
		if filter_errors:
			filtered_registry_sample = registry_sample[(registry_sample.processed ==
True) & (registry_sample.error == 0)]
			if len(filtered_registry_sample) != len(registry_sample):
```

```
                        print('{} videos of {} with nonzero error codes ignored (summary
in self.errorSummary)'.format(len(registry_sample)-len(filtered_registry_sample),
len(registry_sample)))
                else:
                        # An odd error code indicates a fatal error (missing data for the video)
                        filtered_registry_sample = registry_sample[registry_sample.processed &
(registry_sample.error%2 == 0)]
                        if len(filtered_registry_sample) != len(registry_sample):
                                print('{} videos of {} with fatal error codes ignored (summary in
self.errorSummary)'.format(len(registry_sample)-len(filtered_registry_sample),
len(registry_sample)))



                return registry_sample, filtered_registry_sample


        def ensureFinalSlash(self, path):
                if path[-1] != '/':
                        path += '/'
                return path
```

References

[1] A. Xavier da Silveira dos Santos and P. Liberali, *From Single Cells to Tissue Self-organization*, FEBS J. **286**, 1495 (2019).

[2] P. A. Janmey and R. T. Miller, *Mechanisms of Mechanical Signaling in Development and Disease*, J. Cell Sci. **124**, 9 (2011).

[3] W. Zheng and J. R. Holt, *The Mechanosensory Transduction Machinery in Inner Ear Hair Cells*, Annu. Rev. Biophys. **50**, 31 (2021).

[4] B. Martinac, M. Buechner, A. H. Delcour, J. Adler, and C. Kung, *Pressure-Sensitive Ion Channel in Escherichia Coli.*, Proc. Natl. Acad. Sci. **84**, 2297 (1987).

[5] K. R. Levental et al., *Matrix Crosslinking Forces Tumor Progression by Enhancing Integrin Signaling*, Cell **139**, 891 (2009).

[6] C. W. Macosko, *Rheology - Principles, Measurements and Applications* (John Wiley & Sons, 1994).

[7] P. Stoodley, R. Cargo, C. J. Rupp, S. Wilson, and I. Klapper, *Biofilm Material Properties as Related to Shear-Induced Deformation and Detachment Phenomena*, J. Ind. Microbiol. Biotechnol. **29**, 361 (2002).

[8] L. Pavlovsky, J. G. Younger, and M. J. Solomon, *In Situ Rheology of Staphylococcus Epidermidis Bacterial Biofilms*, Soft Matter **9**, 122 (2013).

[9] B. W. Towler, A. Cunningham, P. Stoodley, and L. McKittrick, *A Model of Fluid-Biofilm Interaction Using a Burger Material Law*, Biotechnol. Bioeng. **96**, 259 (2007).

[10] S. Nguyen, M.-H. Vu, M. Vu, and T. Nguyen, *Generalized Maxwell Model for Micro-Cracked Viscoelastic Materials*, Int. J. Damage Mech. **26**, 697 (2017).

[11] T. Zhang, N. G. Cogan, and Q. Wang, *Phase Field Models for Biofilms. I. Theory and One-Dimensional Simulations*, SIAM J. Appl. Math. **69**, 641 (2008).

[12] M. C. Flannery, *Bacteria Everywhere*, Am. Biol. Teach. **72**, 513 (2010).

[13] C. Wolgemuth, *Does Cell Biology Need Physicists?*, Physics (College. Park. Md). (2011).

[14] O. A. Igoshin et al., *Biophysics at the Coffee Shop: Lessons Learned Working with George Oster*, Mol. Biol. Cell **30**, 1882 (2019).

[15] J. Yang, P. E. Arratia, A. E. Patteson, and A. Gopinath, *Quenching Active Swarms : Effects of Light Exposure on Collective Motility in Swarming Serratia Marcescens*, (2019).

[16] S. Macfarlane, H. Steed, and G. T. Macfarlane, *Intestinal Bacteria and Inflammatory Bowel Disease*, Crit. Rev. Clin. Lab. Sci. **46**, 25 (2009).

[17] T. Essock-Burns, A. Wepprich, A. Thompson, and D. Rittschof, *Enzymes Manage Biofilms on Crab Surfaces Aiding in Feeding and Antifouling*, J. Exp. Mar. Bio. Ecol. **479**, 106 (2016).

[18] X. pan Guo, X. li Sun, Y. ru Chen, L. Hou, M. Liu, and Y. Yang, *Antibiotic Resistance Genes in Biofilms on Plastic Wastes in an Estuarine Environment*, Sci. Total Environ. **745**, (2020).

[19] Y. Zhang, H. Ge, W. Lin, Y. Song, F. Ge, X. Huang, and X. Meng, *Effect of Different Disinfection Treatments on the Adhesion and Separation of Biofilm on Stainless Steel Surface*, Water Sci. Technol. **83**, 877 (2021).

[20] J. B. Goldberg and G. B. Pier, *The Role of the CFTR in Susceptibility to Pseudomonas Aeruginosa Infections in Cystic Fibrosis*, Trends Microbiol. **8**, 514 (2000).

[21] N. B. Caberoy, R. D. Welch, J. S. Jakobsen, S. C. Slater, and A. G. Garza, *Global Mutational Analysis of NtrC-Like Activators in* Myxococcus Xanthus *: Identifying Activator Mutants Defective for Motility and Fruiting Body Development*, J. Bacteriol. **185**, 6083 (2003).

[22] H. C. Berg and L. Turner, *Torque Generated by the Flagellar Motor of Escherichia Coil*, Biophys. J. **65**, 2201 (1993).

[23] M. Washizu, Y. Kurahashi, H. Iochi, O. Kurosawa, S. I. Aizawa, H. Hotani, S. Kudo, and Y. Magariyama, *Dielectrophoretic Measurement of Bacterial Motor Characteristics*, IEEE Trans. Ind. Appl. **29**, 286 (1993).

[24] N. Wadhwa and H. C. Berg, *Bacterial Motility: Machinery and Mechanisms*, Nat. Rev. Microbiol. **20**, 161 (2022).

[25] B. Maier and G. C. L. Wong, *How Bacteria Use Type IV Pili Machinery on Surfaces*, Trends in Microbiology.

[26] B. Nan and D. R. Zusman, *Uncovering the Mystery of Gliding Motility in the Myxobacteria*, Annu. Rev. Genet. **45**, 21 (2011).

[27] D. B. Kearns, *A Field Guide to Bacterial Swarming Motility*, Nat. Rev. Microbiol. **8**, 634 (2010).

[28] E. J. G. Pollitt and S. P. Diggle, *Defining Motility in the Staphylococci*, Cell. Mol. Life Sci. **74**, 2943 (2017).

[29] Q. Zhang, D. Nguyen, J. S. B. Tai, X. J. Xu, J. Nijjer, X. Huang, Y. Li, and J. Yan, *Mechanical Resilience of Biofilms toward Environmental Perturbations Mediated by Extracellular Matrix*, Adv. Funct. Mater. **32**, 1 (2022).

[30] R. Rusconi, S. Lecuyer, L. Guglielmini, and H. A. Stone, *Laminar Flow around Corners Triggers the Formation of Biofilm Streamers*, J. R. Soc. Interface **7**, 1293 (2010).

[31] Y. F. Inclan, A. Persat, A. Greninger, J. Von Dollen, J. Johnson, N. Krogan, Z. Gitai, and J. N. Engel, *A Scaffold Protein Connects Type IV Pili with the Chp Chemosensory System to Mediate Activation of Virulence Signaling in Pseudomonas Aeruginosa*, Mol. Microbiol. **101**, 590 (2016).

[32] A. Seminara, T. E. Angelini, J. N. Wilking, H. Vlamakis, S. Ebrahim, R. Kolter, D. A. Weitz, and M. P. Brenner, *Osmotic Spreading of Bacillus Subtilis Biofilms Driven by an Extracellular Matrix*, Proc. Natl. Acad. Sci. **109**, 1116 (2012).

[33] J. Yan, C. D. Nadell, H. A. Stone, N. S. Wingreen, and B. L. Bassler, *Extracellular-Matrix-Mediated Osmotic Pressure Drives Vibrio Cholerae Biofilm Expansion and Cheater Exclusion*, Nat. Commun. **8**, 327 (2017).

[34] J. N. Wilking, T. E. Angelini, A. Seminara, M. P. Brenner, and D. A. Weitz, *Biofilms as*

Complex Fluids, MRS Bull. **36**, 385 (2011).

[35] T. Tanaka and D. J. Fillmore, *Kinetics of Swelling of Gels*, J. Chem. Phys. **70**, 1214 (1979).

[36] C. Fei, S. Mao, J. Yan, R. Alert, H. A. Stone, B. L. Bassler, N. S. Wingreen, and A. Košmrlj, *Nonuniform Growth and Surface Friction Determine Bacterial Biofilm Morphology on Soft Substrates*, Proc. Natl. Acad. Sci. **117**, 7622 (2020).

[37] S. Srinivasan, C. N. Kaplan, and L. Mahadevan, *A Multiphase Theory for Spreading Microbial Swarms and Films*, Elife **8**, (2019).

[38] M. E. Davey, N. C. Caiazza, and G. A. O'Toole, *Rhamnolipid Surfactant Production Affects Biofilm Architecture in Pseudomonas Aeruginosa PAO1*, J. Bacteriol. **185**, 1027 (2003).

[39] F. Song and D. Ren, *Stiffness of Cross-Linked Poly(Dimethylsiloxane) Affects Bacterial Adhesion and Antibiotic Susceptibility of Attached Cells*, Langmuir **30**, 10354 (2014).

[40] V. D. Gordon and L. Wang, *Bacterial Mechanosensing:The Force Will Be with You, Always*, J. Cell Sci. **132**, (2019).

[41] M. D. Koch, M. E. Black, E. Han, J. W. Shaevitz, and Z. Gitai, *Pseudomonas Aeruginosa Distinguishes Surfaces by Stiffness Using Retraction of Type IV Pili*, 1 (2022).

[42] J. Muñoz-Dorado, F. J. Marcos-Torres, E. García-Bravo, A. Moraleda-Muñoz, and J. Pérez, *Myxobacteria: Moving, Killing, Feeding, and Surviving Together*, Front. Microbiol. **7**, 1 (2016).

[43] I. T. Jolliffe, *Principal Component Analysis for Special Types of Data*, in *Principal Component Analysis* (Springer-Verlag, New York, 2002), pp. 338–372.

[44] J. William Costerton, Z. Lewandowski, D. E. Caldwell, D. R. Korber, and H. M. Lappin-Scott, Microbial Biofilms, 1995.

[45] L. Hall-Stoodley, J. W. Costerton, and P. Stoodley, *Bacterial Biofilms: From the Natural Environment to Infectious Diseases*, Nat. Rev. Microbiol. **2**, 95 (2004).

[46] R. M. Harshey, *Bacterial Motility on a Surface: Many Ways to a Common Goal*, Annu.

Rev. Microbiol. **57**, 249 (2003).

[47] A. Doostmohammadi, S. P. Thampi, and J. M. Yeomans, *Defect-Mediated Morphologies in Growing Cell Colonies*, Phys. Rev. Lett. **117**, 1 (2016).

[48] A. Patteson, A. Gopinath, and P. E. Arratia, *The Propagation of Active-Passive Interfaces in Bacterial Swarms*, Nat. Commun. (2018).

[49] K. Little, J. Austerman, J. Zheng, and K. A. Gibbs, *Cell Shape and Population Migration Are Distinct Steps of Proteus Mirabilis Swarming That Are Decoupled on High-Percentage Agar*, J. Bacteriol. **201**, (2019).

[50] Y. Nakamura, N. N. Yamamoto, Y. Kino, N. N. Yamamoto, S. Kamei, H. Mori, K. Kurokawa, and N. Nakashima, *Establishment of a Multi-Species Biofilm Model and Metatranscriptomic Analysis of Biofilm and Planktonic Cell Communities*, Appl. Microbiol. Biotechnol. **100**, 7263 (2016).

[51] C. Prigent-Combaret, O. Vidal, C. Dorel, and P. Lejeune, *Abiotic Surface Sensing and Biofilm-Dependent Regulation of Gene Expression in Escherichia Coli*, J. Bacteriol. **181**, 5993 (1999).

[52] A. P. Hitchens and M. C. Leikind, *The Introduction of Agar-Agar into Bacteriology*, J. Bacteriol. **37**, 485 (1939).

[53] F. Pereira-Pacheco, D. Robledo, L. Rodríguez-Carvajal, and Y. Freile-Pelegrín, *Optimization of Native Agar Extraction from Hydropuntia Cornea from Yucatán, México*, Bioresour. Technol. **98**, 1278 (2007).

[54] G. O. Phillips and P. A. Williams, *Handbook of Hydrocolloids*, 2nd ed. (Woodhead Publishing Limited, 2009).

[55] A. J. Fulthorpe, *The Variability in Gel-Producting Properties of Commercial Agar and Its Influence on Bacterial Growth*, J. Hyg. (Lond). **49**, 127 (1951).

[56] J. Yan, M. D. Bradley, J. Friedman, and R. D. Welch, *Phenotypic Profiling of ABC Transporter Coding Genes in Myxococcus Xanthus*, Front. Microbiol. **5**, 1 (2014).

[57] K. K. Chelvam, L. C. Chai, and K. L. Thong, *Variations in Motility and Biofilm Formation of Salmonella Enterica Serovar Typhi*, Gut Pathog. **6**, 1 (2014).

[58] K. W. Kolewe, S. R. Peyton, and J. D. Schiffman, *Fewer Bacteria Adhere to Softer Hydrogels*, ACS Appl. Mater. Interfaces **7**, 19562 (2015).

[59] E. B. Steager, C. B. Kim, and M. J. Kim, *Dynamics of Pattern Formation in Bacterial Swarms*, Phys. Fluids **20**, (2008).

[60] R. Van Houdt, M. Givskov, and C. W. Michiels, *Quorum Sensing in Serratia*, FEMS Microbiol. Rev. **31**, 407 (2007).

[61] H. H. Tuson and D. B. Weibel, *Bacteria-Surface Interactions*, Soft Matter **9**, 4368 (2013).

[62] E. E. Charrier, K. Pogoda, R. G. Wells, and P. A. Janmey, *Control of Cell Morphology and Differentiation by Substrates with Independently Tunable Elasticity and Viscous Dissipation*, Nat. Commun. **9**, 449 (2018).

[63] C. A. Grattoni, H. H. Al-Sharji, C. Yang, A. H. Muggeridge, and R. W. Zimmerman, *Rheology and Permeability of Crosslinked Polyacrylamide Gel*, J. Colloid Interface Sci. **240**, 601 (2001).

[64] Y. Abidine, V. M. Laurent, R. Michel, A. Duperray, L. I. Palade, and C. Verdier, *Physical Properties of Polyacrylamide Gels Probed by AFM and Rheology*, EPL (Europhysics Lett. **109**, 38003 (2015).

[65] C. E. Kandow, P. C. Georges, P. A. Janmey, and K. A. Beningo, *Polyacrylamide Hydrogels for Cell Mechanics: Steps toward Optimization and Alternative Uses*, Methods Cell Biol. **83**, 29 (2007).

[66] Y. Hu, E. P. Chan, J. J. Vlassak, and Z. Suo, *Poroelastic Relaxation Indentation of Thin Layers of Gels*, J. Appl. Phys. **110**, 108 (2011).

[67] Q. M. Wang, A. C. Mohan, M. L. Oyen, and X. H. Zhao, *Separating Viscoelasticity and Poroelasticity of Gels with Different Length and Time Scales*, Acta Mech. Sin. Xuebao **30**, 20 (2014).

[68] J. Yan, C. Fei, S. Mao, A. Moreau, N. S. Wingreen, A. Košmrlj, H. A. Stone, and B. L. Bassler, *Mechanical Instability and Interfacial Energy Drive Biofilm Morphogenesis*, Elife **8**, 1 (2019).

[69] S. Kesel, B. Von Bronk, C. Falcón García, A. Götz, O. Lieleg, and M. Opitz, *Matrix Composition Determines the Dimensions of: Bacillus Subtilis NCIB 3610 Biofilm Colonies Grown on LB Agar*, RSC Adv. **7**, 31886 (2017).

[70] A. Yang, W. S. Tang, T. Si, and J. X. Tang, *Influence of Physical Effects on the Swarming Motility of Pseudomonas Aeruginosa*, Biophys. J. **112**, 1462 (2017).

[71] A. Be'er and G. Ariel, *A Statistical Physics View of Swarming Bacteria*, Mov. Ecol. **7**, 9 (2019).

[72] V. Gordon, L. Bakhtiari, and K. Kovach, *From Molecules to Multispecies Ecosystems: The Roles of Structure in Bacterial Biofilms*, Phys. Biol. **16**, 041001 (2019).

[73] D. Stickler and G. Hughes, *Ability of Proteus Mirabilis to Swarm over Urethral Catheters*, Eur. J. Clin. Microbiol. Infect. Dis. **18**, 206 (1999).

[74] G. Liu, A. Patch, F. Bahar, D. Yllanes, R. D. Welch, M. C. Marchetti, S. Thutupalli, and J. W. Shaevitz, *Self-Driven Phase Transitions Drive Myxococcus Xanthus Fruiting Body Formation*, Phys. Rev. Lett. **122**, 248102 (2019).

[75] J. M. Skerker and H. C. Berg, *Direct Observation of Extension and Retraction of Type IV Pili*, Proc. Natl. Acad. Sci. U. S. A. **98**, 6901 (2001).

[76] M. Sun, M. Wartel, E. Cascales, J. W. Shaevitz, and T. Mignot, *Motor-Driven Intracellular Transport Powers Bacterial Gliding Motility*, Proc. Natl. Acad. Sci. U. S. A. **108**, 7559 (2011).

[77] L. D. Landau, L. P. Pitaevskii, and A. M. Kosevich, *Theory of Elasticity*, 3rd ed., Vol. 7 (Butterworth-Heineman, 2012).

[78] J. P. Butler, I. M. Tolić-Nørrelykke, B. Fabry, and J. J. Fredberg, *Traction Fields, Moments, and Strain Energy That Cells Exert on Their Surroundings*, Am J Physiol Cell Physiol **282**,

C595 (2002).

[79] U. S. Schwarz, N. Q. Balaban, D. Riveline, A. Bershadsky, B. Geiger, and S. A. Safran, *Calculation of Forces at Focal Adhesions from Elastic Substrate Data: The Effect of Localized Force and the Need for Regularization*, Biophys. J. **83**, 1380 (2002).

[80] B. Sabass, M. D. Koch, G. Liu, H. A. Stone, and J. W. Shaevitz, *Force Generation by Groups of Migrating Bacteria*, Proc. Natl. Acad. Sci. **114**, 7266 (2017).

[81] A. Cont, T. Rossy, Z. Al-Mayyah, and A. Persat, *Biofilms Deform Soft Surfaces and Disrupt Epithelia*, Elife **9**, e56533 (2020).

[82] M. T. Ho Thanh, A. Grella, D. Kole, S. Ambady, and Q. Wen, *Vimentin Intermediate Filaments Modulate Cell Traction Force but Not Cell Sensitivity to Substrate Stiffness*, Cytoskeleton 1 (2021).

[83] R. Zielinski, C. Mihai, D. Kniss, and S. N. Ghadiali, *Finite Element Analysis of Traction Force Microscopy: Influence of Cell Mechanics, Adhesion, and Morphology*, J. Biomech. Eng. **135**, 1 (2013).

[84] Z. Yang, J. S. Lin, J. Chen, and J. H. C. Wang, *Determining Substrate Displacement and Cell Traction Fields-a New Approach*, J. Theor. Biol. **242**, 607 (2006).

[85] A. Persat, Y. F. Inclan, J. N. Engel, H. A. Stone, and Z. Gitai, *Type IV Pili Mechanochemically Regulate Virulence Factors in Pseudomonas Aeruginosa*, Proc. Natl. Acad. Sci. **112**, 7563 (2015).

[86] F. Song, H. Wang, K. Sauer, and D. Ren, *Cyclic-Di-GMP and OprF Are Involved in the Response of Pseudomonas Aeruginosa to Substrate Material Stiffness during Attachment on Polydimethylsiloxane (PDMS)*, Front. Microbiol. **9**, 110 (2018).

[87] V. T. Nayar, J. D. Weiland, C. S. Nelson, and A. M. Hodge, *Elastic and Viscoelastic Characterization of Agar*, J. Mech. Behav. Biomed. Mater. **7**, 60 (2012).

[88] R. M. Donlan and J. W. Costerton, *Biofilms: Survival Mechanisms of Clinically Relevant Microorganisms*, Clin. Microbiol. Rev. **15**, 167 (2002).

[89] W. C. Characklis, *Biofilms* (Wiley, 1990).

[90] H. Mikkelsen, Z. Duck, K. S. Lilley, and M. Welch, *Interrelationships between Colonies, Biofilms, and Planktonic Cells of Pseudomonas Aeruginosa*, J. Bacteriol. **189**, 2411 (2007).

[91] A. E. Patteson, A. Gopinath, M. Goulian, and P. E. Arratia, *Running and Tumbling with E. Coli in Polymeric Solutions*, Sci. Rep. **5**, 15761 (2015).

[92] N. Wadhwa, R. Phillips, and H. C. Berg, *Torque-Dependent Remodeling of the Bacterial Flagellar Motor*, Proc. Natl. Acad. Sci. **116**, 11764 (2019).

[93] P. P. Lele, B. G. Hosu, and H. C. Berg, *Dynamics of Mechanosensing in the Bacterial Flagellar Motor*, Proc. Natl. Acad. Sci. **110**, 11839 (2013).

[94] A. K. Harapanahalli, J. A. Younes, E. Allan, H. C. van der Mei, and H. J. Busscher, *Chemical Signals and Mechanosensing in Bacterial Responses to Their Environment*, PLoS Pathog **11**, e1005057 (2015).

[95] J. H. Naismith and I. R. Booth, *Bacterial Mechanosensitive Channels--MscS: Evolution's Solution to Creating Sensitivity in Function*, Annu Rev Biophys **41**, 157 (2012).

[96] E. Perozo, A. Kloda, D. M. Cortes, and B. Martinac, *Physical Principles Underlying the Transduction of Bilayer Deformation Forces during Mechanosensitive Channel Gating*, Nat Struct Biol **9**, 696 (2002).

[97] L. Casares, R. Vincent, D. Zalvidea, N. Campillo, D. Navajas, M. Arroyo, and X. Trepat, *Hydraulic Fracture during Epithelial Stretching*, Nat. Mater. **14**, 343 (2015).

[98] A. Hejazi and F. R. Falkiner, Serratia Marcescens, 1997.

[99] C. F. Guimarães, L. Gasperini, A. P. Marques, and R. L. Reis, *The Stiffness of Living Tissues and Its Implications for Tissue Engineering*, Nat. Rev. Mater. **5**, 351 (2020).

[100] J. Irianto, C. R. Pfeifer, Y. Xia, and D. E. Discher, *SnapShot: Mechanosensing Matrix*, Cell **165**, 1820 (2016).

[101] D. T. Butcher, T. Alliston, and V. M. Weaver, *A Tense Situation: Forcing Tumour*

Progression, Nat. Rev. Cancer **9**, 108 (2009).

[102] R. G. Wells, *Tissue Mechanics and Fibrosis*, Biochim. Biophys. Acta - Mol. Basis Dis. **1832**, 884 (2013).

[103] R. Sinkus, J. Lorenzen, D. Schrader, M. Lorenzen, M. Dargatz, and D. Holz, *High-Resolution Tensor MR Elastography for Breast Tumour Detection*, Phys. Med. Biol. **45**, 1649 (2000).

[104] M. Swoger, S. Gupta, E. E. Charrier, M. Bates, H. Hehnly, and A. E. Patteson, *Vimentin Intermediate Filaments Mediate Cell Morphology on Viscoelastic Substrates*, ACS Appl. Bio Mater. **5**, 552 (2022).

[105] T. Yeung, P. C. Georges, L. A. Flanagan, B. Marg, M. Ortiz, M. Funaki, N. Zahir, W. Ming, V. Weaver, and P. A. Janmey, *Effects of Substrate Stiffness on Cell Morphology, Cytoskeletal Structure, and Adhesion*, Cell Motil. Cytoskeleton **60**, 24 (2005).

[106] A. E. Patteson et al., *Loss of Vimentin Enhances Cell Motility through Small Confining Spaces*, Small **15**, 1903180 (2019).

[107] H. H. Tuson, L. D. Renner, and D. B. Weibel, *Polyacrylamide Hydrogels as Substrates for Studying Bacteria*, Chem. Commun. **48**, 1595 (2012).

[108] N. Mori and K.-A. Chang, *Introduction to MPIV*.

[109] W. C. Lin, K. R. Shull, C. Y. Hui, and Y. Y. Lin, *Contact Measurement of Internal Fluid Flow within Poly(n- Isopropylacrylamide) Gels*, J. Chem. Phys. **127**, (2007).

[110] E. P. Chan, Y. Hu, P. M. Johnson, Z. Suo, and C. M. Stafford, *Spherical Indentation Testing of Poroelastic Relaxations in Thin Hydrogel Layers*, Soft Matter **8**, 1492 (2012).

[111] C. Y. Hui, Y. Y. Lin, F. U. C. Chuang, K. R. Shull, and W. C. Lin, *A Contact Mechanics Method for Characterizing the Elastic Properties and Permeability of Gels*, J. Polym. Sci. Part B Polym. Phys. **44**, 359 (2006).

[112] G. Bradski, *The OpenCV Library*, Dr. Dobb's J. Softw. Tools (2000).

[113] G. Diss, D. Ascencio, A. DeLuna, and C. R. Landry, *Molecular Mechanisms of Paralogous*

Compensation and the Robustness of Cellular Networks, J. Exp. Zool. Part B Mol. Dev. Evol. **322**, 488 (2014).

[114] G. Giaever et al., *Functional Profiling of the Saccharomyces Cerevisiae Genome*, Nature **418**, 387 (2002).

[115] S. Ohno, *The Creation of a New Gene from a Redundant Duplicate of an Old Gene*, in *Evolution by Gene Duplication* (Springer Berlin Heidelberg, Berlin, Heidelberg, 1970), pp. 71–82.

[116] B. Vandersluis, J. Bellay, G. Musso, M. Costanzo, B. Papp, F. J. Vizeacoumar, A. Baryshnikova, B. Andrews, C. Boone, and C. L. Myers, *Genetic Interactions Reveal the Evolutionary Trajectories of Duplicate Genes*, Mol. Syst. Biol. **6**, 1 (2010).

[117] D. C. Krakauer and J. B. Plotkin, *Redundancy, Antiredundancy, and the Robustness of Genomes*, Proc. Natl. Acad. Sci. **99**, 1405 (2002).

[118] E. Kuzmin, M. Rahman, B. VanderSluis, M. Costanzo, C. L. Myers, B. J. Andrews, and C. Boone, *τ-SGA: Synthetic Genetic Array Analysis for Systematically Screening and Quantifying Trigenic Interactions in Yeast*, Nat. Protoc. **16**, 1219 (2021).

[119] E. J. Dean, J. C. Davis, R. W. Davis, and D. A. Petrov, *Pervasive and Persistent Redundancy among Duplicated Genes in Yeast*, PLoS Genet. **4**, (2008).

[120] E. A. Baker, S. P. R. Gilbert, S. M. Shimeld, and A. Woollard, *Extensive Non-Redundancy in a Recently Duplicated Developmental Gene Family*, BMC Ecol. Evol. **21**, (2021).

[121] K. Hannay, E. M. Marcotte, and C. Vogel, *Buffering by Gene Duplicates: An Analysis of Molecular Correlates and Evolutionary Conservation*, BMC Genomics **9**, (2008).

[122] H. B. Thomaides, E. J. Davison, L. Burston, H. Johnson, D. R. Brown, A. C. Hunt, J. Errington, and L. Czaplewski, *Essential Bacterial Functions Encoded by Gene Pairs*, J. Bacteriol. **189**, 591 (2007).

[123] G. C. diCenzo and T. M. Finan, *Genetic Redundancy Is Prevalent within the 6.7 Mb Sinorhizobium Meliloti Genome*, Mol. Genet. Genomics **290**, 1345 (2015).

[124] G. Butland et al., *ESGA: E. Coli Synthetic Genetic Array Analysis*, Nat. Methods **5**, 789 (2008).

[125] J. Lehár, A. Krueger, G. Zimmermann, and A. Borisy, *High-order Combination Effects and Biological Robustness*, Mol. Syst. Biol. **4**, 215 (2008).

[126] D. C. Rees, E. Johnson, and O. Lewinson, *ABC Transporters: The Power to Change*, Nat. Rev. Mol. Cell Biol. **10**, 218 (2009).

[127] B. S. Goldman et al., *Evolution of Sensory Complexity Recorded in a Myxobacterial Genome*, Proc. Natl. Acad. Sci. **103**, 15200 (2006).

[128] D. J. Bretl and J. R. Kirby, *Molecular Mechanisms of Signaling in Myxococcus Xanthus Development*, J. Mol. Biol. **428**, 3805 (2016).

[129] J. Abellón-Ruiz, D. Bernal-Bernal, M. Abellán, M. Fontes, S. Padmanabhan, F. J. Murillo, and M. Elías-Arnanz, *The CarD/CarG Regulatory Complex Is Required for the Action of Several Members of the Large Set of Myxococcus Xanthus Extracytoplasmic Function σ Factors*, Environ. Microbiol. **16**, 2475 (2014).

[130] C. Xie, H. Zhang, L. J. Shimkets, and O. A. Igoshin, *Statistical Image Analysis Reveals Features Affecting Fates of Myxococcus Xanthus Developmental Aggregates*, Proc. Natl. Acad. Sci. U. S. A. **108**, 5915 (2011).

[131] C. Orelle, K. Mathieu, and J. M. Jault, *Multidrug ABC Transporters in Bacteria*, Res. Microbiol. **170**, 381 (2019).

[132] C. Durmort and J. S. Brown, *Streptococcus Pneumoniae Lipoproteins and ABC Transporters*, in *Streptococcus Pneumoniae: Molecular Mechanisms of Host-Pathogen Interactions* (Elsevier Inc., 2015), pp. 181–206.

[133] A. Wagner and J. Wright, *Alternative Routes and Mutational Robustness in Complex Regulatory Networks*, BioSystems **88**, 163 (2007).

[134] V. A. Rhodius et al., *Design of Orthogonal Genetic Switches Based on a Crosstalk Map of Σs, Anti-Σs, and Promoters*, Mol. Syst. Biol. **9**, (2013).

[135] T. Mascher, A. B. Hachmann, and J. D. Helmann, *Regulatory Overlap and Functional Redundancy among Bacillus Subtilis Extracytoplasmic Function σ Factors*, J. Bacteriol. **189**, 6919 (2007).

[136] Y. Luo and J. D. Helmann, *Extracytoplasmic Function σ Factors with Overlapping Promoter Specificity Regulate Sublancin Production in Bacillus Subtilis*, J. Bacteriol. **191**, 4951 (2009).

[137] M. Paget, *Bacterial Sigma Factors and Anti-Sigma Factors: Structure, Function and Distribution*, Biomolecules **5**, 1245 (2015).

[138] E. J. Capra, B. S. Perchuk, J. M. Skerker, and M. T. Laub, *Adaptive Mutations That Prevent Crosstalk Enable the Expansion of Paralogous Signaling Protein Families*, Cell **150**, 222 (2012).

[139] M. T. Laub and M. Goulian, *Specificity in Two-Component Signal Transduction Pathways*, Annual Review of Genetics.

[140] M. A. Rowland and E. J. Deeds, *Crosstalk and the Evolution of Specificity in Two-Component Signaling*, Proc. Natl. Acad. Sci. U. S. A. **111**, 5550 (2014).

[141] J. M. Skerker, B. S. Perchuk, A. Siryaporn, E. A. Lubin, O. Ashenberg, M. Goulian, and M. T. Laub, *Rewiring the Specificity of Two-Component Signal Transduction Systems*, Cell **133**, 1043 (2008).

[142] A. I. Podgornaia and M. T. Laub, *Determinants of Specificity in Two-Component Signal Transduction*, Curr. Opin. Microbiol. **16**, 156 (2013).

[143] A. Gagarinova, M. Babu, J. Greenblatt, and A. Emili, *Mapping Bacterial Functional Networks and Pathways in Escherichia Coli Using Synthetic Genetic Arrays*, J. Vis. Exp. (2012).

[144] J. A. Johnstun, V. Shankar, S. S. Mokashi, L. T. Sunkara, U. E. Ihearahu, R. L. Lyman, T. F. C. Mackay, and R. R. H. Anholt, *Functional Diversification, Redundancy, and Epistasis among Paralogs of the Drosophila Melanogaster Obp50a-d Gene Cluster*, Mol. Biol. Evol. **38**,

2030 (2021).

[145] L. Plamann, Y. Li, B. Cantwell, and J. Mayor, *The Myxococcus Xanthus AsgA Gene Encodes a Novel Signal Transduction Protein Required for Multicellular Development*, J. Bacteriol. **177**, 2014 (1995).

[146] R. G. Taylor and R. D. Welch, *Recording Multicellular Behavior in Myxococcus Xanthus Biofilms Using Time-Lapse Microcinematography*, J. Vis. Exp. (2010).

[147] F. Madeira, M. Pearce, A. R. N. Tivey, P. Basutkar, J. Lee, O. Edbali, N. Madhusoodanan, A. Kolesnikov, and R. Lopez, *Search and Sequence Analysis Tools Services from EMBL-EBI in 2022*, Nucleic Acids Res. 1 (2022).

[148] D. B. Allan, T. Caswell, N. C. Keim, and C. M. van der Wel, *Trackpy*, Trackpy v0.4.2.

[149] L. J. Ritchie, E. R. Curtis, K. A. Murphy, and R. D. Welch, *Profiling Myxococcus Xanthus Swarming Phenotypes through Mutation and Environmental Variation*, J. Bacteriol. **203**, (2021).

[150] M. Kærn, T. C. Elston, W. J. Blake, and J. J. Collins, *Stochasticity in Gene Expression: From Theories to Phenotypes*, Nat. Rev. Genet. **6**, 451 (2005).

[151] J. Paulsson, O. G. Berg, and M. Ehrenberg, *Stochastic Focusing: Fluctuation-Enhanced Sensitivity of Intracellular Regulation*, Proc. Natl. Acad. Sci. **97**, 7148 (2000).

[152] M. Thattai and A. van Oudenaarden, *Stochastic Gene Expression in Fluctuating Environments*, Genetics **167**, 523 (2004).

[153] J. L. Spudich and D. E. Koshland, *Non-Genetic Individuality: Chance in the Single Cell*, Nature **262**, 467 (1976).

[154] M. Thattai and A. van Oudenaarden, *Intrinsic Noise in Gene Regulatory Networks*, Proc. Natl. Acad. Sci. **98**, 8614 (2001).

[155] L. Jelsbak and L. Søgaard-Andersen, *Pattern Formation: Fruiting Body Morphogenesis in Myxococcus Xanthus*, Curr. Opin. Microbiol. **3**, 637 (2000).

[156] G. B. Ruvkun and F. M. Ausubel, *A General Method for Site-Directed Mutagenesis in Prokaryotes*, Nature **289**, 85 (1981).

[157] A. M. Kierzek, J. Zaim, and P. Zielenkiewicz, *The Effect of Transcription and Translation Initiation Frequencies on the Stochastic Fluctuations in Prokaryotic Gene Expression*, J. Biol. Chem. **276**, 8165 (2001).

[158] D. J. Wilkinson, *Stochastic Modelling for Quantitative Description of Heterogeneous Biological Systems*, Nat. Rev. Genet. **10**, 122 (2009).

[159] H. Nordberg, M. Cantor, S. Dusheyko, S. Hua, A. Poliakov, I. Shabalov, T. Smirnova, I. V. Grigoriev, and I. Dubchak, *The Genome Portal of the Department of Energy Joint Genome Institute: 2014 Updates*, Nucleic Acids Res. **42**, 26 (2014).

[160] M. Jeanmougin, A. de Reynies, L. Marisa, C. Paccard, G. Nuel, and M. Guedj, *Should We Abandon the T-Test in the Analysis of Gene Expression Microarray Data: A Comparison of Variance Modeling Strategies*, PLoS One **5**, 1 (2010).

[161] H. Vuong, K. Shedden, Y. Liu, and D. M. Lubman, *Outlier-Based Differential Expression Analysis in Proteomics Studies*, J. Proteomics Bioinforma. **4**, 116 (2011).

[162] E. R. Dougherty and I. Shmulevich, *On the Limitations of Biological Knowledge*, Curr. Genomics **13**, 574 (2012).

[163] K. Sankaran and S. P. Holmes, *Latent Variable Modeling for the Microbiome*, Biostatistics **20**, 599 (2019).

[164] A. Smiti, *A Critical Overview of Outlier Detection Methods*, Comput. Sci. Rev. **38**, 100306 (2020).

[165] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, *When Is "Nearest Neighbor" Meaningful?*, in *Database Theory -- ICDT'99*, edited by C. Beeri and P. Buneman (Springer Berlin Heidelberg, 1999), pp. 217–235.

[166] U. N. Singh, *Polyribosomes and Unstable Messenger RNA: A Stochastic Model of Protein Synthesis*, J. Theor. Biol. **25**, 444 (1969).

[167] O. G. Berg, *A Model for the Statistical Fluctuations of Protein Numbers in a Microbial Population*, J. Theor. Biol. **71**, 587 (1978).

[168] M. S. H. Ko, *A Stochastic Model for Gene Induction*, J. Theor. Biol. **153**, 181 (1991).

[169] F. M. F. Nunes, A. C. Aleixo, A. R. Barchuk, A. D. Bomtorin, C. M. Grozinger, and Z. L. P. Simões, *Non-Target Effects of Green Fluorescent Protein (GFP)-Derived Double-Stranded RNA (DsRNA-GFP) Used in Honey Bee RNA Interference (RNAi) Assays*, Insects **4**, 90 (2013).

[170] J. K. Lodge, K. Weston-Hafer, and D. E. Berg, *Transposon Tn5 Target Specificity: Preference for Insertion at G/C Pairs.*, Genetics **120**, 645 (1988).

[171] L. Kroos, A. Kuspa, and D. Kaiser, *A Global Analysis of Developmentally Regulated Genes in Myxococcus Xanthus*, Dev. Biol. **117**, 252 (1986).

[172] R. A. Jilk, J. C. Makris, L. Borchardt, and W. S. Reznikoff, *Implications of Tn5-Associated Adjacent Deletions*, J. Bacteriol. **175**, 1264 (1993).

[173] H. H. Kazazian, *Mobile Elements: Drivers of Genome Evolution*, Science (80-.). **303**, 1626 (2004).

[174] D. Kaiser, *Social Gliding Is Correlated with the Presence of Pili in Myxococcus Xanthus.*, Proc. Natl. Acad. Sci. U. S. A. **76**, 5952 (1979).

[175] D. Wall and D. Kaiser, *Alignment Enhances the Cell-to-Cell Transfer of Pilus Phenotype*, Proc. Natl. Acad. Sci. U. S. A. **95**, 3054 (1998).

[176] H. B. Kaplan, A. Kuspa, and D. Kaiser, *Suppressors That Permit A-Signal-Independent Developmental Gene Expression in Myxococcus Xanthus*, J. Bacteriol. **173**, 1460 (1991).

[177] M. Varon and E. Rosenberg, *Transcriptional Regulation of Genes Required for Antibiotic TA Synthesis in Myxococcus Xanthus*, FEMS Microbiol. Lett. **136**, 203 (1996).

Merrill E. Asp

Curriculum Vitae

373 Physics Building, Syracuse University
masp01@syr.edu – (775) 720-7201

Education

May 2023 (projected graduation date)
Ph.D. Physics
Syracuse University

2016
B.S. Physics, Math minor
Brigham Young University

Research Assistantships

2019 - Current
SU Physics Dept./Bioinspired Institute, Alison Patteson
Bacterial culture. Fluorescence microscopy. Time-lapse microscopy. Bulk rheology. Hydrogel synthesis (PAA). Image processing with ImageJ and Python. Mentored five undergraduates and three high school students.

- Synthesized a range of hydrogel substrates for mechanosensing bacterial experiments, and characterized their surface energy, permeability, and rheology
- Studied the effects of substrate mechanics on bacterial growth across multiple species
- Developed software for supervising and automating image processing
- Developed high-throughput imaging hardware and software in collaboration with Roy Welch (SU Biology) to study collective bacterial motion in *Myxococcus xanthus*

2014 - 2016
BYU Physics Dept., Mark Transtrum
Dimensional reduction of analytical cell signaling models with Python.

2015 - 2016
BYU College of Humanities, Christopher Oscarson
Trained part-of-speech taggers on large literary corpora. Analysis with R.

Publications

(Featured article)
Alison Patteson, **Merrill E. Asp**, Paul Janmey, "Materials science and mechanosensitivity of living matter" *Applied Physics Reviews* (2022) doi.org/10.1063/5.0071648

Merrill E. Asp, Minh Tri Ho Thanh, Danielle A. Germann, Robert J. Carroll, Alana Franceski, Roy D. Welch, Arvind Gopinath, Alison E. Patteson, "Spreading rates of bacterial colonies depend on substrate stiffness and permeability" *PNAS Nexus* (2022) doi.org/10.1093/pnasnexus/pgac025

Merrill E. Asp, Elise Jutzeler, Katherine Kerr, Dawei Song, Alison E. Patteson, "A torsion-based rheometer for measuring viscoelastic material properties" *The Biophysicist* (2022) doi.org/10.35459/tbp.2020.000172

(In review)
Merrill E. Asp*, Jessica Comstock*, Isabella Lee, Fatmagül Bahar, Alison E. Patteson, Roy D. Welch, "Phenotypic similarity as an indicator of functional redundancy within large homologous groups" biorxiv.org/content/10.1101/2022.07.25.501402v1

(In review)
Joshua Tamayo, Yuchen Zhang, **Merrill E. Asp**, Alison E. Patteson, Arezoo M. Ardekani, Arvind Gopinath, "Swarming bacterial fronts: Dynamics and morphology of active swarm interfaces propagating through passive frictional domains" *Soft Matter* biorxiv.org/content/10.1101/2020.04.18.048637v3

(In preparation)
Merrill E. Asp*, Eduardo Caro*, Roy D. Welch, Alison E. Patteson, "Stochastic bounds of aggregation dynamics distinguish near-wild-type from wild-type strains in social bacteria"

(Submitted)
Merrill E. Asp, Minh-Tri Ho Thanh, Subarna Dutta, Jessica Comstock, Roy D. Welch, Alison E. Patteson, "Mechanobiology as a new tool for addressing the genotype-to-phenotype problem in microbiology" *Biophysics Reviews*

*Denotes co-first author contribution

Grants and Awards

- 2022 Social Justice Award (SU Physics Dept.)
- 2021 NYSS-APS Physics Outreach Grant ($2060)
    - Co-organized a new outreach program
    - Multiple visits to 3 high schools
    - 7 classes (physics, biology, earth science), over 100 students
- 2021 Summer Dissertation Fellowship ($4000)
- 2020 Outstanding Teaching Assistant Award

Talks and Presentations

- “Bridging the Gap: Bringing Research from Universities to Schools”
  2022 Topical Symposium of the New York State Section of the APS

- “Bacteria spreading increases with substrate stiffness”
  APS March Meeting 2022

- “How do bacteria ‘feel’ their environment?”
  Summer Soft and Living Matter Seminar,
  Syracuse University 2021

- “Control of biofilm growth through substrate mechanics”
  Virtual APS March Meeting 2021

- “How do bacteria ‘feel’ their environment?”
  Virtual APS Division of Fluid Dynamics (DFD) Meeting 2020

- “How do bacteria ‘feel’ their environment?”
  Stevenson Biomaterials Lecture & Research Poster Session, Syracuse University 2020

- “Searching for minimal mechanisms that can achieve biological adaptation” APS Four Corners Section 2015

- “Who can perform an observation?” Utah Academy of Sciences, Arts, & Letters Annual Conference 2011

Professional Development

- Certificate in University Teaching
- Future Professoriate Program (SU)
- Bioinspired Institute Graduate and Postdoctoral Development Program (SU)

Teaching Experience

Syracuse University

- Newtonian Mechanics Instructor, Summer 2019
- MCAT Prep Guest Lecturer, 2019
- Newtonian Mechanics Teaching Assistant, 2017 - 2020

Brigham Young University

- Physics Dept. Tutor (Introductory to senior-level physics courses), 2013 – 2016

Service

- Voting member of the College of Arts & Sciences Curriculum Committee, Syracuse University
- Organized Physics Outreach visits to local high schools, 2021 - 2022