ENGINEERING A FLUORESCENT BARCODING SYSTEM FOR HIGHLY MULTIPLEXED, SINGLE-CELL ANALYSIS OF BIOMOLECULAR AND CELLULAR LIBRARIES

by

Stefanie M. Berges

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Chemical Engineering

Summer 2017

ENGINEERING A FLUORESCENT BARCODING SYSTEM FOR HIGHLY MULTIPLEXED, SINGLE-CELL ANALYSIS OF BIOMOLECULAR AND CELLULAR LIBRARIES

by

Stefanie M. Berges

Approved: _____
Eric M. Furst, Ph.D.
Chair of the Department of Chemical & Biomolecular Engineering

Approved: _____
Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____
Ann L. Ardis, Ph.D.
Senior Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

David W. Colby, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Maciek R. Antoniewicz, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Wilfred Chen, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Edward R. Lyman, Ph.D.
Member of dissertation committee

ACKNOWLEDGMENTS

I would like to express my thanks and gratitude to many people who have supported me during graduate school. First and foremost, I would like to thank my family, specifically my parents Lou and Cindy, my in-laws who are like another set of parents to me Gary and Debbie, and my fiancé Greg. I would not have been able to make it to this point without their love and support. Also, I would like to thank all of my friends at the University of Delaware, especially those in my incoming class, whose kindness and comradery helped make my graduate school experience more colorful and well-rounded.

I would also like to thank my advisor Dr. David Colby for his mentorship and scientific guidance throughput my five years of scientific development at UD. In addition, I want to thank my departmental committee members, Dr. Wilfred Chen and Dr. Maciek Antoniewicz, for their helpful advice, scientific dialogue, and constructive feedback. In addition, I would like to thank Dr. Bramie Lenhoff for providing me mentorship, advice, and guidance. Also, I would like to thank my lab members, Ming Dong, Kyle Doolan, Kyle McHugh, Olga Morozova, and Elisa Ovadia for their scientific help and feedback, emotional support, and friendship. For their help in supporting my research project, I would like to thank all of my undergraduate researchers, especially Quentin Dubroff and Seth Ritter.

I would also like to thank the Chemical Engineering Department staff for their help and support with departmental activities. In particular, I would like to thank Kathie Young for her advice and vast knowledge of administrative policies, help with

paperwork, and her involvement in departmental recruitment. Karen Black in the Fisher biostockroom was very helpful and made ordering and restocking the lab much easier. I would also like to thank George Whitmeyer for his help with setup and lab safety, as well as organizing the Colburn holiday band, which I enjoyed participating in annually.

I would like to again reiterate my greatest thanks to the many people who have supported me throughput my time in graduate school. Sharing scientific successes and pitfalls with them has been invaluable, and I would not have made it to this point without them. I dedicate this work to my family, friends, and scientific mentors who have guided and supported me during my scientific journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A primary goal of biologists is to characterize the dynamic and complex behaviors of biological systems. Large, robust data sets that examine many biological molecules in a variety of conditions are desirable to gain a more multifaceted view of the cell. In addition, single-cell analysis technologies are used to characterize cellular heterogeneity and reduce biological noise that exists within isogenic populations. Noise in gene and protein expression arises from the stochasticity of underlying biochemical reactions, and can confer phenotypic variation which may be advantageous in certain circumstances. However, established technologies for high-throughput, single-cell proteomic analysis have limited throughput.

Multiplexing methods, such as fluorescent barcoding, can dramatically decrease the number of samples and in turn enable collection of more robust data sets including many replicates, conditions, and proteins. Fluorescent barcoding is a powerful tool for identification of different cells within a heterogeneous mixture using a unique fluorescent identifier or 'barcode'. Fluorescent barcoding can potentially reduce the number of samples thousands of fold, thereby facilitating massively-parallel single-cell analysis of biomolecular and cellular libraries. Current fluorescent cell barcoding systems are composed of small numbers of barcodes (~10-100), and in some cases are single-use and have toxicity issues.

The work described here presents the creation of the largest fluorescent barcoding system to date consisting of over 980 unique, genetically-encoded barcodes.

We made a library of plasmids encoding protein scaffolds that are composed of different lengths and combinations of epitope tags connected by flexible linkers. Cells expressing protein barcodes were identified by their distinct fluorescence upon immunolabeling. Multiplexing capability was greatly expanded by the discovery that barcodes with four distinct fluorescence intensities can be created by expression of different epitope tag repeat lengths. The effect of barcode expression on cellular growth, and the influence of different promoters and growth conditions on barcode expression was examined. A software package was developed to rapidly analyze barcode flow cytometry data, decreasing analysis time ~10-fold.

The multiplexing power of the fluorescent barcoding system was demonstrated in two applications. Barcode fusion did not hinder binding of an α-prion antibody for recombinant prion protein, suggesting barcodes can be used for multiplexed analysis of biomolecular libraries including high-throughput, quantitative protein-protein interaction studies. Barcodes were also used to simultaneously measure the dynamic response of endogenous yeast proteins in single-cells to environmental perturbations. Changes in protein abundance and variability as well as expression distributions were observed, suggesting cells may employ a bet-hedging mechanism to more quickly adapt to fluctuating environments. In addition, long epitope tag repeats facilitated immunodetection of endogenous, low abundance proteins in yeast by increasing the detection limit ~40-fold, potentially enabling analysis of > 1,600 low abundance proteins by flow cytometry.

Chapter 1

INTRODUCTION

1.1 Motivation and Goals

A major goal in the field of biology is to understand the complex and dynamic interactions and behaviors of biological systems, which can include molecules, cells, tissues, and organisms. Another area of interest in systems biology is to use large-scale data sets to derive computational predictive models for biological systems. For example, transcriptomics, proteomics, and interactomics data sets were synthesized to create a perturbed metabolic network model of galactose utilization in yeast [1]. Omics data is arguably the primary driving force behind systems biology [2]. Omics is the large-scale study of biological molecules that contribute to cellular function, and includes genomics, epigenomics, transcriptomics, proteomics, metabolomics, interactomics, and lipidomics [3]. Such studies aim to characterize the quantitative and dynamic behavior of biological molecules with spatiotemporal resolution and to elucidate cellular pathways and networks.

In order to fully describe a cellular system, large and robust data sets that examine many biological molecules under a variety of conditions and with many replicates are desirable. In addition, systems biology and omics studies can benefit from single-cell analysis technologies because they permit characterization of cellular heterogeneity and reduction of biological noise. Phenotypic heterogeneity exists within isogenic non-isogenic populations, and arises from the inherent stochasticity of gene and protein expression [4]. Stochastic processes create noise, or variation, and

has been observed in a variety of cellular systems [5], [6]. Gene and protein expression noise is thought to provide an advantage in certain circumstances, and underlies adaptive advantages such as bacterial persistence [7], [8] and bet-hedging [9]–[11]. In order to fully understand cellular behavior, single-cell analysis is desirable as it reduces noise and can reveal heterogeneity that is not captured by average measurements [12].

Established methods for high-throughput, single-cell proteomic analysis such as imaging, flow cytometry, microfluidics [13], and mass cytometry [14] often require cumbersome robotics systems and multi-well plates to analyze thousands of samples that are needed for systems-wide studies. As a result, single-cell proteomics has not been as widely implemented as single-cell genomics or transcriptomics [14]–[16]. Technologies to increase the throughput of single-cell proteomics methods can enhance systems biology studies by decreasing the number of samples required, and in turn enabling more replicates, conditions, and proteins to be studied.

Cellular fluorescent barcoding is a powerful tool that can be used to identify different types of cells within a heterogeneous mixture using a unique fluorescent identifier or 'barcode'. Thus, fluorescent barcoding enables multiplexed analysis of biomolecular and cellular libraries, potentially resulting in hundreds to thousands fold reduction in the number of samples needed for a study. Current fluorescent cell barcoding systems suffer from a number of drawbacks (**Table 1.1**). Some barcoding systems are single-use, meaning that barcodes are not genetically encoded and cells have to be barcoded individually before each experiment [17]–[23]. Other fluorescent cell barcoding systems have reported toxicity [24]–[26], and some methods rely on microscopy, which is lower throughput than flow cytometry [25], [27]. Existing

fluorescent cell barcoding systems have a limited number of spectrally distinct barcodes, on the order of 10 [19]–[21], [24], [26]–[28] to 100 [17], [18], [23], [25].

Table 1.1: Drawbacks of existing fluorescent barcoding systems

Barcoding Method	Max # Barcodes	Single Use	Toxicity	Analysis Method
Zinc Finger oligonucleotide barcoding[2]	6			Microscopy
Fluorescent protein barcoding[3]	41		X	Flow Cytometry
Organelle and fluorescent protein barcoding[4]	64		X	Microscopy
Dye barcoding[1,5]	96	X		Flow Cytometry
Epitope tag barcoding (this work)	**>980**			Flow Cytometry

1. Krutzik, P.O. and Nolan, G.P. (2006) *Nature Methods*
2. Mali, P. *et. al.* (2013) *Nature Methods*
3. Mohome, M. *et. al.* (2017) *Molecular Therapy*
4. Chen, R. *et. al.* (2015) *ACS Synthetic Biology*
5. Mattheakis, L.C. *et. al.* (2004) *Analytical Biochemistry*

The primary goal of this work is to engineer an improved fluorescent barcoding system for massively-parallel analysis of biomolecular and cellular libraries, potentially enabling multiplexed experimentation and analysis of thousands of different biological molecules in a single sample. In the first part of this work we discuss our design for a genetically-encoded fluorescent barcoding system. Fluorescent barcodes are produced upon immunolabeling of cell-expressed, engineered protein scaffolds, which are composed of different epitope tags connected by flexible linkers. Out of this work, a general method for exponential expansion of tandem DNA sequences was developed. Also, a tangential application using long tandem repeating epitope tags for improved flow cytometric immunodetection of endogenous low abundance proteins in single-cells is described. In addition, the instability of plasmids containing tandem nucleotide repeat regions in *E. coli* is explored using multiple methods including deep sequencing.

In the second part of this work, we discuss the development of a multi-color fluorescent barcoding system for high-throughput, single-cell analysis of biomolecular

and cellular libraries. Specifically, we present an approach to create up to four distinct intensities using a single fluorophore, greatly enhancing the number of possible barcodes over previously established binary schemes. A 190-member barcode library was created by subcloning six-different types of epitope tags differing in repeat number, and the barcode library was characterized and normalized FACS. In addition, the effects of constitutive and inducible promoters on barcode expression and relative cell growth rates are explored.

In the third part of this work, we discuss the expansion of the fluorescent barcoding system to over 1,100 members by incorporation of 5 addition epitope tags. Specifically, 18 plasmids encoding combinations of 5 different epitope tags were created using homologous recombination and overlap PCR. Barcode expression was validated by flow cytometry and clones were isolated by FACS. Then, the barcode system was expanded to create 11-epitope tag combinations by subcloning the 18 5-eptiope tag plasmids with the 190-plasmid library containing combinations of 6 additional epitope tags. We estimated that the barcode libraries contain a total of ~1,100-1,500 unique barcodes using a software package that was developed for rapid identification and quantification of barcoded populations from flow cytometry data.

In the last part of this work, we illustrate the utility of fluorescent barcoding for multiplexed single-cell analysis of biomolecular and cellular libraries. In one application, we find that barcode expression does not affect the apparent binding affinity of an α-prion antibody for recombinant prion protein, suggesting fluorescent barcodes can be used for multiplexed analysis of biomolecular libraries including protein-protein interaction studies. In a second application, we simultaneously examine the dynamic, single-cell expression profiles of endogenous yeast GFP fusion

proteins in different environments using fluorescent barcodes. This study uncovered interesting responses to fluctuating environments, including bimodal expression profiles and changes in protein expression abundance and variation, which may suggest that cells employ bet-hedging in order to adapt more easily to sudden environmental fluctuations. Overall, we have engineered an improved fluorescent barcoding system with greater than 10-fold more barcodes than existing systems, and demonstrate its use for multiplexed single-cell analysis of bimolecular and cellular libraries.

1.2 Design and Application of Previous Fluorescent Cell Barcoding Systems

Cellular fluorescent barcoding is an emerging technology that is used to multiplex samples for single cell-based assays or identify cells in mixed populations. An advantage of cellular barcoding is that it enables high-throughput single-cell analysis, is compatible with standard methods such as flow cytometry and microscopy, and does not require specialized robotics or microarray technologies. Fluorescent cell barcoding has demonstrated utility for many applications including cell signaling studies and drug screens, protein-protein interaction screens, and cancer and stem cell tracking *in vitro* and *in vivo.*

The first example of cellular fluorescent barcoding was first published in 2004, and used binary combinations of 5 different colors of fluorescent quantum dots to create 10 distinct barcodes that were introduced into cells via peptide-mediated delivery [19]. This method is advantageous because it can be used with microscopy or flow cytometry, but is not genetically encoded so cell samples have to be barcoded in multi-well plates and then mixed together for analysis. Also, three barcodes were used in a cellular assay to identify CHO cell lines expressing different GPCRs and to

measure changes in the cells' agonist-induced calcium levels [19]. Recently, another fluorescent nanoparticle based barcoding system used three colors and up to six intensity levels to create 20 nanoparticle barcodes [22]. These nanoparticles were loaded into cells via endocytosis and used for microscopy based tracking.

In 2006, a fluorescent-dye based barcoding method was developed which used different combinations and concentrations of three different fluorescent dyes to create 96 barcodes with up to four florescence intensities [18]. This method is advantageous because it permits multiplexing up to ~100-fold, but it is not genetically-encoded so cells have to be stained in multi-well plates before analysis, and it is not compatible with live cells. The dye-based fluorescent barcoding system was used to screen small molecule phosphatase and kinase inhibitors in a single cell-based assay. In addition, dye barcoding was applied to study the heterogeneous signaling response of different types of mouse primary cells to varying cytokine concentrations.

A related mass cytometry barcoding method used binary combinations of seven lanthanide metals to barcode 96 cells by surface-linked chemical conjugation [17]. Advantages of this method include the ability to measure ~40 parameters in a single-cell and the ability to create ~100 barcodes. However, mass barcodes are not genetically encoded so cells have to be barcoded individually before mixing and analysis. Mass barcoding also requires a mass cytometer, which are not commonly available. Mass barcodes were used to multiplex the analysis of peripheral blood mononuclear cells (PBMCs) to twelve different cytokine stimuli during eight different time points, as well as the response of PBMCs from eight different donors to 12 stimuli. In addition, the signaling response of PBMCs to kinase inhibitors was characterized using an eight-point dilution and 12 different stimulus conditions, as

well as 14 phospho-antibodies to characterize signaling pathway response and 10 antibodies specific for cell surface markers to identify different cell types.

Recently, fluorescent protein based barcoding systems have been developed by multiple groups independently. One group used three different fluorescence proteins to create 12 barcodes, and applied four barcodes in a cell based assay to determine the activity of cells expressing HIV-1 protease variants via an eGFP reporter [28]. Another study used combinations of up to three fluorescent proteins and six colors to create 41 fluorescent protein barcodes [26]. They tracked proliferation rates of 21 barcoded clones derived from a mouse glioblastoma cell line *in vitro* and in a mouse model. A third group created 26 total barcodes using three fluorescence proteins expressed under two different translational control elements to create off, low, and high intensities for a fluorescent protein [24]. However, a 10 to 20-fold decrease in cell viability was observed when combinations of two or more fluorescent proteins were transduced. Six fluorescent barcodes were applied to track clonal growth *in vitro* in response to different microRNAs, as well as the *in vivo* growth of mouse hematopoetic stem cells and human cord blood cells. Overall, fluorescent protein based barcoding systems are advantageous because they can be easily used in multiple cell types, and are compatible with flow cytometry and microscopy, but have limited applications due to their relatively small library sizes and in some cases reported toxicity issues when multiple fluorescent proteins are expressed in the same cell.

Another fluorescent protein based barcoding system used color and protein localization to identify cells [25]. 64 barcodes were created using four different fluorescent proteins and four localizations. The fluorescent barcoding system was applied in a fluorescent yeast two hybrid assay to screen for protein-protein

interactions between eight coiled-coil leucine zipper proteins. Specifically, bait proteins were tagged with a peroxisome targeting sequence and the prey proteins were fused to RFP. Absence of an interaction was indicated by diffuse red fluorescence, while presence of an interaction was indicated by red puncta, and 10-20 cells were used to calculate an interaction score. A success of this barcoding system was the ability to create more than 50 barcodes and demonstration of fluorescent barcoding for multiplex protein-protein interaction screens. Disadvantages include the use of lower-throughput microscopy resulting in low cell counts and false positive interactions, as well as toxicity issues when pairs of fluorescent proteins were targeted to certain organelles within cells and when certain combinations of coiled-coil proteins were expressed.

A unique approach to fluorescent cell barcoding is the use of cell surface displayed zinc finger domains that hybridize to different fluorescently labeled oligonucleotides [27]. Specifically, six cells were barcoded by expression of a unique zinc finger domain and sequential labeling with two different fluorescently labeled oligonucleotides. Advantages of this method include that is it genetically-encoded and can be used for multiplexed experimentation and analysis, whereas drawbacks include the dependence on microscopy and adherent cell types, as well as limited library sizes. The zinc finger barcodes were used for in applications for cellular immunopurification and increased transduction efficiency using oligonucleotide tagged lentivirus.

1.3 Single-Cell Analysis and Protein Expression Noise

Genetically identical cells can exhibit phenotypic heterogeneity, which is due to noise in gene and protein expression [5], [29]. Noise originates from stochastic biochemical processes involving small numbers of molecules, including random birth

and death of mRNA and protein molecules and promoter on/off transitions due to stochastic transcription factor binding [4], [30]–[32]. For example, a study examining single-cell transcription events in *E. coli* using a microfluidics platform found that protein expression occurs in stochastic bursts, and that the variation in burst size and frequency contributed to deviations in gene expression [33]. This phenomenon has been observed by others [34], and was quantified using a Gamma distribution model to describe gene and protein expression heterogeneity [35].

Other sources of noise include cell-cycle differences and unequal partitioning of cellular components during cell division [4], [36]. For example, 900 yeast promoters were assessed in different environmental conditions using single-cell reporter protein measurements. Gene expression noise was found to be higher in more nutrient poor conditions, and varied significantly due to cell-cycle differences [36]. In addition, gene and protein expression noise was observed to vary significantly with chromosomal location due to differences in chromatin conformations and histone modifications [37], [38]. For example, a study examining GFP expression levels at 500 different genomic locations found that chromosomal position affects protein expression levels in yeast up to 15-fold and noise up to 20-fold.

Additionally, pathway specific feedback loops have been studied as an additional source of noise. Studies have used both natural and synthetic gene regulatory networks to understand phenotypic heterogeneity. For example, the *E. coli* lac operon exhibited bistable state behavior at intermediate induction conditions leading to two phenotypic states [39]. A study examining the mulitstability of the galactose signaling network in *S. cerevisiae* found that positive and negative feedback loops contributed to multiple stable states and cellular memory [40]. Moreover,

different pathway or gene-specific factors may influence gene and protein expression noise. Studies have found that essential genes, including those involved in proteasome and protein synthesis [32], [41], exhibit lower noise than others [42], while stress related genes tended to be noisier on average.

Cellular heterogeneity and gene expression noise can confer adaptive advantages when cells encounter an environmental change, and phenotypic heterogeneity has been observed in both isogenic and genetically diverse cells including *E. coli*, yeast, and mammalian cancer cells. In *E. coli*, two coexisting cell states, namely expression of low and high amounts of membrane lactose permeases, were observed at induction intermediate concentrations using the lac operon [43]. It was found that infrequent disassociation of the lac repressor protein was responsible for bursts of protein expression resulting in the lactose metabolizing phenotype.

Studies in yeast have found that cellular heterogeneity in protein expression can confer adaptive advantages to fluctuating or adverse environments [44]. A subset of yeast proteins exhibited bimodal expression patterns under nitrogen starvation, oxidative or reducing environments. Upon further investigation, it was found that the high expressing phenotype conferred a growth advantage over short starvation time scales while the low expressing phenotype was more advantageous over long times [10]. In another study, yeast cells were engineered to have either slow or fast transitions from two phenotypic states as a result of stochastic gene expression [11]. Under fast environmental fluctuations, frequent switchers had a growth advantage while the converse was true when environmental changes occurred infrequently. Taken together, these results suggest microbial cells can exhibit multiple phenotypic

states characterized by noise in gene and protein expression, which can confer a fitness advantage in certain circumstances.

Cellular heterogeneity among cancer cell populations has been shown to confer drug resistance. For example, a subpopulation of cancer cells were observed to be drug-tolerant, exhibiting more than 100-fold reduced drug sensitivity [45]. It was found that this cell subset had a modified chromatin state with higher expression of the histone demethylase KDMA5A, suggesting epigenetic mechanisms may underlie drug resistance in this case. In another study, the proteome dynamics of a lung carcinoma cell line in response to an anti-cancer drug was examined [46]. The protein expression profiles of 1,200 endogenous fluorescent protein fusion clones were quantified, and it was found that a subset of proteins exhibited bimodal expression profiles that conferred improved drug resistance. Overall, cellular heterogeneity has been observed in many cell types and has many potential underlying sources including gene and protein expression noise. Single-cell analysis methods are necessary to fully capture and understand cellular heterogeneity.

Chapter 2

MATERIALS AND METHODS

2.1 Yeast Barcode Expression Plasmid Construction

The yeast expression vector pBC1 was derived from two plasmids: p416-25Q-GPD was purchased from Addgene and pUC57-SS-cmyc-AGalpha1 was synthesized by Life Technologies. pBC1 was constructed from the p416-25Q-GPD vector and pUC57-SS-cmyc-AGalpha1 was created by restriction digest with XbaI and XhoI restriction enzymes and ligation of p416 backbone and SS-cmyc-AGalpha1 insert. pBC1 contains the C-terminal domain of the alpha-agglutinin mating protein (AGalpha1) downstream, a secretion signal (SS), a cmyc tag, and a constitutive GPD promoter upstream of the repeat region.

The pBC1-GAL plasmid was created by changing the GPD promoter for the inducible GAL promoter. The GAL inducible promoter was obtained from the pCTCON2 plasmid. The pBC2 plasmid was adapted with additional G4S3 linkers and an AfeI cloning site for subcloning of protein fusions of interest to barcodes using a commercially synthesized oligonucleotide. GFP epitope tag integration plasmids were created by restriction digestion of the pRSII40X plasmid series (Addgene) with SacI and KpnI. A synthesized oligonucleotide containing N-terminal and C-terminal regions of GFP homology, a restriction enzyme site for linearization, and a multiple cloning site for the epitope repeats was subcloned into the pRSII plasmid backbone.

Lastly, epitope tag repeats were subcloned from the pBC1 plasmids to the pRSII plasmids.

2.2 Creation of Tandem Repeating Epitope Tag Plasmids for Multiple Unique Intensity Barcodes

Epitope tag oligonucleotides were synthesized by Integrated DNA Technologies. Oligonucleotides contain a single copy of HA, HIS, FLAG, GLU-GLU, AU1, or HSV epitope tag followed by $(G_4S)_3$ and $(G_4S)_1$ linkers, as well as a PacI site upstream of the epitope tag, and an AsiSI site and a 3' unique restriction site flanking the $(G_4S)_1$ linker, such as XmaI for HIS. Each single epitope tag was subcloned into the pBC1 constitutive yeast expression vector, and epitope tag repeat number was expanded exponentially from 1 to 2, 4, 8, 16, and in some cases 32 and 64 repeats by iterative restriction digestion and ligation. To increase the number of repeats, pBC1 containing *n* epitope tag repeats was digested with PacI and 3' unique site restriction enzymes to create insert and separately digested with AsiSI and 3' unique site restriction enzymes to create an acceptor vector. PacI and AsiSI have compatible two-base pair overhangs that cannot be digested by either enzyme after ligation. The resulting construct contained 2^n epitope tag repeats, with each repeat separated by an undigestible 5'AsiSI/3'PacI site. The 3' unique restriction enzyme sites used to expand the number of repeats were BamHI, ClaI, XmaI, EcoRI, HindIII, and SalI for HA, HSV, HIS, AU1, Glu-Glu, and FLAG, respectively.

Epitope tag repeats were checked for the correct size by restriction digest. Plasmids were transformed into high-efficiency chemically competent NEB5α *E. coli* or NEB Stable *E. coli* (New England Biolabs), and colonies were selected on LB agar plates containing 100 μg/mL ampicillin. Sanger sequencing failed to provide an

accurate nucleotide sequence of greater than four repeats due to sequence redundancy. Ultimately, Pacbio SMRT next generation sequencing was used to determine the distribution of epitope tag repeat lengths.

2.3 Development of Multiple Color Fluorescent Barcodes by Subcloning Epitope Tag Repeat Combinations.

Each of the six epitope tags used initially are flanked on either side by a unique pair of restriction enzyme sites, allowing the creation of barcode plasmids with combinations of epitope tags with varying repeat lengths. For example, to construct a barcode plasmid containing 4HA and 1HSV, an acceptor vector containing pBC1-1HSV an an insert plasmid containing pBC1-4HA were digested with SpeI and BamHI restriction enzyme sites. After purification of the insert and vector DNA fragments, DNA was ligated in a 3:1 molar insert to vector ratio either overnight at 16°C or at room temperature for 1 hour, and transformed into NEB Stable or NEB 10ß cells using chemical transformation or electroporation. Single *E coli* colonies were picked from a selective agar plate grown overnight at 37°C and screened by restriction digest for the correct insert sizes.

Barcode plasmids were also confirmed by protein expression and cellular immunolabeling. Specifically, plasmids were transformed into the yeast strain BY4741 (MATa his3Δ0 leu2Δ0 met15Δ0 ura3Δ0) [47] by the standard lithium-acetate method [48] or by electroporation [49], and selected for on URA dropout agar plates for 3-4 days. Single colonies were grown at 30°C in selective SD glucose media to early or mid log phase, and if required transferred to SG galactose media to induce barcode expression for an additional 24 hours, immunolabeled, and analyzed by flow cytometry.

2.4 Fluorophore Antibody Conjugation and Multicolor Flow Cytometry Panel Development

Epitope tag antibodies were purchased from commercial sources (Abcam, Sigma Aldrich, Life Technologies, BioRad, BioLegend, Jackson Immunoresearch, EMD Millipore). Antibodies for HA, V5, E2, and AcV5 were purchased from Abcam. Secondary Alexa Fluor conjugate antibodies, quantum dot conjugate antibodies, and HIS antibody was purchased from Life Technologies. FLAG was purchased from Sigma Aldrich. AU5, AU1, and GLU antibodies were purchased from Biolegend. HSV and T7 antibodies were purchased from Millipore. Fluorophores were covalently conjugated to antibody lysine residues using either succinimidyl ester chemistry [50] for small molecule Alexa Fluor 488, 647, and 700 or Marina Blue dyes (Life Technologies), or Lightning-Link antibody conjugation kits (Innova Biosciences) for protein fluorophores including APC-Cy7, PE-TexasRed, PE-Cy5, PE-Cy5.5, PE-Cy7, and PerCP. For Alexa Fluor dyes, antibody conjugates were dialyzed overnight against PBS pH 7.4 and molar ratios of fluorophore to antibody were between 3:1 and 10:1 as calculated from spectrophotometry measurements. For lightning-link technology, molar ratios of fluorophore to antibody were estimated to be between 1:1 and 2:1 by spectrophotometry. In addition, in some experiments different antibody species were used to enhance the signal from some of the dimmer fluorophores. Specifically, rat α-FLAG and α-rat PerCP conjugate, human α-V5 with biotin α-human and streptavidin QDot 525 conjugate, chicken α-cmyc and α-chicken AF488 or AF647 conjugate, and rabbit α-HIS and α-rabbit QDot705 conjugate.

2.5 Barcode Immunofluorescence and Flow Cytometry Analysis

A known number of yeast cells expressing barcodes were incubated in PBS pH 7.4 + 0.1% BSA containing at least a 10-fold stoichiometric excess of α-epitope tag

antibodies for 1 hour at room temperature with mixing after 30 minutes. The number of proteins per cell was estimated by us and others [51] to be 50,000-100,000 copies per cell. Cells were washed twice with PBS+BSA and if necessary incubated with fluorophore-conjugated secondary antibodies for 30 minutes on ice. Samples were washed twice in PBS+BSA and analyzed using either an Accuri C6 flow cytometer or a FACSAriaII cell sorter.

Antibodies were titrated to determine the binding affinity, and in most cases a saturating condition was used for immunolabeling (typically 10-100nM). For 13-color flow cytometry, however, compensation and spillover prohibited saturating conditions to be used for all fluorophores. It was experimentally determined that 1-5 nM cmyc with 35 nM α-chicken AF647 and 10 nM HA-PE gave the best compromise between brightness and spillover.

Flowjo software was used to calculate compensation matrices and analyze data. To calculate compensation, cell samples labeled with a single fluorophore or fluorophore conjugated beads were used. For some experiments, fluorescence minus one controls were used to determine negative and positive populations. Yeast clones expressing single barcodes labeled with cmyc and one additional epitope tag were used to distinguish positive populations, quantify the percent of barcodes captured and estimate the error between barcodes.

To manually analyze a flow cytometry data set of barcode mixtures, first a gate was drawn on the cell population in order to exclude debris and minimize forward and side scatter differences between cells. Then, a conservative gate was drawn on the cmyc channel for normalization of protein expression. Barcoded subpopulations were visualized on 2-D scatter plots by the fluorescence intensity of one epitope tag versus

the normalization fluorescence, and gates were drawn around the subpopulations as to minimize overlap between them and remove any cells with ambiguous barcodes.

2.6 Expansion of Barcode Library by Random Homologous Recombination of Epitope Tags and Fluorescence Activated Cell Sorting (FACS)

In order to generate additional barcodes, we developed a strategy to create new epitope tag combinations in one reaction using random recombination of the flexible glycine-serine homologous linkers that surround epitope tags. Specifically, nine additional epitope tags, namely S tag, AcV5, AU5, StrepTagII, V5, E2, T7, VSVG, and E tag, flanked by $(G_4S)_3$ linkers and two sets of restriction enzymes, were synthesized by Genewiz. Three approaches were used to create DNA plasmids encoding epitope tag combinations: homologous recombination with full restriction digest, partial restriction digest, or overlap PCR. For the restriction digest and homologous recombination approach, inserts containing epitope tags surrounded by linkers were generated by digesting the synthesized plasmid with a log 10 titration of NheI or SphI restriction enzymes. The insert and vector were purified and transformed into yeast. A third strategy used fully digested epitope-linker DNAs as primers in an overlap PCR reaction to create new epitope tag combinations.

The three yeast barcode libraries were analyzed for barcode combinations using flow cytometry, and each unique combination was enriched and isolated by FACS. After sorting, yeast clones expressing barcodes were selected on agar plates and tested for barcode expression using flow cytometry. After barcoded yeast clones were confirmed, plasmids were rescued by zymoprep and transformation into NEB Stable *E. coli*. Barcode plasmids were also confirmed for the correct epitope tag combination by Sanger sequencing.

2.7 Construction of Libraries Containing Thousands of Unique Barcodes

To construct a large and diverse barcode plasmid library, epitope tags with repeat lengths that generate unique fluorescent signatures when expressed and immunolabeled in yeast were combined in three sequential reactions to generate up to 216 barcodes. In the first round, plasmids containing 1 and 4 HA were subcloned as mentioned above in a one pot reaction with 1 and 4 HSV to create 8 unique barcodes. 4HIS and 1 and 4 AU1, and 4 GLU-GLU and 1FLAG were combined in the same manner to create 5 and 3 unique barcodes respectively. Then, the HA-HSV and HIS-AU1 libraries were crossed using subcloning to make 54 unique barcodes. Finally, the HA-HSV-HIS-AU1 library was crossed with the GLU-FLAG library to make up to 216 unique barcode combinations using 6 epitope tags.

To further expand the number of barcodes, we constructed 18 barcode libraries using a unique plasmid from the second round of barcode creation, which contain combinations of AcV5, AU5, V5, E2, and T7 epitope tags, as the acceptor vector and the library of 216 barcodes as the insert. Specifically, DNA was digested with SpeI and XhoI restriction enzyme sites to linearize the vector and excise DNA encoding the barcode and AGAlpha1 protein. Additionally, the inserts were digested with a cocktail of restriction enzymes, including NdeI, BstNI, AclI, EciI, and DraI, to digest the mini-plasmids. Mini-plasmids are non-full-length plasmids composed of backbone DNA fragments with partially or fully deleted barcode regions that arose from plasmid instability in *E. coli* due to repeating DNA regions.

DNA inserts and vectors were gel purified, ligated overnight at 16°C, purified, transformed into NEB 10ß cells, and plated at three different dilutions to determine the library size. After overnight growth on agar plates, *E coli* colonies were miniprepped to obtain plasmid DNA. Barcode library DNA was transformed into yeast using the

high efficiency lithium acetate method, and yeast libraries were grown up in selective media and dilutions were plated on selective agar to determine library size. Yeast libraries were grown in glucose liquid media to log phase and induced in galactose media for 24 hours at 30°C prior to immunolabeling and flow cytometry.

2.8 Construction and Analysis of Barcode Pacbio SMRT Library

Barcodes contain tandem GC-rich DNA sequences, prohibiting Sanger sequencing for constructs containing more than 4 repeats. Therefore, we used Pacbio SMRT sequencing to determine the distribution of epitope tag repeat lengths and barcode combinations in the library. Approximately 100 plasmids containing different repeat lengths and combinations of HA, HSV, HIS, GLU-GLU, AU1, and HSV epitope tags were prepared for Pacbio sequencing by restriction digest with XbaI and XhoI to excise the DNA fragment encoding the barcode ORF. BluePippin was used to remove most of the backbone fragments from the sample, and the remaining DNA inserts were adapted for SMRT sequencing. After sequencing, raw data was filtered using a circular consensus sequence (CCS) cutoff of 3 and a quality score of 0.9. The 'reads of insert' protocol from SMRT analysis portal was used to generate CCS reads.

We created a method to analyze the distribution of repeat lengths in the Pacbio library. Specifically, we used the LALIGN algorithm [52] implemented by W.R. Pearson at UVA, which finds internal, non-intersecting duplications in nucleotide sequences. The default gap open penalty of -12 and gap extension penalty of -4 were used to compute expectation values. An expectation threshold value of 1e-4 was used to filter out incorrect alignments. Additionally, Python scripts were written to further process the LALIGN data. These scripts determined the epitope tag repeat lengths and

their frequency, the frequency of particular barcode combinations, and the barcode combination for each Pacbio sequence read.

2.9 Detection and Quantification of Low-Abundance, Endogenous Repeat Fusion Proteins in Cells

GFP epitope tag repeat plasmids were constructed as described above. Plasmids were linearized with XhoI, purified, and transformed into yeast GFP fusion strains using the lithium acetate ssDNA method. Single clones were isolated after 3-4 days of growth on selective URA dropout plates, and FLAG integration and GFP knockout was confirmed by immunolabeling and flow cytometry. Specifically, cells were fixed for two hours in 4% formaldehyde. After two washes, cell walls were digested by addition of 50 μg/mL zymolyase in PBS for 1 hour at 30°C. Spheroplasts were permeabilized by incubation with 0.25% Triton-X in PBS for 30 minutes with rotation at room temperature. Cells were washed 3 times in PBS and incubated with 100 μg/mL RNase A for 15 minutes at 37°C. Then, cells were incubated with 10nM α-FLAG in at least 10-fold stoichiometric excess for 2 hours at room temperature, washed three times with PBS supplemented with 4% BSA and incubated with 35 nM Alexa Fluor 647 conjugated α-mouse antibody for 1 hour at room temperature and SYTOX Green nucleic acid stain or propidium iodide for 10 minutes. After three washes with PBS and 4% BSA, cells were analyzed on an Accuri C6 flow cytometer. Cells were also analyzed using a Zeiss LSM 800 confocal microscope equipped with a 63x oil lens.

2.10 Expression and Purification of Recombinant Mouse Prion Protein

The NEB T7 *E. coli* strain containing the pET11a-MoPrP 23-230 plasmid was grown overnight in a 250 mL starter culture from a single colony. The starter culture

was inoculated into a 30L New Brunswick Scientific BioFlo 4500 bioreactor containing 20L of TB media. After cells reached an OD_{600} of 1, IPTG was used to induce protein expression for 4 hours. The protein was purified as previously described by isolation and homogenization of inclusion bodies, size exclusion chromatography, oxidation for two weeks at 4°C, reverse phase HPLC, and lyophilization [53]. Recombinant prion protein purity was confirmed by SDS-Page, Coomassie and silver stain. For fluorescent labeling, a stoichiometric excess of Alexa Fluor 647 NHS ester was incubated in 0.1M sodium bicarbonate buffer, pH 8 with lyophilized prion protein. Labeled, precipitated PrP was recovered by centrifugation and solubilized in refolding buffer consisting of 4M Urea, 0.25 mM oxidized glutathione (GSSG), and 0.1M Tris (pH 8) for 90 min to create the alpha helical form of the prion protein [54]. For labeling experiments, prion protein in refolding buffer was rapidly diluted into PBS pH 7.4 at 100 nM or lower to avoid precipitation.

2.11 Creation and Assessment of Barcoded ICSM18 2.6.1 scFv

A plasmid containing pCTCON2-ICSM18 2.6.1 was previously constructed by Kyle Doolan. The ICSM18 2.6.1 sequence was isolated by PCR using primers to amplify the gene and change the restriction enzyme sites used for subcloning. The DNA fragment was cloned into the pBC2 plasmid and two libraries containing a mixture of barcode plasmids. Cells were grown up in SD media and induced in SG media for 24-48 hours at 20°C or 30°C. For immunolabeling, Alexa Fluor 647 labeled alpha helical prion protein was titrated with cells displaying ICSM18 2.6.1 or barcoded ICSM18 2.6.1 in PBS pH 7.4. Cells were immunolabeled with cmyc and α-chicken Alexa Fluor 488 to normalize for differences in protein expression level, and barcode antibodies if needed, as described above.

2.12 Construction of Barcoded Yeast GFP Fusion Strains

Yeast clones from the yeast GFP fusion collection [55] were unfrozen from -80°C stocks and grown overnight in 96-well plates with shaking at 800 rpm using an orbital microplate shaker (Benchmark Scientific) at 30°C. Cells were replicated onto SD-HIS plates using a prong replicator and grown overnight. 100ng each of either a single barcode plasmid or barcode plasmid libraries were transformed into each strain as previously described [48]. After 3-4 days of selection on SD-His-Ura plates, single clones were picked to select yeast clones expressing a single barcode. Single strains were preserved in 96-well plates.

2.13 Identification of Unique Barcoded Yeast GFP Fusion Strains and Environmental Perturbations

Yeast GFP fusion strains expressing a unique and known combination of T7, V5, AU5, AcV5, and E2 epitope tags were pooled in mixtures to screen for an unknown combination of HA, HSV, HIS, AU1, GLU-GLU, and FLAG epitope tags. For identification of unique barcodes, mixtures of barcoded yeast GFP fusion strains were grown up in SD to log phase and induced in SG media. In some cases, cells were fixed in 1-4% formaldehyde for 10 minutes and permeabilized in 100% methanol to lessen cellular autofluorescence. Cells expressing barcodes were immunolabled as described above and analyzed on a FACSAriaII cell sorter.

For perturbation experiments, after induction of barcode expression cells were exposed to an environmental change (e.g. ethanol, oxidative stress, heat shock) at a range of concentrations and fixed at various time points, usually less than two hours. For immunostaining, cells were permeabilized, immunolabeled, and analyzed as described previously.

2.14 Conversion of Arbitrary Fluorescence to Protein Abundance

Recombinant eGFP (recGFP) produced in E. coli and purified by ion exchange chromatography was a gift from Dr. Wilfred Chen at the University of Delaware. recGFP was assessed by Coomassie stain as 90% pure by ImageJ. recGFP concentration was estimated to be 0.67 ± 0.08 mg/mL by BCA assay (Thermo Scientific Pierce). Quantitative Western blotting was used to determine the abundance of GFP in the TDH3-GFP yeast strain. Briefly, 40, 50, 60, 70, 80, 90, and 100 ng of recGFP and lysate from a known number TDH3-GFP cells were loaded onto an SDS-PAGE gel and transferred onto nitrocellulose membrane. α-GFP antibody (Life Technologies) and α-mouse HRP conjugated antibody (Life Technologies) were used to detect chemiluminescence. GFP intensities were quantified using ImageJ (Schneider et al., 2012), and a standard curve of chemiluminescence versus amount of recGFP was plotted. The abundance of GFP in the TDH3-GFP fusion strain was calculated from the standard curve as 2.7 million molecules per cell.

A relationship between arbitrary GFP fluorescence and GFP protein abundance was determined by linear regression after analyzing the TDH3-GFP fusion strain by flow cytometry. To correlate FLAG signal with protein abundance, eGFP with 16 FLAG was expressed as a fusion to AGalpha1, and cells were immunolabeled with α-FLAG and Alexa Fluor 647 conjugated α-mouse antibodies and analyzed by flow cytometry.

Chapter 3

DEVELOPMENT OF ONE-COLOR EPITOPE TAG REPEAT BARCODES AND THEIR USE FOR DETECTION OF ENDOGENOUS, LOW-ABUNDANCE PROTEINS IN SINGLE-CELLS

3.1 Introduction

In this chapter, fluorescent cellular barcode design considerations are discussed, including the desire for a high degree of barcode diversity, the need for high-throughput measurement capability, and the advantages of a genetically-encoded barcoding system that can be reused for multiple experiments. A new method for iterative exponential expansion of tandem repeating DNA sequences, and its use for the creation of single-color barcodes composed of repeating epitope tags is presented. Furthermore, evidence supporting instability of tandem epitope tags in *E. coli* is presented, including DNA gel electrophoresis analysis, protein expression analysis by flow cytometry, and next-generation sequencing analysis of tandem repeating barcode DNA. Also, we show that the flow cytometric detection and quantification of endogenous low-abundance proteins in single-cells is enabled by the use of protein fusions to long epitope tag repeats.

3.1.1 Fluorophores and Protein Detection Methods

Fluorophores are molecules that produce fluorescence, or in other words emit light at a longer wavelength after absorbing light. Fluorophores are used widely to detect proteins, nucleic acids, organelles, and even cells. The three major types of fluorophores are: small organic dyes, fluorescent proteins, and quantum dots [56]. The

two most widely used methods for protein detection using a fluorescent reporter are immunolabeling with fluorophore-conjugated antibodies or genetic fusions with fluorescent proteins. Proteins are typically studied in cells and tissues using fluorescence microscopy or flow cytometry [56].

Fluorescent protein fusions are advantageous because they enable direct detection via genetic fusion to a protein of interest, and can be used to visualize proteins in living cells. Disadvantages of fluorescence proteins include the limited number of spectrally distinct proteins [57] and the potential impact on protein function due to their relatively large size. Moreover, fluorescent barcoding systems that are fluorescent protein based have been unable to achieve large numbers (>100) of barcodes due to these limitations [24]–[26].

Immunolabeling has distinct advantages over fluorescence protein fusions. Immunolabeling is compatible with a wide range (>40) of fluorophores, and therefore can be used to detect a greater variety of proteins in a single cell [58], and it is less likely to impact protein function as it is an indirect detection method. However, immunolabeling is often limited to surface proteins or requires cell fixation and permeabilization for intracellular proteins. Immunolabeling also requires high affinity, high specificity antibodies [56]. Previous barcoding systems based on immunolabeling have been successful in achieving ~100 barcodes [17].

Epitope tags are short sequences of amino acids, typically 6-15 residues in length, that can be genetically fused to proteins for detection or purification [59]. Epitope tags protein fusions are created by genetic fusion to the beginning or end of a DNA sequence encoding the protein of interest. Once the tagged protein is expressed, it can be detected by immunostaining with a specific α-epitope tag antibody.

Additionally, epitope tags are advantageous because they have a minimal impact on protein function and structure due to their small size [60], and there are dozens of commercially available epitope tags and corresponding antibodies. The epitope tags and their amino acid sequences used in this work are listed in **Table 3.1**.

Table 3.1: Epitope tags and amino acid sequences used in this work.

Epitope	**Sequence**
c-Myc	EQKLIEEDL
HA	YPYDVPDYA
HIS	HHHHHH
AU1	DTYRYI
Glu-Glu	EYMPME
FLAG	DYKDDDDK
HSV	QPELAPEDPED
T7	MASMTGGQQMG
E2 tag	SSTSSDFRDR
V5	GKPIPNPLLGLDST
AU5	TDFYLK
AcV5	SWKDASGWS
E-tag	GAPVPYPDPLEPR
VSV-G	YTDIEMNRLGK
Strep Tag II	WSHPQFEK
S tag	KETAAAKFERQHMDS

3.1.2 Yeast Surface Display

Yeast surface display is a powerful method that is used to engineer proteins with desirable properties. For example, yeast surface display has been used to engineer a fibronectin domain with picomolar binding affinity [61], as well as proteins with enhanced thermal stability and soluble secretion efficiency [62]. To perform yeast surface display, a protein of interest is typically genetically fused to one of two yeast cell mating proteins, namely the C-terminal portion of the α-agglutinin protein or the a-agglutinin (AGA) protein [63]. Notably, the α-agglutinin system requires

transformation of a single plasmid encoding the protein of interest fused to alpha-agglutinin. In contrast, the a-agglutinin display system requires both a plasmid and genetic integration. Using this method, the protein to be displayed is genetically fused to the AGA2 protein and expressed from a plasmid, and the yeast strain for protein expression is genetically modified by chromosomal integration of a gene encoding the AGA1 protein [64].

3.1.3 Methods for High-Throughput Cell Biology

The two major tools used for high-throughput cell biology are flow cytometry and fluorescence microscopy. Flow cytometry is a powerful tool for high-throughput, single-cell analysis and has led to many novel discoveries in cell biology. Flow cytometry is routinely used in immunology to identify cell subtypes and rare cell phenotypes, such as in the identification of rare HIV specific T cells [65]. In flow cytometry, a sample of cells containing fluorophores, that have been introduced by genetic fusions, staining, immunolabeling, are introduced into a rapid (>10,000 events per second) laminar flow stream. As the cells travel in the flow stream, they are focused into single file by hydrodynamic focusing. Then, single cells are interrogated at the flow cell by a series of lasers, causing the fluorophores in the cell to be excited and subsequently emit fluorescence which is captured by photodetector arrays equipped with spectral filters [66]. Flow cytometry is particularly powerful because it can measure multiple biomolecules simultaneously in a single cell (nucleic acids, proteins, ions, lipids), with each entity to be detected assigned a different, spectrally distinct fluorophore. Currently, state of the art flow cytometers can measure up to 20 parameters in a single cell [58].

Automated fluorescence microscopy is another tool that is frequently used for high-throughput cell biology. For example, it has been used for many studies examining yeast proteomics in response to environmental fluctuations [10], [67], [68]. These systems can be used to quantify fluorescence by automated imaging of cells cultured in multi-well plates and image processing using software to quantify pixel intensity [69]. Fluorescent barcoding systems that are fluorescence microscopy based are often lower throughput and measure fewer cells per barcode than flow cytometry, and can therefore have more inherent noise [25], [27].

3.1.4 Instability of Tandem Repeating DNA

Tandem DNA repeat sequences are present in bacterial and eukaryotic genomes and on circular plasmids. Changes in the length of repetitive DNA sequences are known to underlie a number of human genetic conditions including Huntington's disease, spinocellular ataxia, and mynotonic dystrophy [70]. In addition, deletion or changes to plasmid DNA harboring repeat regions have detrimental consequences including reduced plasmid yield and quality, and can hamper molecular cloning [71]. Repetitive nucleotide sequences are known to undergo expansion or deletion events via a variety of mechanisms including RecA mediated homologous recombination [72] and recA independent mechanisms such as replication slippage caused by the formation of secondary structure [73], sister chromosome exchange, and single-strand annealing [74]. The frequency of recombination in plasmids has been shown to be affected by the length of the repeat and spacing between repeats [75]. For unstable plasmid propagation, some have found that the genetic background of the *E. coli* strain can minimize the frequency of recombination events [71].

3.2 Design Considerations for an Improved Fluorescent Cell Barcoding System

To create distinct fluorescent barcodes, we created thousands of DNA plasmids encoding protein barcodes, which can be used to identify different cells in a heterogeneous mixture. Barcodes are composed of epitope tags, differing in both the type of tag and number of repeats, connected by flexible glycine-serine linkers (**Figure 3.1**). For convenience, we fused barcodes to the yeast surface alpha-agglutinin mating protein, although barcodes could also be fused to a different protein or expressed intracellularly. Different types of epitope tags are used to create barcodes with more fluorophores or 'colors', whereas different repeat numbers of an epitope tag are used to create spectrally distinct intensities of a particular color.

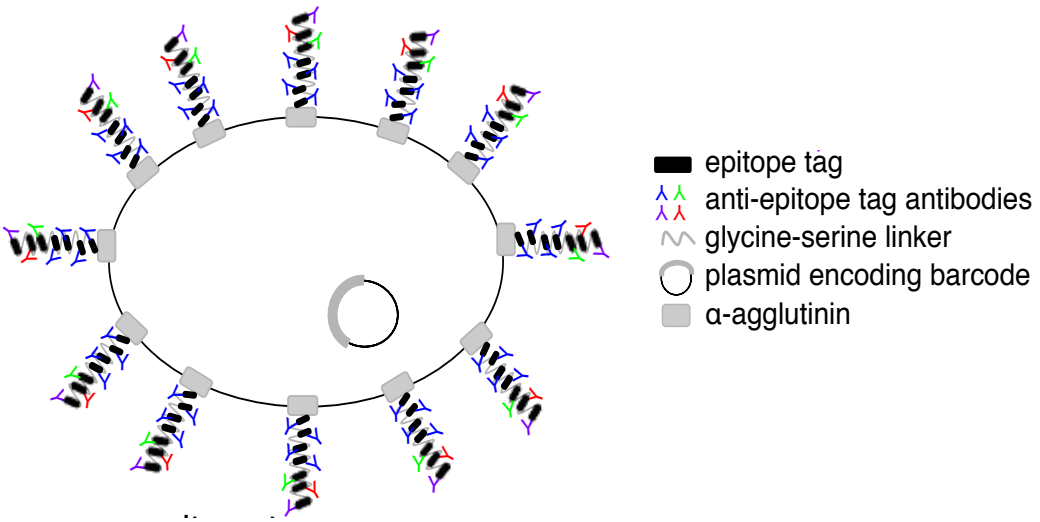


Figure 3.1: Fluorescent barcoding design. Fluorescent barcodes are composed of epitope tags connected by flexible linkers that produce spectrally distinct colors when expressed in cells and immunolabeled with fluorophore conjugated α-epitope tag antibodies. Barcodes are fused to the yeast surface protein alpha-agglutinin and are genetically encoded on a plasmid.

To use the barcoding system, first barcode plasmids are transformed into cells of interest. Then, the barcoded cells are pooled into a single tube, immunolabeled with fluorophore conjugated antibodies, and analyzed by flow cytometry (**Figure 3.2**). After analysis, different types of cells present in the heterogeneous sample are deconvolved by their fluorescent barcode's unique combination of colors and fluorescent intensities. Barcodes can be used to distinguish members of biomolecular and cellular libraries in a heterogeneous mixture. For example, barcodes can be used to identify different types of cells, cells expressing different fluorescent fusion proteins, cells with different genetic knockouts, and cells expressing different protein mutants. Additional variables that have a fluorescent readout can be multiplexed with fluorescent barcodes, including intracellular fluorescent protein fusions, protein-protein interaction assays, nucleic acid or lipid stains, cell-cycle analysis, ion indicator dyes, and live/dead cell staining.

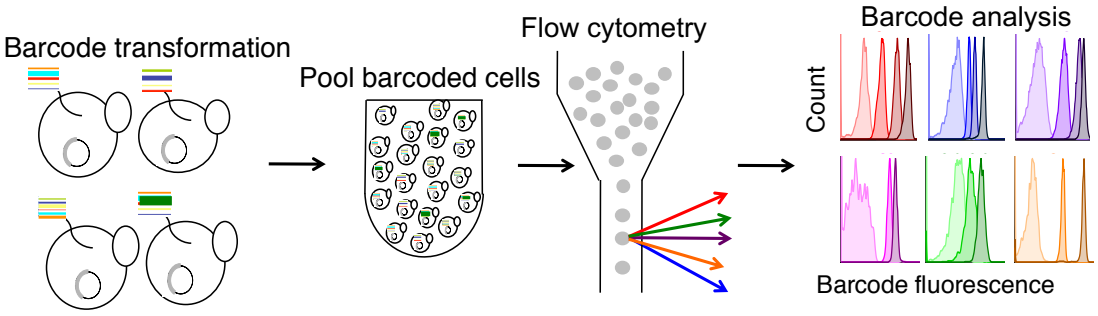


Figure 3.2: Fluorescent barcoding workflow. Plasmids encoding distinct fluorescent barcodes are transformed into cells of interest. Then, barcoded cells are pulled into a single tube, immunolabeled, and analyzed by flow cytometry. Finally, the flow cytometry data is analyzed to determine the fluorescent barcode associated with each cell.

3.3 Exponential Expansion of Epitope Tag DNA Sequences and Instability of Long Epitope Tag Repeats in *E. coli*

We (**Appendix A, C**) developed a method to construct tandem epitope tag repeats in order to create barcodes with distinct fluorescent intensities. This method uses iterative restriction digest and destructive ligation, and is generally applicable to any DNA sequence of interest. For example, a plasmid encoding a single HIS tag followed by $(G_4S)_3$ was digested with PacI and XmaI restriction enzymes to generate the HIS-$(G_4S)_3$ insert to be duplicated, and the same plasmid was separately digested with AsiSI and XmaI to generate an acceptor vector, which retained the sequence encoding HIS-$(G_4S)_3$. Ligation of the gel purified products resulted in duplication of sequence encoding the HIS-$(G_4S)_3$ as well as destruction of the AsiSI and PacI sites which had been cut prior to ligation, while one copy of each of these sites was retained on the new vector. The length of the insert was doubled upon each repetition of the process to generate plasmids encoding (HIS-$(G_4S)_3)_4$, (HIS-$(G_4S)_3)_{8,}$, and (HIS-$(G_4S)_3)_{16}$.

The same process was applied to generate analogous series of plasmids encoding tandem repeats of the HA, FLAG, AU1, GLU, and HSV epitope tags. Restriction enzyme digest was used to excise the repeat regions and DNA was run on an agarose gel to check for the correct size (**Figure 3.3**). Also, Sanger sequencing was used to confirm the repeat lengths of the barcode plasmids. However, plasmids with more than four repeats failed to sequence due to their repetitive structure and high GC content.

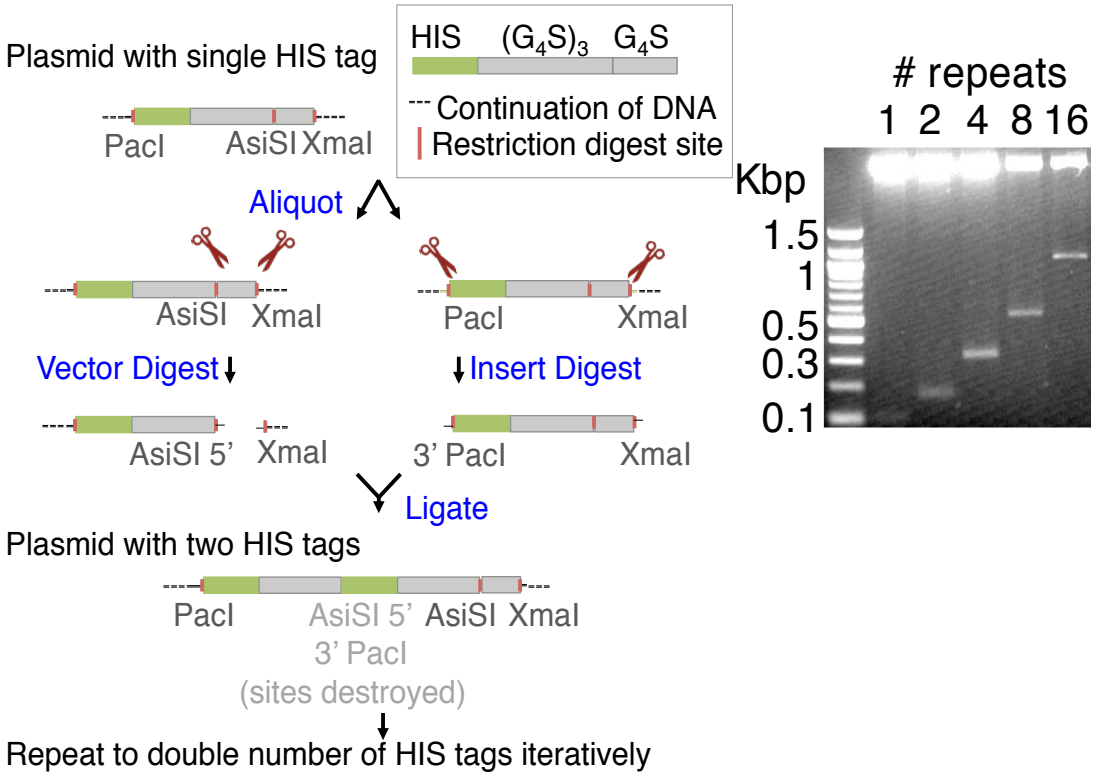


Figure 3.3: Exponential tandem expansion of epitope tag repeats. The left panel shows a general method for the exponential expansion of repeat sequences by destructive ligation of AsiSI and PacI restriction enzyme sites. The use of two restriction enzyme sites which are destroyed upon ligation, in conjunction with a third enzyme whose site is preserved, is key to iterative duplication of tandem sequences. The process was repeated to iteratively double the number of epitope tags up to sixteen repeats. The right panel shows an example of a DNA gel with plasmids digested to excise the repeat regions of differing lengths.

Occasionally, barcode plasmids recovered from *E. coli* colonies contained unexpected repeat lengths. For example, we observed a plasmid that appeared to contain ~6 repeats by restriction digest after attempted subcloning of a 16 repeat plasmid (**Figure 3.4**). Similarly, plasmids recovered after attempted subcloning of 32

and 64 repeat GLU constructs contained repeat lengths smaller than the starting plasmid. After encountering these difficulties, we hypothesized that plasmids with 16 or more repeats were unstable in *E. coli*. To test this hypothesis, we subcloned 1, 4, and 16 repeats of HA, HSV, HIS, AU1, GLU, and FLAG epitope tags that had been previously confirmed to be the correct size into the pBC1-GAL backbone vector. DNA plasmids were purified from *E. coli* in a library format and transformed into yeast cells.

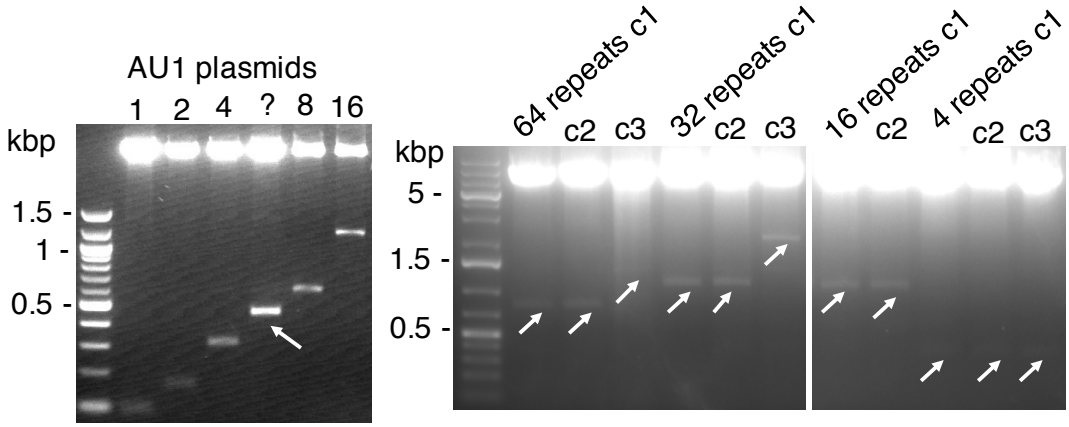


Figure 3.4: Instability of long epitope tag repeat plasmids. Repeat plasmids were subcloned using the expansion method and transformed into *E. coli*. Single colonies were picked and plasmids were purified, restriction digested to excise the repeat region, and run on an agarose gel. The left panel shows an unexpected AU1 plasmid of ~6 repeats in length. The right panel shows DNA repeat regions after attempted subcloning of 4, 16, 32, and 64 GLU repeats. Notably, all 64 repeat plasmids and two out of three of the 32 repeat plasmids have the incorrect size, whereas all of the 16 and 4 repeat plasmids are the correct size.

After barcode expression and immunolabeling, we observed that cells expressing 1 and 4 repeat barcodes formed single populations with an expected

increase in fluorescence with repeat length (**Figure 3.5**), suggesting that 1 and 4 repeat plasmids are stable in *E. coli*. However, immunolabeling of cells expressing the 16 repeat plasmids produced a heterogeneous mixture of fluorescence intensities, with no observed increase in fluorescence. This suggests that 16 repeat plasmids are unstable in *E. coli*.

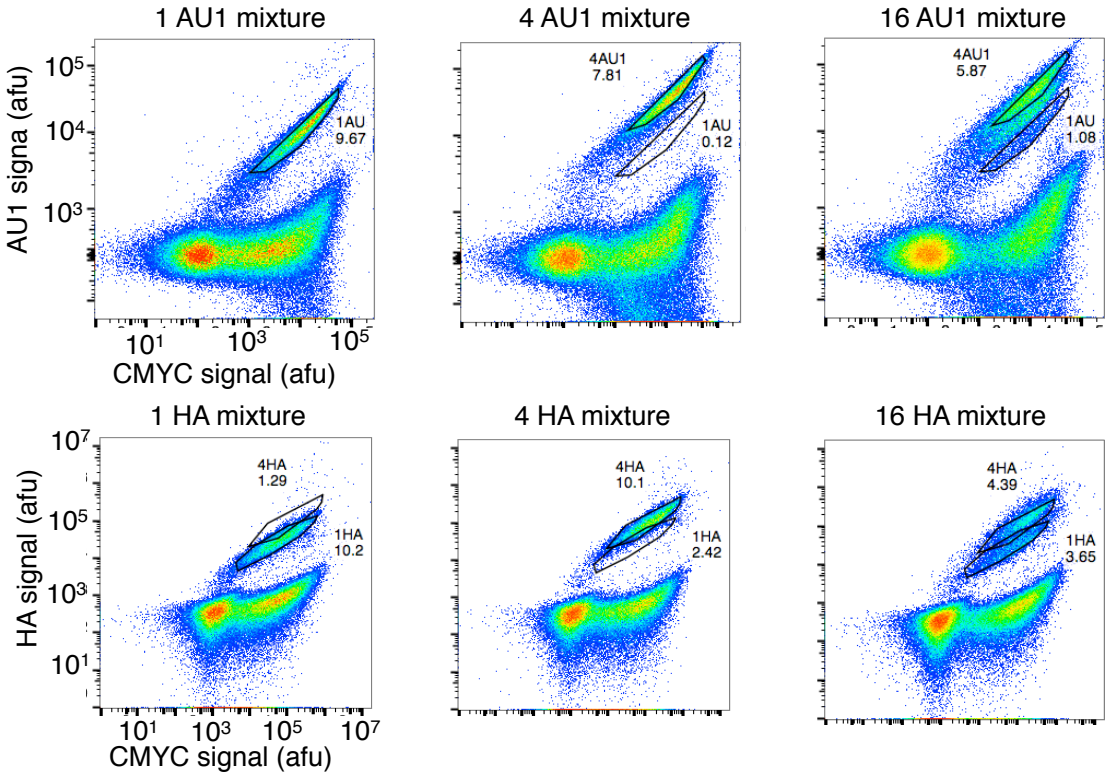


Figure 3.5: Long epitope tag repeats are unstable. Mixtures containing either 1, 4, or 16 repeat plasmids were transformed, expressed, and immunolabeled in yeast cells. Cells expressing 1 and 4 repeats create populations with more homogenous fluorescence and exhibit an expected increase in repeat length. Cells expressing 16 repeat proteins have more heterogeneous fluorescence and no increase in fluorescent signal was observed.

Additionally, we isolated yeast clones expressing 16 repeat barcodes from the library and analyzed the immunolabeled cells by flow cytometry. We found that daughter cells from each clone had a uniform but subtly different fluorescent signature, suggesting that the observed fluorescence heterogeneity was caused by clonal due to repeat instability (**Figure 3.6**). Importantly, yeast cells expressing 16 repeat barcodes derived from plasmids that were checked for the correct length on an agarose gel produced a uniform population with increased fluorescent signal (See Section 3.4). This observation suggests that long repeat plasmids are not unstable in yeast.

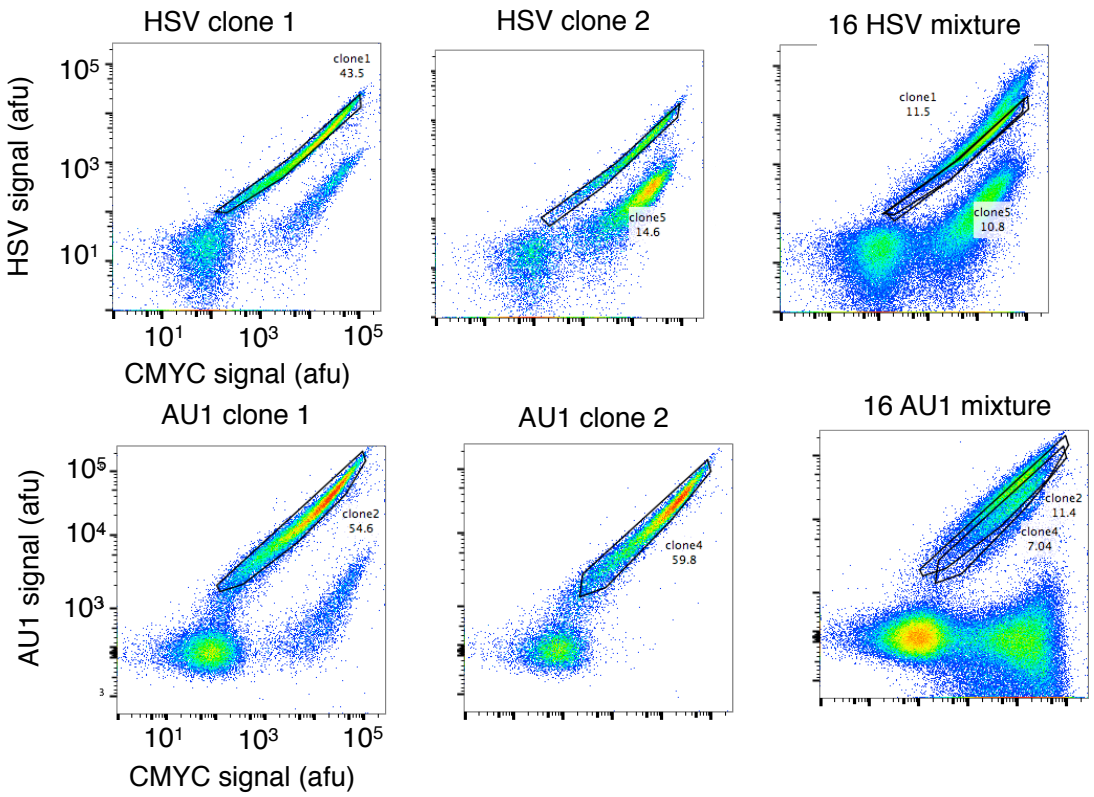


Figure 3.6: Repeat plasmid instability underlies fluorescence heterogeneity. Single yeast clones were isolated from 16 repeat libraries and analyzed by flow cytometry. Immunolabeled daughter cells from single clones produced uniform fluorescent signatures, suggesting clonal variation caused by repeat plasmid instability is the underlying cause of the observed fluorescence heterogeneity of the 16 repeat barcodes.

3.4 Analysis of Epitope Tag Repeat Lengths and Barcode Plasmids by SMRT Sequencing

To further investigate epitope tag repeat instability, we used SMRT sequencing (Pacific Biosciences) to investigate the lengths of epitope tag repeats in plasmids. SMRT sequencing is a type of next generation sequencing exhibiting long read lengths and uniform coverage (limited GC or AT bias). First, repeat plasmids were restriction digested to isolate the open reading frame and size exclusion was used to remove

backbone DNA fragments. After sequencing, raw data was filtered and circular consensus reads were determined using Pacbio SMRT analysis portal.

Along with our collaborator, Dr. Greg Vorsanger at Johns Hopkins University, we developed a computational method for barcode identity and repeat length analysis (**Appendix A**). The method uses the LALIGN algorithm to determine the number of epitope tag repeats and types of epitope tags present in each SMRT sequence read. Then, custom Python software was used to compile and synthesize the information from all reads (**Appendix B**). CCS consensus sequences were filtered for a quality score of 0.9 and at least 3 passes, and LALIGN alignments with an expectation value $> 10^{-4}$ were rejected.

In general, we found the majority sequence reads contained epitope tag repeats of the expected lengths, namely 1, 2, 4, 8, and 16, and 32, with the exception of GLU whose dominant lengths were 1 and 14 repeats (**Figure 3.7**). Additionally, we found a significant number of reads with unexpected repeat lengths ranging from 3-50 repeats. Reads of unexpected repeat lengths ranged from 0.11% for HIS to 0.48% for HSV (**Table C.2**). Taken together, this illustrates that epitope tag repeat plasmids are unstable in *E. coli* and area capable of forming a range of sizes. Furthermore, we examined the SMRT data on a barcode basis and found that the number of barcodes with different repeat lengths differed significantly from expected. We found 717 unique barcode sequences comprised of combinations and repeat lengths of HA-HSV-HIS-AU1-GLU-FLAG epitope tags (**Table C.1**), but we expected that there were only 110 unique barcodes in the sample (**Table C.3**)

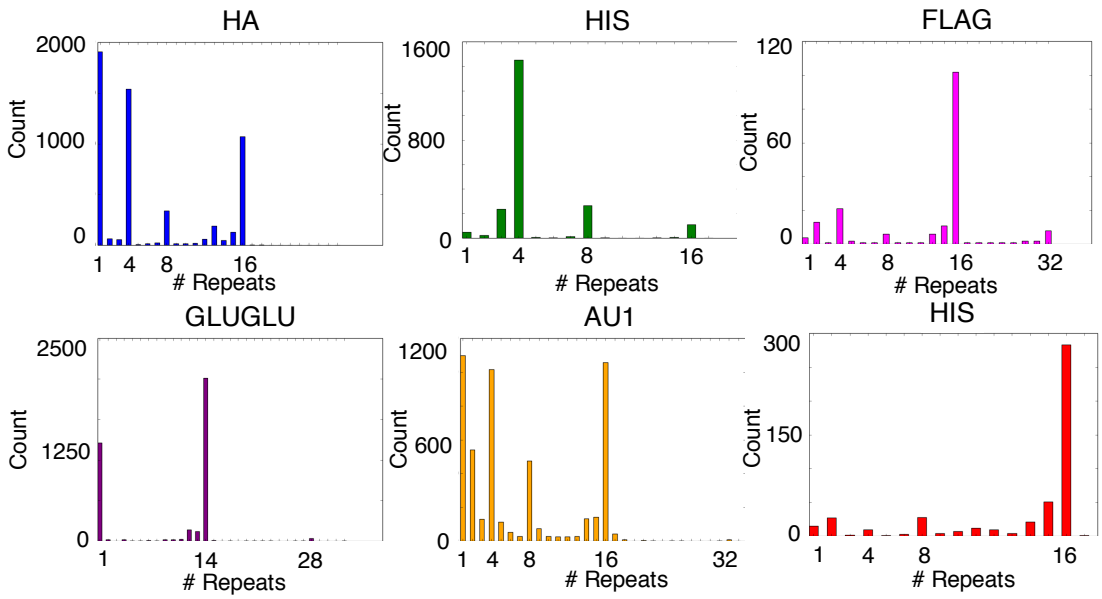


Figure 3.7: Distribution of epitope repeat lengths found using SMRT sequencing. Epitope repeat lengths had a central tendency towards expected sizes, namely 1, 2, 4, 8, and 16 repeats. However, a significant number of reads contained repeats of unexpected sizes, ranging from 3-50 repeats, supporting the hypothesis that epitope tag repeat plasmids are unstable in *E. coli*.

3.5 Cellular Expression of Single-Color, Multiple-Intensity Fluorescent Barcodes

Epitope tag repeat sequences were expressed in cells as genetic fusions to the C-terminal domain of alpha-agglutinin under the constitutive GPD promoter. The fusion proteins also contained N-terminal cmyc tags, which, due to the presence of the GPI anchor at the C-terminus, allowed monitoring of full-length expression of the epitope tag repeat fusion (**Figure 3.8**). Cells expressing the fusion proteins were immunolabeled with an antibody specific for cmyc and analyzed by flow cytometry. Surprisingly, a general trend was observed in which cmyc signal increased slightly

with number of repeats demonstrating that epitope repeat sequences up to 34 kDa are well-expressed.

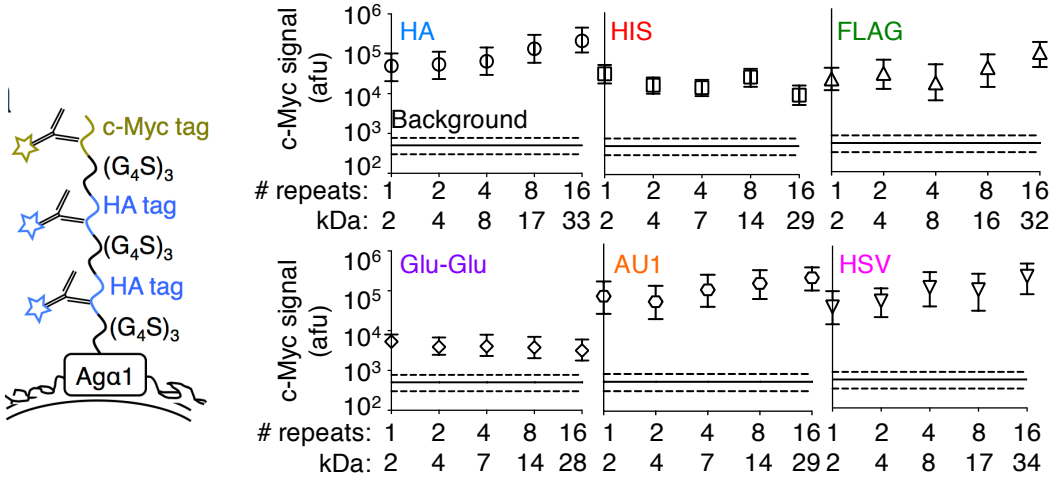


Figure 3.8: Cellular expression of epitope tag repeats. Epitope tag repeats contain an N-terminal cmyc tag for expression normalization and a C-terminal fusion to the alpha-agglutinin yeast surface protein. In general, epitope tag repeats were well-expressed and their expression did not decrease with repeat length.

Immunolabeling of cells expressing tandem epitope tag repeats of varying lengths, fused to AGα1, with primary antibodies followed by secondary antibodies conjugated to Alexa Fluor 647, generally resulted in increases in fluorescent signal with increasing numbers of epitope tag copies (**Figure 3.8**). Fusion of the protein to tandem epitope tag repeats resulted in four to ten-fold increases in fluorescence intensity for HIS, FLAG, AU1, and HSV epitopes as compared to a single epitope tag. Large increases in signal, 54-fold and 101-fold, were observed for 16 HA and 16

GLUGLU repeats, respectively, under the conditions tested. Such large increases likely arose from the choice of antibody labelling concentration.

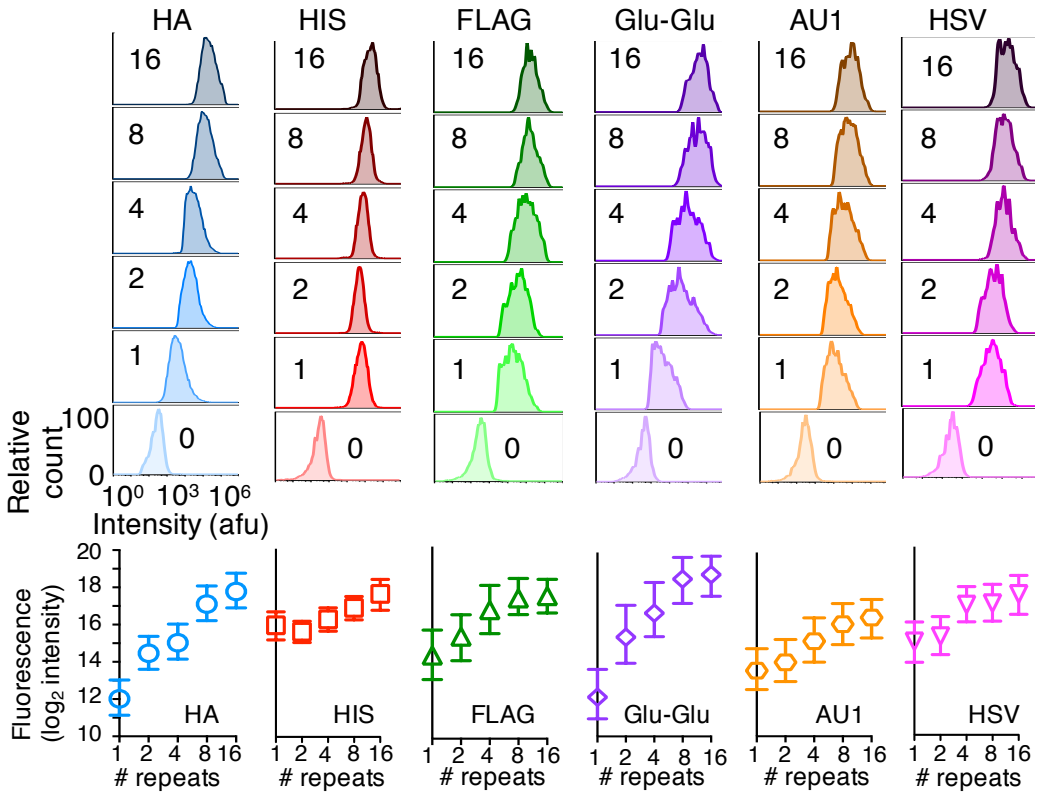


Figure 3.9: Multiple epitope tag repeats increase immunofluorescence intensity. Cells expressing the C-terminal alpha-agglutinin domain fused to a range of epitope tag repeat lengths were analyzed by flow cytometry after labeling with unconjugated primary antibodies and Alexa Fluor 647-conjugated secondary antibodies. Immunofluorescence intensity increased with repeat number up to 101-fold.

Antibodies specific for HA and AU1, directly conjugated to PE and PE-Cy7 respectively, were used to immunolabel cells expressing AGα1 fused to several tandem epitope tag repeats of varying length. Fluorescent signals increased 31-fold for

16 HA repeats and 34-fold for 16 AU1 repeats as compared to a single epitope tag **(Figure 3.9)**. The use of tandem epitope repeats separated by flexible linkers may therefore be especially beneficial for immunolabeling with directly conjugated primary antibodies, as they often produce a lower median fluorescent signal than fluorophore-conjugated secondary antibodies, which benefit from signal amplification during secondary labeling.

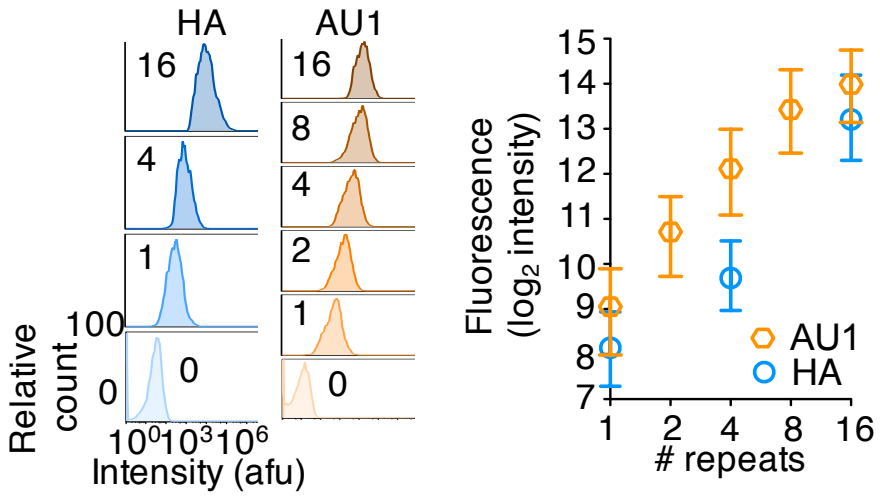


Figure 3.10: Epitope tag repeats increase immunofluorescence intensity with direct detection. Cells expressing the C-terminal alpha-aggutinin domain fused to a range of epitope tag repeat lengths were analyzed by flow cytometry after labeling with fluorophore-conjugated antibodies. Immunofluorescence intensity for repeat proteins was improved over 30-fold as compared to a single epitope tag when cells were immunolabeled with fluorophore-conjugated antibodies.

3.6 Application of Long Epitope Tag Repeat Fusions for Improved Flow Cytometric Analysis of Endogenous, Low-Abundance Proteins in Single-Cells

Given the large increase in immunofluorescent signal afforded by long epitope tag repeats, we posited that they would be useful for detection of low abundance proteins. 1 and 16 FLAG epitope tag repeats were subcloned into an integrating plasmid containing regions of GFP homology and transformed into seven yeast GFP fusion clones [55] that have been shown to be undetectable by flow cytometry by us and others [41]. The resulting clones expressed 1 or 16 FLAG fusion proteins from the endogenous promoter instead of a GFP fusion. Epitope tag fusion clones were grown to log phase, immunolabeled, and analyzed by flow cytometry and confocal microscopy (**Figure D.1**). The median fluorescence intensity did not vary drastically from experiment to experiment (**Figure 3.10**), illustrating that 16FLAG repeats are a reliable and reproducible method for protein immunodetection by flow cytometry.

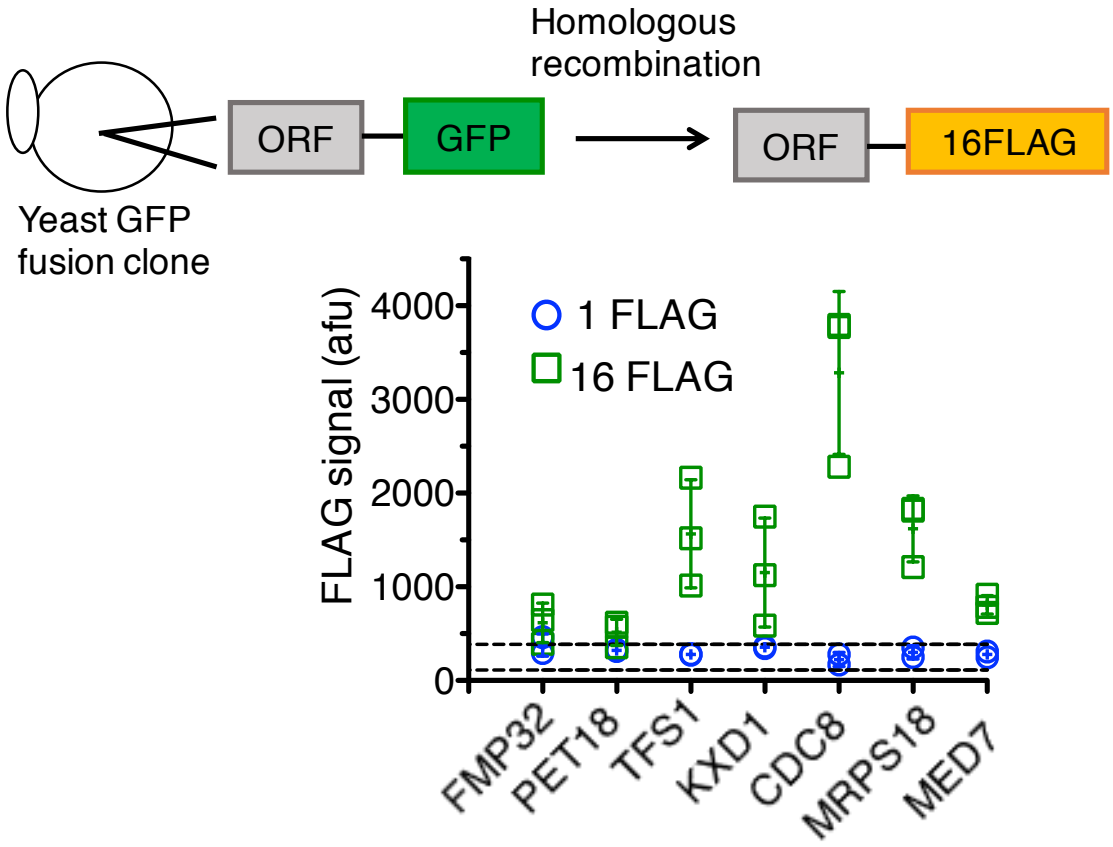


Figure 3.11: Detection of endogenous, low abundance yeast proteins by long epitope tag repeat fusion. 16FLAG repeat fusions were integrated into the yeast genome using a plasmid with homology to GFP. Detection of 16FLAG fusion proteins was reproducible (n = 3 experiments), and low abundance proteins were not detected with 1FLAG fusions.

Proteins ranging in abundance from approximately 200 to 7,000 molecules per cell were detected above background, and their single cell expression profiles were elucidated, using 16 FLAG fusions but not 1FLAG or GFP fusions (**Figure 3.11**). Fusion of low abundance proteins to 16 FLAG repeats improved the limit of detection by 40-fold over previous flow cytometry based methods [41], [76] permitting detection of proteins expressed as levels as low as 200 molecules per cell. Moreover,

addition of FLAG repeats did not affect the apparent protein expression level (**Figure D.2**), suggesting the increase in limit of detection is due to the enhanced signal afforded by brighter fluorophores and polyclonal antibodies. The protein expression distribution is comparable for FLAG repeats and GFP fusions as shown by detection of a highly abundant yeast GFP fusion protein (**Figure D.3**) .

Epitope tag repeats enabled quantification of low abundance proteins. Quantitative Western blotting was used to correlate GFP fluorescence and protein abundance (**Figure D.4**). To relate 16FLAG signal and protein abundance, eGFP and 16FLAG were expressed in cells as a fusion to AGα1, immunolabeled, and measured by flow cytometry. 16 FLAG signal was correlated to protein abundance by linear regression, and the relationship was used to quantify the abundance of the epitope tag fusion proteins. Quantification by 16FLAG repeats loosely correlates with previously reported protein abundances ($R^2 = 0.24$) (**Figure D.5**) which is in agreement with other studies [10], [32], [41].

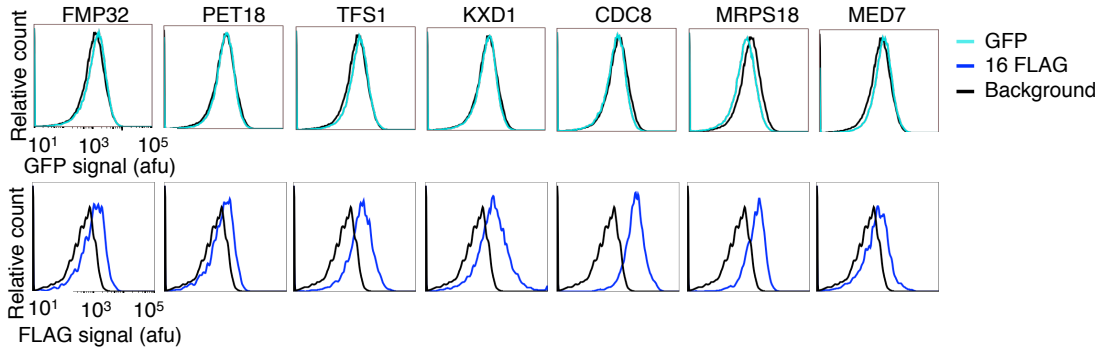


Figure 3.12: Single-cell protein expression profiles of low abundance endogenous proteins. Protein expression profiles for low abundance proteins expressed at levels as low as 200 molecules per cell were elucidated by flow cytometric analysis of immunolabeled cells expressing 16FLAG fusion proteins, but not GFP fusions.

3.7 Discussion

To improve the number of spectrally distinct cell barcodes over previous systems, we used immunolabeling with fluorophore-conjugated antibodies rather than fluorescence proteins. As aforementioned, immunolabeling permits the use of fluorescence proteins, quantum dots, and small organic fluorophores, increasing the number of colors available and therefore the number of possible distinct fluorescent barcodes. Furthermore, fluorescent cell barcodes are analyzed using flow cytometry rather than fluorescence microscopy, allowing for greater throughput [14]. Current flow cytometers are capable of analyzing up to 18 fluorescence parameters simultaneously at a rate of >10,000 cells per second [58], potentially enabling 2^{18} or more than 260,000 fluorescence barcodes with only a binary scheme.

Epitope tag repeat barcodes up to 34 kDa, fused to the C-terminal alpha agglutinin domain in yeast for surface expression, exhibited full-length constitutive expression at similar or greater levels as a single epitope tag. An apparent increase in relative expression with repeat number could be due to greater accessibility, through the cell wall, of the cmyc tag to antibodies in solution since the epitope tag is farther away from the cell surface. Expression of the GLU epitope tag fusion proteins appeared weaker than other epitope tags, likely because the GLU DNA sequence used was not codon optimized for expression in yeast. Overall, the data suggests that unstructured epitope tag repeat proteins up to 34 kDa are well expressed. These observations agree with the finding that protein disorder does not strongly correlate with degradation rates *in vivo* [77].

Epitope tag repeat fusions enhanced the immunodetction of surface-displayed proteins by up to 101-fold using flow cytometry. Moreover, 16 FLAG epitope repeat fusions were shown to enhance the limit of detection of endogenous yeast proteins by

40-fold, enabling more than 1,600 proteins expressed at as low as 200 molecules per cell to be studied by flow cytometry. The larger than expected increase in fluorescent signal is due to the higher brightness of Alexa Fluor 647 as compared to GFP, as well as the enhancement of fluorescence signal by polyclonal antibodies. The proof of principle experiment shown here could easily be expanded upon to study hundreds or thousands of yeast proteins with single-cell resolution, potentially uncovering interesting bimodal protein expression profiles or those with large variation. The long epitope repeat fusion approach could also be useful for detection of low abundance proteins in other microbes and mammalian cells.

Overall, these findings demonstrate that immunofluorescence signals can be substantially increased through fusion of a protein of interest to long, unstructured polypeptides composed of tandem repeats of epitope tags separated by flexible linkers, enabling single-cell detection and quantification of endogenous, low abundance proteins by flow cytometry. Fusing proteins of interest to tandem epitope tag repeat polypeptides may be useful to overcome hurdles associated with detecting proteins in several cases, including detection of low abundance antigens, detection by weakly binding antibodies, use of non-optimal excitation wavelengths or filters for fluorophores used, and for fluorophores which are dim, the latter two of which are commonly encountered in multicolor flow cytometry. Long epitope tag repeat fusions may also improve the efficiency of other antibody-mediated processes, including immunopurification and Western blotting, due to avidity effects. Additionally, the DNA repeat sequence expansion method described here may useful for generating proteins composed of repeating domains for molecular recognition, protein

purification, and biosynthetic polymer synthesis. The method could also be useful in the construction of vectors encoding CRISPR guide RNAs in tandem.

Chapter 4

ENGINEERING A SIX-COLOR, HIGH-THROUGHPUT CELLULAR FLUORESCENT BARCODING SYSTEM

4.1 Introduction

In this chapter, we develop barcodes with unique fluorescence intensities, and quantify uniqueness as a function of cells captured and false positives. The construction of hundreds of genetically-encoded fluorescent barcodes by combination of epitope tags is discussed, and the diversity of the barcode library is assessed by flow cytometry. In addition, the effect of barcode length on plasmid transformation bias and cell growth rate is examined. The effect of fluorophore brightness on the number of unique fluorescent barcodes and the influence of the presence of certain epitope tags on barcode brightness is assessed.

4.1.1 Fluorescent Cell Barcoding Systems with Multiple Intensities

Previous fluorescent barcoding systems that have achieved multiple fluorescence intensities using a single fluorophore have done so by modulating dye [18] or polymer dot loading [22] amount. These systems are disadvantageous compared to genetically encoded barcoding schemes because they are single-use. Previous genetically encoded fluorescent barcoding systems have not been successful in achieving multiple intensities using a single fluorophore, because of too much variance in fluorescent signal possibly due to fluorophore brightness or low protein expression levels [24], [25], [27]. Recently, it was demonstrated that different

translational control elements can be used to modulate protein expression levels and create three different intensities [24]. To our knowledge, our approach of fluorescence normalization and immunodetection is the first instance of a genetically encoded cellular barcoding system that can overcome these limitations to achieve four distinct intensities using a single fluorophore.

4.1.2 Multicolor Flow Cytometry

Multicolor flow cytometry is a powerful tool because it enables tens of parameters of interest to be measured in single-cells. Current flow cytometers can measure more than 17 parameters simultaneously [65]. There are a number of variables that have to be considered when designing a multicolor flow cytometry panel, such as the lasers and detectors available on the instrument, compatible fluorophores and their relative brightness, antibody-fluorophore conjugation or secondary detection, and compensation [78]. The wavelengths of the lasers and detectors dictate which fluorophores can be used for the experiment. Typically, the brightest fluorophores are more desirable, as they have high quantum yields and extinction coefficients, their spectrum overlaps minimally with cellular autofluorescence, and they are capable of being measured with high sensitivity [78]. Fluorophores brightness will also depend on the flow cytometer being used due to differences in filter position and bandwidth and laser power.

It is often necessary to directly conjugate fluorophores to antibodies in multicolor flow cytometry due to the limited number of antibody species and isotypes available. This can be accomplished by a variety of different chemistries including linkage to cysteine and lysine groups as well as carbohydrates [50]. Alternatively, unconjugated antibodies against the target of interest and polyclonal fluorophore-

conjugated antibodies specific for the species and isotype of the unconjugated antibody ca be used to amplify the fluorescent signal.

In addition, it is desirable to use fluorophores that have non-overlapping spectral emission because of the added need for compensation [79]. Compensation is the mathematical correction of flow cytometry data for spectral overlap. Specifically, it is the subtraction, by use of a constant coefficient, of fluorescence spillover into more than one detector. Compensation can be detrimental to discrimination of positive and negative events because subtraction of large spillover often causes negative population broadening [65].

4.2 Construction of Barcodes with Multiple Unique Fluorescent Intensities

We have developed a collection of genetically-encoded protein barcodes for multiplexed cell analysis using immunofluorescence and flow cytometry. Our cell barcoding method can enable massive experimental and analytical sample parallelization, thereby reducing associated cost and effort (**Figure 3.2**). The barcode collection is composed of DNA sequences that when expressed in cells produce proteins that specifically identify the cells containing each sequence. Each barcode is composed of epitope tags differing in repeat number and combination connected by flexible $(G_4S)_3$ linkers (**Figure 3.1**). Barcodes are genetically encoded on a plasmid, and when expressed are displayed on the yeast cell surface by fusion to the C-terminal domain of the alpha-agglutinin mating protein. Each barcode contains an N-terminal cmyc tag for normalization of protein expression. Fluorophore-conjugated antibodies can specifically bind to epitope tags, producing unique fluorescent signatures that are measured with single-cell resolution by flow cytometry.

To create the barcode collection, first DNA encoding an epitope tag connected by a flexible linker was expanded exponentially up to 64 repeats (**Figure 3.3**), resulting in 35 plasmids. Then, barcode plasmids encoding different numbers of epitope tag repeats were constitutively expressed in yeast, immunolabeled, and analyzed using flow cytometry. We were able to create up to four distinct barcodes with different fluorescence intensities for a single color, using HA, AU1, and FLAG epitope tags, three distinct intensities for HSV, and two for HIS and GLU (**Figure 4.1**).

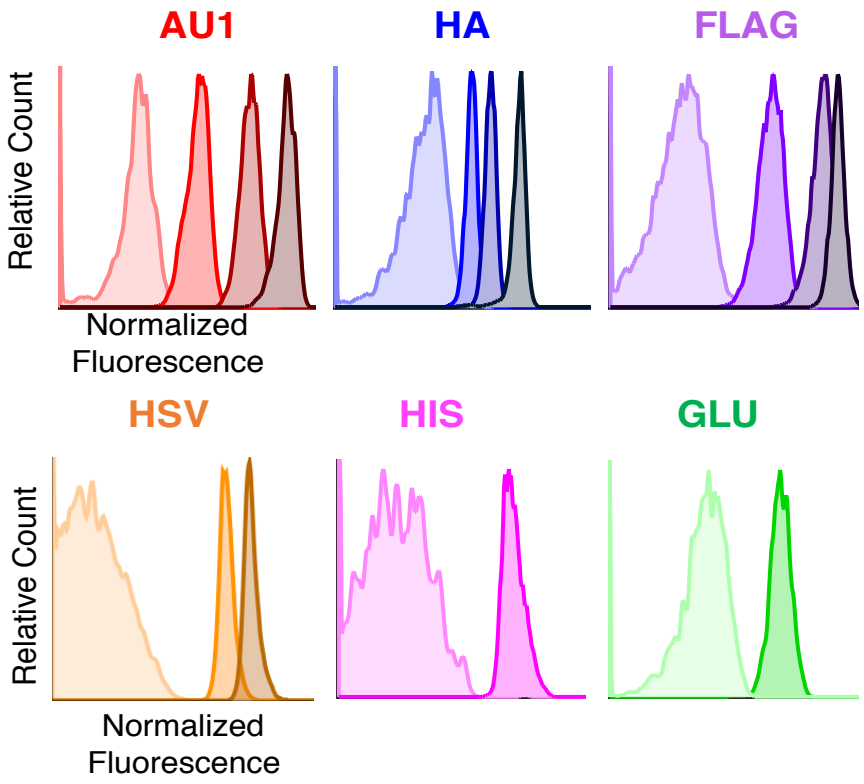


Figure 4.1: Construction of barcodes with distinct fluorescence intensities. Different fluorescent intensities were created by variation of epitope tag repeat length and normalization of total fluorescence signal to correct for variation in protein expression.

Distinct fluorescence intensities were achieved by normalization of total fluorescence signal (**Figure 4.2**). For each cell, the fluorescent signal from its immunolabeled epitope tags was normalized by the fluorescent signal from its immunolabeled cmyc tags. This normalization corrects for the variation in fluorescence signal resulting from differences in protein expression between cells. Specifically, up to four distinct intensities could be achieved using a single fluorophore after immunolabeling cells expressing 0, 1, 4, or 16 epitope tag repeats and normalizing the epitope signal by the cmyc signal.

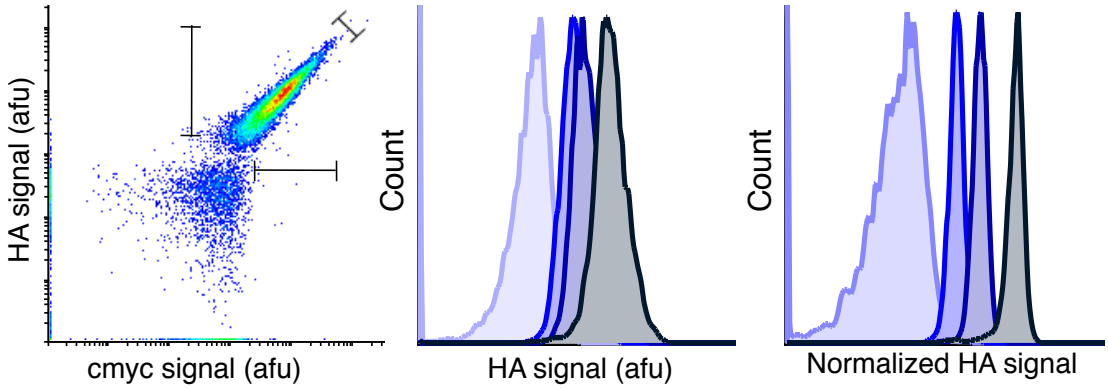


Figure 4.2: Creation of up to four distinct intensities per fluorophore. Cells exhibit large variations in fluorescence due to differences in protein expression. This variation can be minimized by normalization of one fluorescent signal by another for each cell, effectively correcting for differences in fluorescence due to protein expression. Up to two distinct barcodes could be distinguished for each fluorophore by total fluorescence signals. After normalizeation of HA signal by cmyc signal for each cell, up to four unique fluorescence intensities were achieved for a single fluorophore.

A barcode is considered to be unique if its immunofluorescent intensity minimally overlaps with that of other barcodes. The error associated with incorrectly

identifying cells as belonging to a particular barcoded population was quantified by manual gating. The percentage of incorrectly identified events as a function of the percentage of events captured showed that at least 10% of events can be captured with less than 1% incorrect for AU1, HA, and FLAG barcodes (**Figure 4.3**). Using this analysis method, we found that 0, 1, 4, and 16 repeats could produce distinct intensities for HA, AU1, and FLAG, and 0, 1, and 4 repeats for HSV.

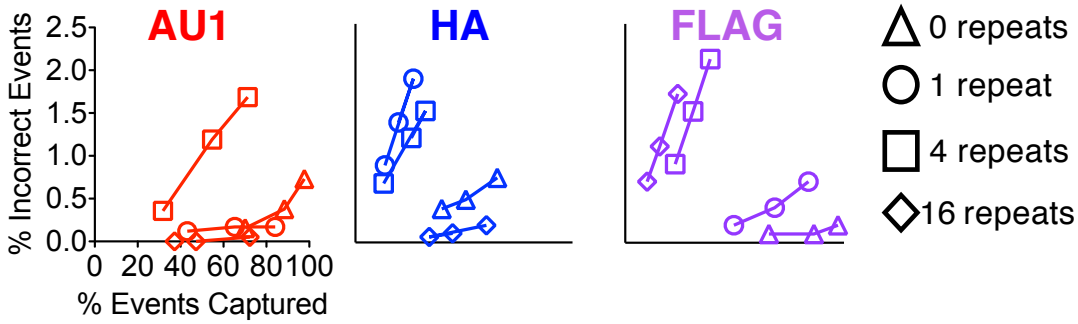


Figure 4.3: Quantification of distinct fluorescence intensities. A barcode with a particular fluorescence intensity was defined as being distinct if at least 10% of cells could be captured with less than 1% of cells belonging to a different barcode.

4.3 Barcodes with distinct fluorescent intensities are fluorophore-dependent

The number of fluorescence intensities that can be achieved using a single fluorophore is dependent on the brightness of the fluorophore used for detection. Initially, we found that up to four fluorescence intensities could be achieved using HA, AU1, and FLAG epitope tags immunolabeled with the brightest fluorophores, namely PE and Alexa Fluor 647. However, when dimmer fluorophores were used, the number of distinct barcodes decreased due to greater variation associated with dimmer

fluorescent signals. For example, four distinct intensities were achieved when FLAG barcodes were labeled with mouse α-FLAG and α-mouse Alexa Fluor 647, but not with α-FLAG PE-CY7 conjugate (**Figure 4.4**).

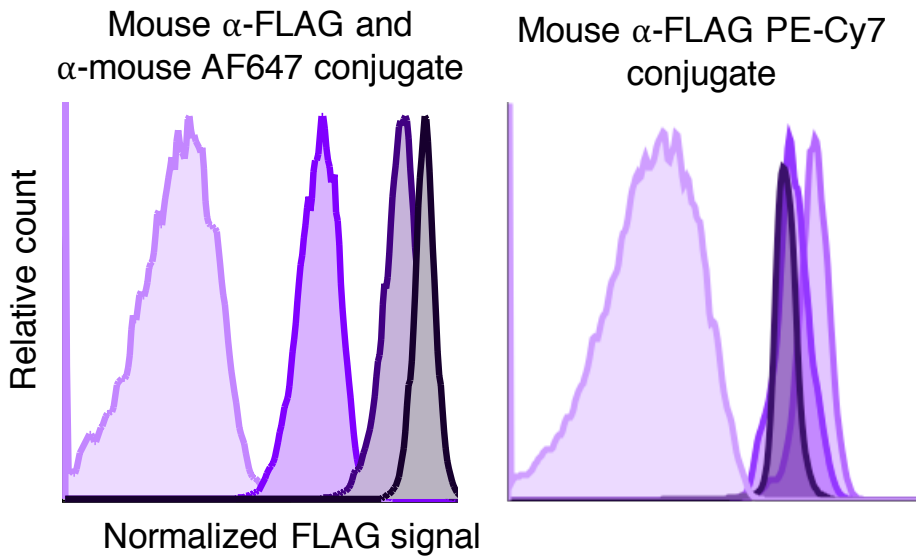


Figure 4.4: The number of distinct fluorescence intensities is fluorophore dependent. For example, FLAG was found to have four distinct intensities when antibodies that produce a very bright signal, such as Alexa Fluor 647, were used for immunolabeling. However, when dimmer fluorophores were used, such as PE-Cy7, only two unique fluorescence intensities could be achieved.

4.4 Creation of a Fluorescent Barcode Plasmid Library

In order to expand the fluorescent barcoding system beyond single-color barcodes, we used restriction digest based subcloning to combine different types of epitope tags together, resulting in barcodes with multiple fluorescent colors and intensities. Each epitope tag and linker is flanked by a unique pair of restriction enzyme sites, which can be used to excise particular epitope tags and combine them

together. Specifically, we created a barcode plasmid library comprised of up to 216 possible combinations of HA, HSV, HIS, AU1, GLU, and FLAG epitope tags with three intensities for HA, HSV, and AU1, and two intensities for HIS, GLU, and FLAG.

Three rounds of subcloning were used to generate the barcode library (**Figure 4.5**). In the first round, three subcloning steps were used to generate 16 plasmids, namely combinations of 1 and 4 HA and HSV (8 plasmids), 4HIS and 1 and 4AU1 (5 plasmids), and 4GLU and 1 FLAG (3 plasmids). In the second round, the HA-HSV library was crossed with the HIS-AU1 library for a total of 54 plasmids, and in the third round the HA-HSV-HIS-AU1 library was crossed with the GLU-FLAG library for a maximum possible 216 unique barcodes. We also made a library consisting of only epitope tag repeat lengths that would produce binary fluorescent barcodes, comprised of 1HA, 1HSV, 4HIS, 1AU1, 4GLU, and 1FLAG epitope tags.

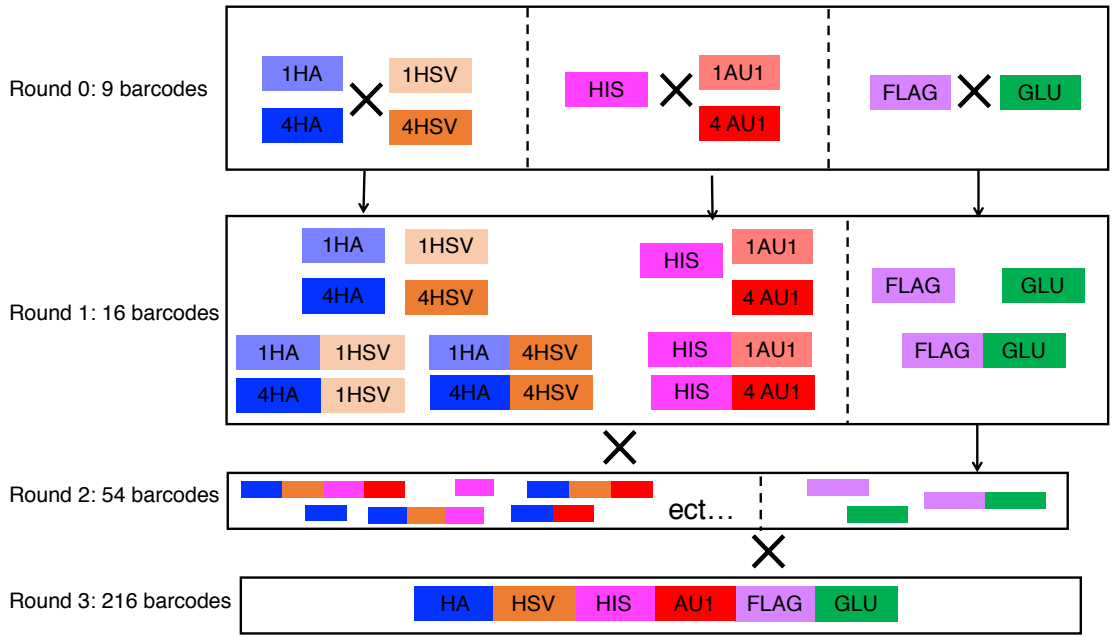


Figure 4.5: Barcode library creation by combination of epitope tag DNA sequences. A library of barcode plasmids was created by combining epitope tag repeat lengths that when expressed in cells and immunolabeled, resulted in distinct fluorescent barcodes. Three rounds of subcloning using restriction digest were used to create a library containing up to 216 distinct barcodes.

4.5 Assessment of Barcode Library Diversity by Immunofluorescence and Flow Cytometry

In order to determine how many unique barcodes that had been created, we transformed the barcode plasmid library into yeast, induced protein expression by galactose induction, immunolabeled the cells, and analyzed the barcoded cells by flow cytometry. For immunolabeling, we used 100nM of each antibody fluorophore conjugate, and for cmyc 35nM α-chicken Alexa Fluor 488 was used with the unconjugated α-cmyc antibody. Seven of the thirteen available colors on the FACSAria were used to analyze the library. Fluorophores were selected by their brightness and to minimize spillover between channels. Further information about the

epitope-fluorophore pairs and lasers and detectors used to analyze the barcoded library can be found in **Table 4.1**.

Table 4.1: Fluorophore panel and flow cytometer configuration used for barcode library analysis.

Epitope	**Flurophore**	**Laser**	**Detector**
GLU	Marina Blue	355	450/50
cmyc	Alexa Fluor 488	488	530/30
HA	PE	532	575/25
HSV	PE-Cy5.5	532	710/50
FLAG	PE-Cy7	532	780/60
AU1	Alexa Fluor 647	633	660/20
HIS	APC-Cy7	633	780/60

To distinguish between barcoded cell subpopulations within a mixture, we used a gating strategy to segment the cells by epitopes with binary unique fluorescent intensities and then by epitopes with multiple fluorescent intensities (**Figure 4.6**). First, cells were gated on forward and side scatter to exclude debris. Then, cells with high cmyc signal, which is indicative of protein expression level, were further examined for barcode fluorescence. Gating on cmyc also excludes those cells that do not express barcodes, which is characteristic of yeast surface display [64], and is typically 30-50% of the total cells.

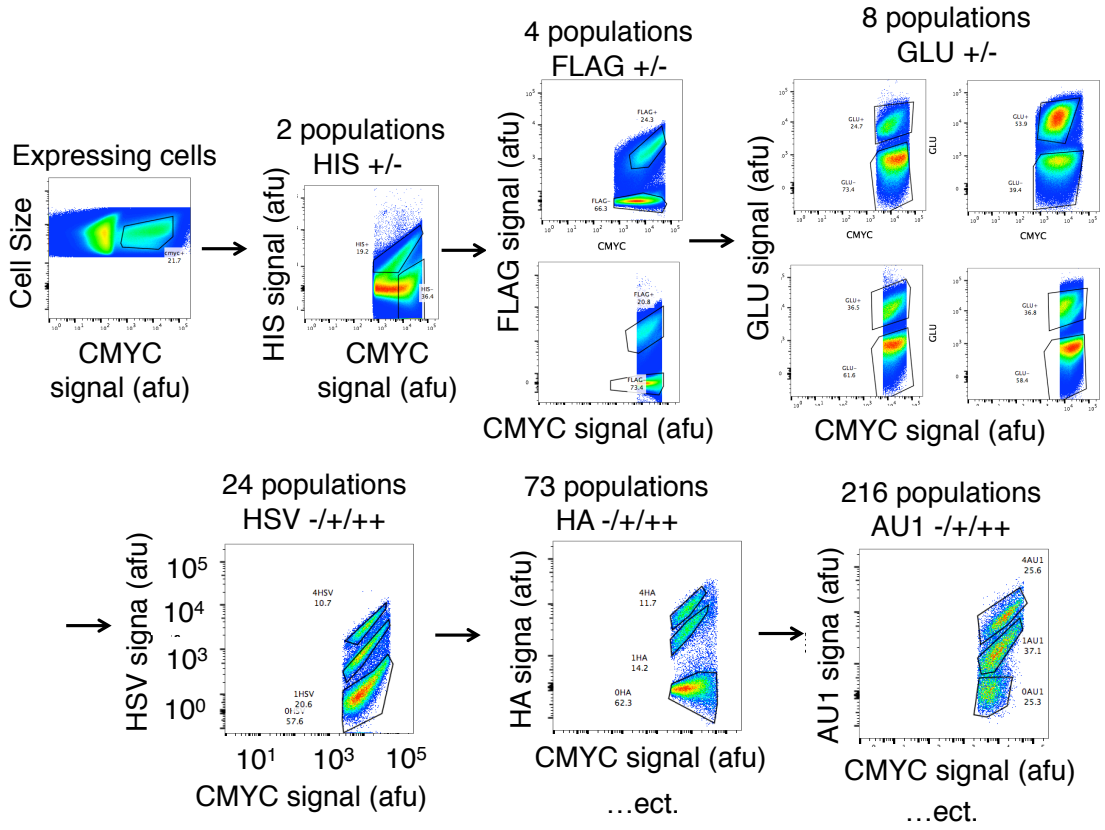


Figure 4.6: Method used to distinguish barcoded cell populations. Cells were gated on cmyc positive events to consider only those expressing barcodes. Then, cells were subdivided by epitope tags that produce binary fluorescent intensities (HIS, FLAG, GLU), and then by epitope tags that produce multiple fluorescence intensities (HSV, HA, AU1).

During barcode deconvolution analysis, it is critical to first segment cells by epitope tags which produce binary intensities after immunolabeling. This is because the presence or absence of other epitope tags can dramatically affect the fluorescence of epitopes with multiple intensities, and thus the location of a cell population expressing a particular barcode. For example, the fluorescence of cells expressing 1HA is increased dramatically when a GLU epitope is present (**Figure 4.7**). Other

examples include decreasing AU1 signal with increasing HSV signal, and increasing HA signal with HSV signal.

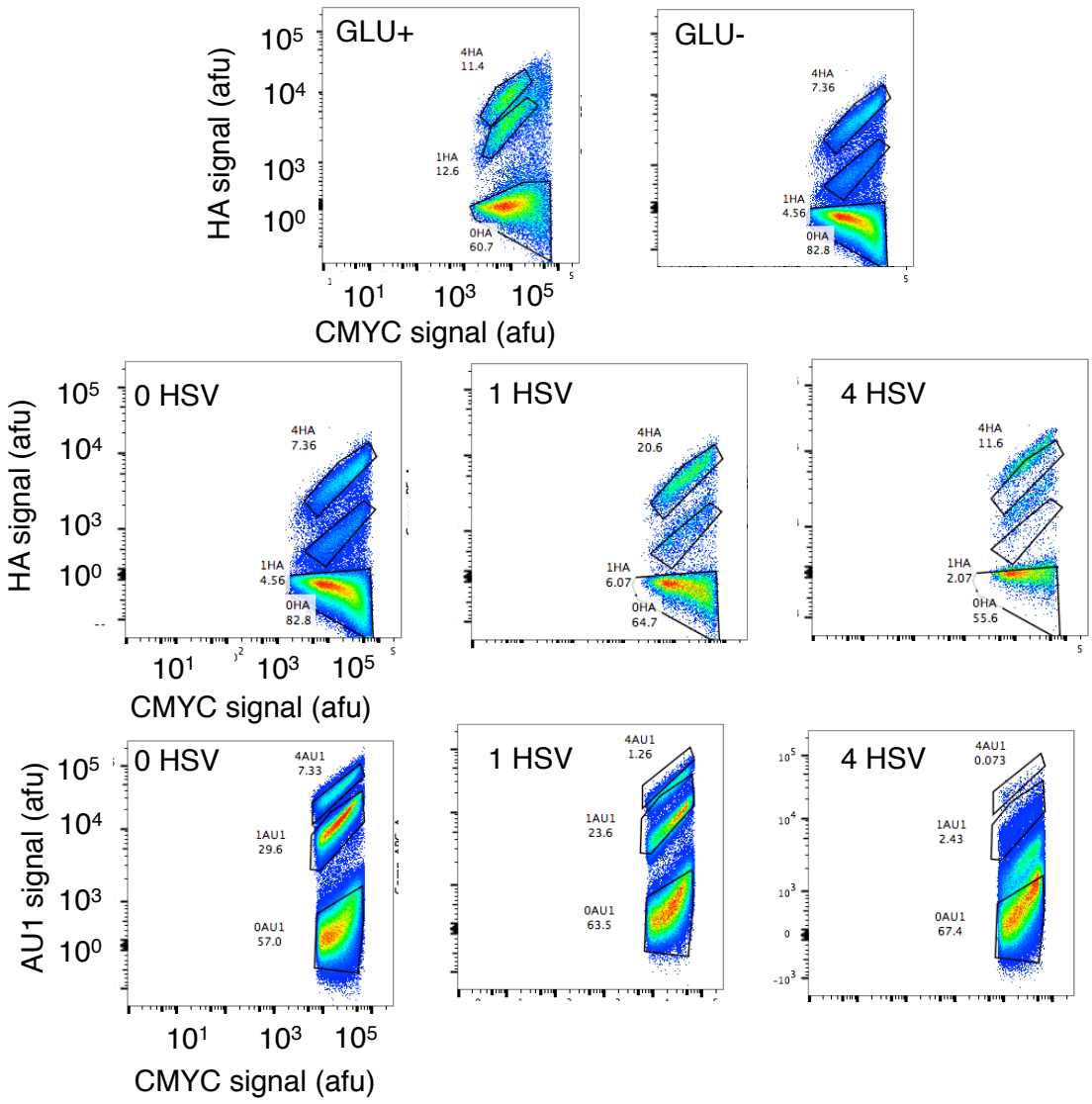


Figure 4.7: Barcode distinguishability for fluorophores with multiple intensities is affected by the presence or absence of certain epitope tags. Examples include an increase in signal for the 1HA population due to the presence of a GLU epitope tag (top), an increase in HA signal (middle) and a decrease in AU1 signal (bottom) with increasing HSV length. Identical gates are overlaid to illustrate fluorescence differences between barcodes.

The relative abundance of barcodes in the library was assessed by flow cytometry. 190 barcodes out of 216 possible were found to be abundant at greater than 0.01% (**Table E.1**). However, the distribution of barcodes in the library was biased such that only 13 barcodes represented 50% of the library. Also, average repeat length inversely related to barcode abundance, such that ~4 repeats was the average length for barcodes present at 1-10% abundance and ~7 repeats was the average length for barcodes present at 0.01-0.1% with an expected repeat length of 6.5 (**Figure 4.8**). Moreover, more abundant barcodes (0.1-10%) contained significantly more 0 and 1 HA, HSV, and AU1 repeats fewer 4 repeats, while the converse was true for less abundant barcodes (0.01-0.1%). Taken together, these results suggest that there is a transformation bias favoring smaller barcodes, which could be due to transformation preference favoring smaller plasmids and/or plasmids containing fewer repeat regions.

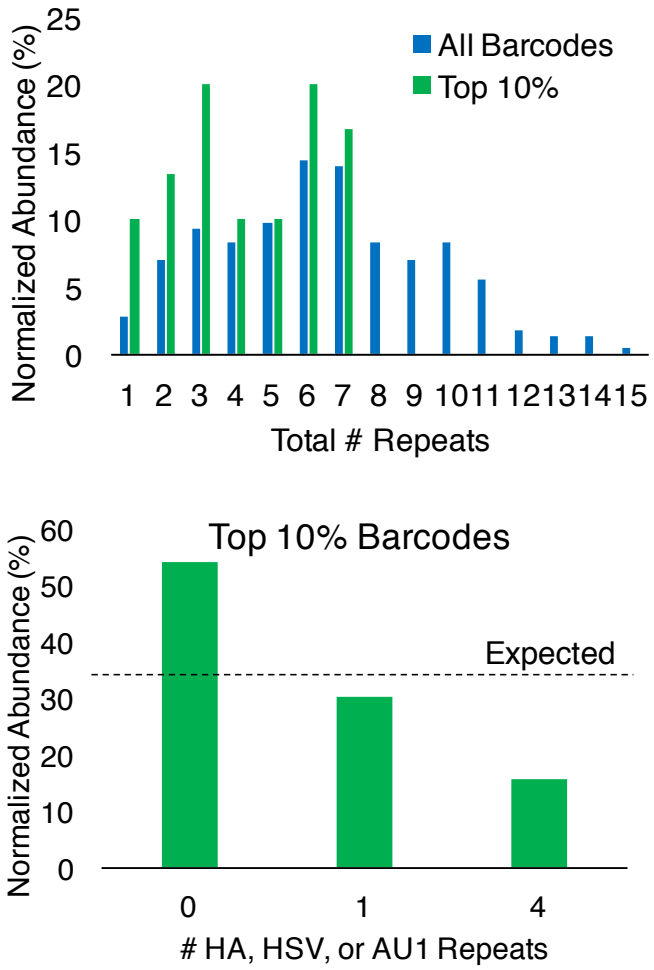


Figure 4.8: Relationship between repeat length and library abundance. The distribution of repeat lengths in the top 10% of barcodes is significantly smaller (4 repeats on average) than that of the overall library (6.5 repeats on average). In the top 10% of barcodes, zero HA, HSV, or AU1 repeats are highly enriched and four repeats are underrepresented. Taken together, this suggests that there is a bias favoring plasmids with shorter repeat lengths, possibly due to a transformation and/or ligation preference for smaller plasmids. This could explain the observed over-representation of certain barcodes in the library.

4.6 Correction of Barcode Library Abundance Bias by FACS

In order to correct the over-representation of particular barcodes, the library was sorted using FACS as to minimize the overly abundant barcodes and to simultaneously enrich the barcodes with low abundance. Specifically, the library was sorted using three logic strategies: 1. HA+ AND AU1+ AND GLU-, 2. HA+ AND AU1- and GLU+, 3. HSV+ AND (FLAG+ OR HIS+). The relative abundance of barcodes in the sorted libraries can be found in **Table E.2**. After sorting, the five barcode libraries were combined in equal proportions in order to normalize the relative abundance of barcodes in the library (**Figure 4.9**).

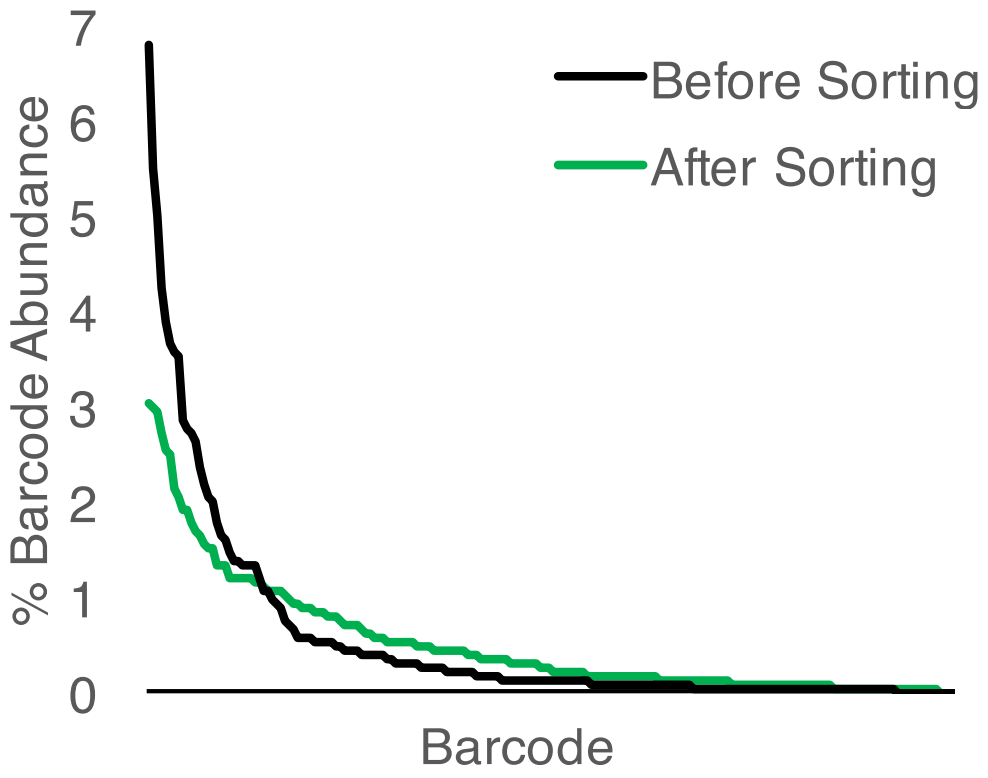


Figure 4.9: Normalization of 190-member barcode library. Flow cytometry analysis of the barcode library showed an over-representation of certain barcodes, such that only 13 barcodes comprised 50% of the abundance. After FACS, the library was normalized such that that 30 barcodes represented 50% of the library.

FACS successfully normalized the barcode library such that highly abundant barcodes were lessened and barcodes with low abundance were enriched (**Table 4.2**). The number of barcodes present at greater than 3% decreased by 50% from 8 to 4, and the number of barcodes between 0.1 and 1% abundance increased by approximately 1.5-fold, from 76 to 108.

Table 4.2: Abundance of barcodes in library before and after normalization by FACS.

Abundance	Before Sort	After Sort
>2%	16	8
1-2%	13	25
0.1-1%	76	107
0.01-0.1%	74	50
< 0.01%	37	26

4.7 Optimization of Promoter and Barcode Expression Conditions

Initially, a strong constitutive GPD promoter was used to drive barcode expression for ease of use. To examine whether barcode expression affected cell growth rates, a mixture of cells constitutively expressing 10 barcodes of different lengths was grown overnight at 30°C from stocks in either log phase, stationary phase, 4°C or -80°C, and then immunolabeled to determine barcode distribution (**Figure 4.10**). No significant differences in barcode abundance was found after less than 24 hours.

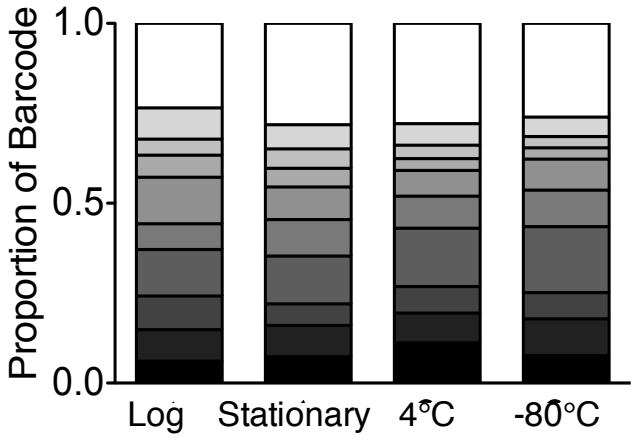


Figure 4.10: Effect of different environmental conditions on barcoded cell growth. After overnight growth after revival from log phase, stationary phase, 4°C or -80°C stocks, a mixture of cells expressing barcodes with different lengths did not exhibit any significant growth biases.

In addition, the abundance of cells expressing eight barcodes of different lengths, ranging from 1 to 21 repeats or approximately 2 kDa to 40 kDa, was examined over three days of growth at 30°C (**Figure 4.11**). After three days of growth, cells expressing smaller barcodes ranging from one to five repeats outcompeted those expressing larger barcodes (seven or more repeats). This suggests that expression of larger barcodes hampers cell growth rate over time scales longer than one day.

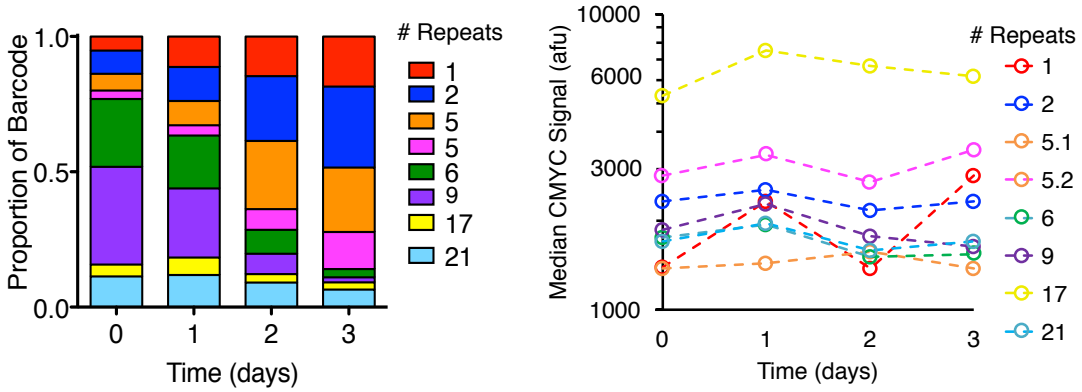


Figure 4.11: Growth of cells constitutively expressing barcodes over longer time scales. Over three days of growth, cells expressing shorter barcodes (less than 5 repeats) outcompeted those with longer barcodes. No decrease in CMYC signal was observed, suggesting barcodes were not degraded.

In order to lessen the barcode-induced cellular growth bias, the constitutive GDP promoter driving barcode expression was swapped with a galactose (GAL) inducible promoter. Induction times and temperatures were tested in order to maximize barcode expression levels. Specifically, a library of cells expressing barcodes with different lengths and combinations of HA, HSV, HIS, AU1, GLU, and FLAG was grown overnight in non-inductive glucose media to log phase. Then, cells were passaged into galactose media with 1% glucose for induction at 20°C or 30°C over three days. Expression was monitored by immunolabeling with an antibody against the cmyc tag (**Figure 4.12**).

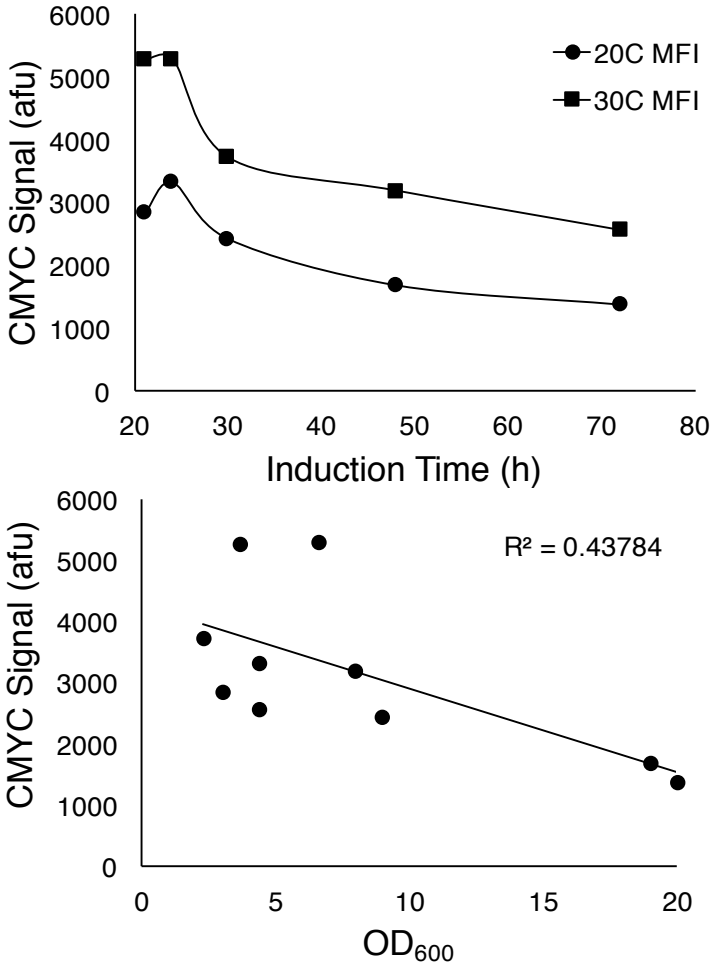


Figure 4.12: Optimization of barcode induction conditions. Barcode expression was monitored over three days of induction in galactose media at 20°C or 30°C (top panel). 21-24 hours of induction at 30°C resulted in the highest expression levels. Barcode expression levels were higher in log phase cells (bottom panel).

Barcode expression was found to be highest at earlier induction times of 21-24 hours, and expression levels were 1.5-2 fold higher at 30°C as compared to 20°C. Expression decreased by 40-50% after three days of induction at both temperatures

tested. In addition, barcodes were more highly expressed in log phase cells, and decreased with a function of cell density ($R^2 = 0.44$).

4.8 Discussion

It was found that up to four unique barcodes could be produced using a single fluorophore if the antibodies used for immunodetection had high brightness. This was only achievable using two spectrally distinct fluorophores, PE and Alexa Fluor 647, given the constraint of the flow cytometers available to us. However, it may be possible if a flow cytometer equipped with a violet laser was used. In this instance, extremely bright fluorophores like quantum dots could be employed [80].

Although 16 epitope tag repeats produced a fourth distinct intensity in some cases upon immunolabeling, only 1 and 4 repeats were included when the barcode library was constructed. Due to the instability of long repeats DNA, each plasmid had to be screened for the correct size insert in order to make sure the repeat region had not become shorter during the subcloning process. This process would become too laborious when large numbers of barcodes were created in a library format.

A saturating amount of antibody was used to maximize fluorescent signal and minimize the deviation of barcoded populations. In some cases, this was not sufficient to generate a high enough signal to distinguish a significant amount of cells from the negative. Thus, four repeats were used instead of one for GLU and HIS epitope tags to create binary barcodes for these epitope tags. The use of four epitope repeats allowed the majority of cells to be distinct from the negative when dim fluorophores such as Marina Blue and APC-Cy7 were used. On a molecular level, four epitope repeats likely accommodate at least two fluorophore-conjugated antibodies, while one or two

repeats would likely only accommodate one, effectively doubling the fluorescent signal when four repeats are used.

The fluorescence of a given epitope tag can be affected by the presence or absence of other tags. This phenomenon is most noticeable for epitopes that produce multiple intensities upon immunolabeling because of their narrow fluorescence distribution compared to epitopes that produce binary intensities. The observed differences may be the result of one or a combination of the following: differences in protein expression level due to barcode length, errors in compensation due to mathematical subtraction of a large fluorescent signal out of channel with a dimmer signal [79], energy transfer between fluorophores, and protein structure blocking epitope tag sites from being bound by antibodies.

It was found that cells constitutively expressing barcodes of different lengths were not outcompeted over short time scales (one day or less) or after revival from stationary phase cultures, or stocks stored at 4°C or -80°C. However, over longer time scales of two or more days, cells expressing shorter barcodes outcompeted those expressing longer barcodes. This suggests that the growth bias imposed by constitutive barcode expression could be overcome for experiments lasting one day or less by increasing the proportion of cells with longer barcodes to shorter barcodes in the inoculum. Another potential solution would be to construct barcodes of equal lengths by the addition of scrambled epitope tags or other spacers. Ultimately, inducible barcode expression was adopted because of concern for using barcodes in a library format in which some rare members may be lost due to growth biases. Barcodes expression was found to decrease with cell density, indicating that cells in stationary phase do not express barcodes as well as those in log phase. This could be due to a

variety of factors including nutrient limitation causing a decrease in heterologous protein translation and secretion, and structural changes to the yeast cell wall during stationary phase [81]–[83].

Chapter 5

GENERATION OF THOUSANDS CELLULAR FLUORESCENT BARCODES USING ELEVEN-COLORS

5.1 Introduction

In this chapter, the creation of more than 1,000 unique barcodes is discussed. Specifically, combinations of seven different epitope tags were constructed using a library approach via homologous recombination or overlap extension PCR. FACS enrichment and cloning of rare barcodes is presented. Restriction digest and ligation was used to combine 11 epitope tags to create additional diversity. Analysis of barcode plasmid instability and work-arounds are presented, as well as flow cytometry analysis of barcode abundances. In addition, design considerations for optimizing a 13-color flow cytometry panel are discussed.

5.1.1 Molecular cloning techniques

There are a variety of molecular cloning approaches that can be used to combine multiple DNA fragments combinatorically, including yeast-mediated homologous recombination, restriction enzyme digestion, Golden Gate assembly, and Gibson assembly. Homologous recombination is a process by which linear DNA fragments with short homologous regions (30-50bp) can be combined in yeast cells [84]. This method has been used for both chromosomal insertion and deletion [85], as well as for ligase-free plasmid cloning [86]. For example, homologous recombination has been used to create a randomly shuffled antibody library with up to 10^7 members

[87]. Homologous recombination can be used to assemble many (up to 25) DNA fragments up to many kilobases in length for synthetic biology applications [88], [89].

There are also a variety of *in vitro* assembly methods for molecular cloning, including traditional restriction enzymes, Golden Gate cloning, and Gibson assembly. Traditional restriction enzymes combine DNA fragments together with high fidelity, but each fragment requires a unique pair of enzyme sites and typically multiple fragments cannot be subcloned simultaneously. In contrast, Golden Gate cloning uses type IIS restriction enzymes that cut outside of their recognition sequence, leaving overhangs of typically 4 nucleotides in length [90], [91]. Cut sites can be designed so that there are up to 256 unique overhangs, therefore allowing the assembly of multiple fragments using a single restriction enzyme. Using Golden Gate cloning, up to nine fragments and six repetitive DNA fragments have been assembled with 85-90% efficiency in a single reaction [92]. Also, Gibson assembly is a one-pot approach that can be used to combine multiple dsDNA fragments with short (20-40bp) homologous regions [93]. Specifically, dsDNA fragments are subjected to partial exonuclease digestion, followed by annealing of homologous regions, PCR to fill in the gaps, and ligation to repair the nicks. Gibson assembly is a useful tool for gene and genome cloning, as it has been used to assemble a synthetic genome with more than 500kb in length.

5.1.2 Fluorescence Activated Cell Sorting (FACS)

Fluorescence activated cell sorting (FACS) is an extension of flow cytometry in which cells with desired fluorescence properties can be physically separated from undesired cells. FACS is frequently used in protein engineering to isolate rare cells expressing high-affinity binding proteins [94], as well as in antibody development to

isolate B cells for high-affinity antibody production [95]. For example, FACS has be used to isolate bone marrow stem cells from mice for myocardial repair applications [96], as well as to enrich cells expressing high-affinity antibodies for protein engineering applications [97]. FACS allows desired cells to be sorted in real time into tubes or multi-well plates. Specifically, a particular charge is applied to a droplet that contains a single cell, and the charged droplet is separated into the correct container by electromagnetic deflection plates [98].

5.2 Expansion of Barcode Library Using Additional Epitope Tags

Seven additional epitope tags were used to further expand the barcode library from hundreds to thousands of combinations (**Table 5.1**). We used two approaches, homologous recombination and overlap extension PCR, to rapidly expand the barcode library using a one-pot method (**Figure 5.1**). DNA fragments either contained an epitope tag surrounded by flexible glycine-serine linkers with less than 80% homology with respect to each other, which was previously shown to be the threshold for homologous recombination in yeast [87], or only glycine-serine linkers to act as spacers.

In the homologous recombination approach, DNA fragments of different sizes containing glycine-serine linkers with or without epitope tags were generated by partial or full restriction digest. DNA fragments and a vector fragment containing a GAL promoter and the C-terminal alpha-agglutinin protein domain for yeast surface display were transformed into yeast. For the overlap extension PCR approach, DNA fragments containing two glycine-serine linkers with or without an epitope tag were generated by complete restriction digest. Then, fragments were used in an overlap PCR reaction in which the glycine-serine linkers would prime each other to make new

epitope tag combinations. The PCR product was transformed into yeast along with the backbone vector fragment to create new barcodes.

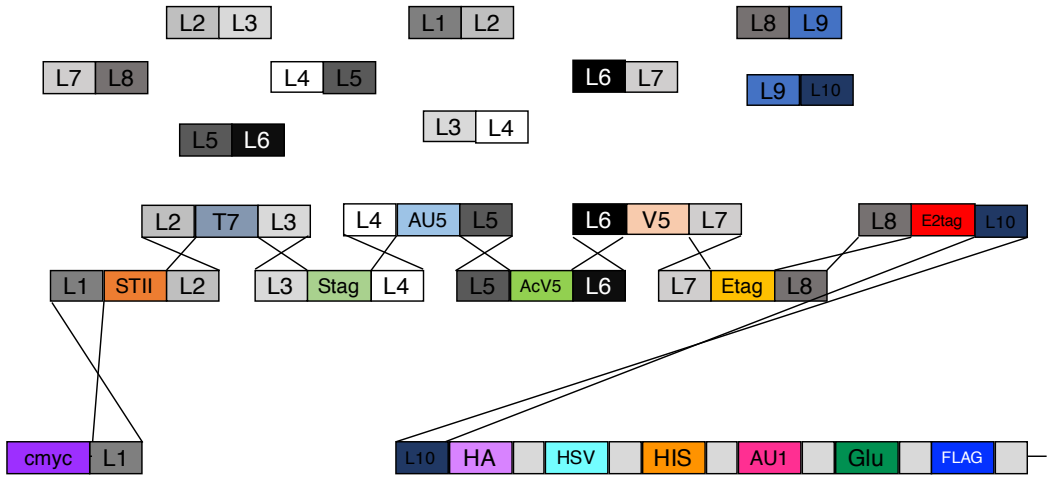


Figure 5.1: Rapid generation of barcode combinations by homologous recombination. Homologous recombination of DNA fragments containing glycine-serine linkers with or without an epitope tag were used to expand the barcode library in a one-pot approach. Seven fragments and nine crossovers are required, in theory, to create a plasmid.

5.3 Enrichment of Unique Barcode Combinations by FACS

Barcode library diversity was assessed by immunolabeling and flow cytometry of five out of seven of the epitope tags, T7, V5, AcV5, AU5, and E2, due to cytometer constraints. Analysis of protein expression and epitope tag abundances showed that 10% of the expressing cells in the library generated by homologous recombination contained combinations of two or more epitope tags. This subset was enriched using FACS using four sorting strategies to encompass all 32 possible barcode combinations (**Figure 5.2**). After the first round of sorting, 90% of the cells in each library were

enriched for combinations of two or more epitope tags. An additional round of FACS was used to isolate specific epitope tag combinations.

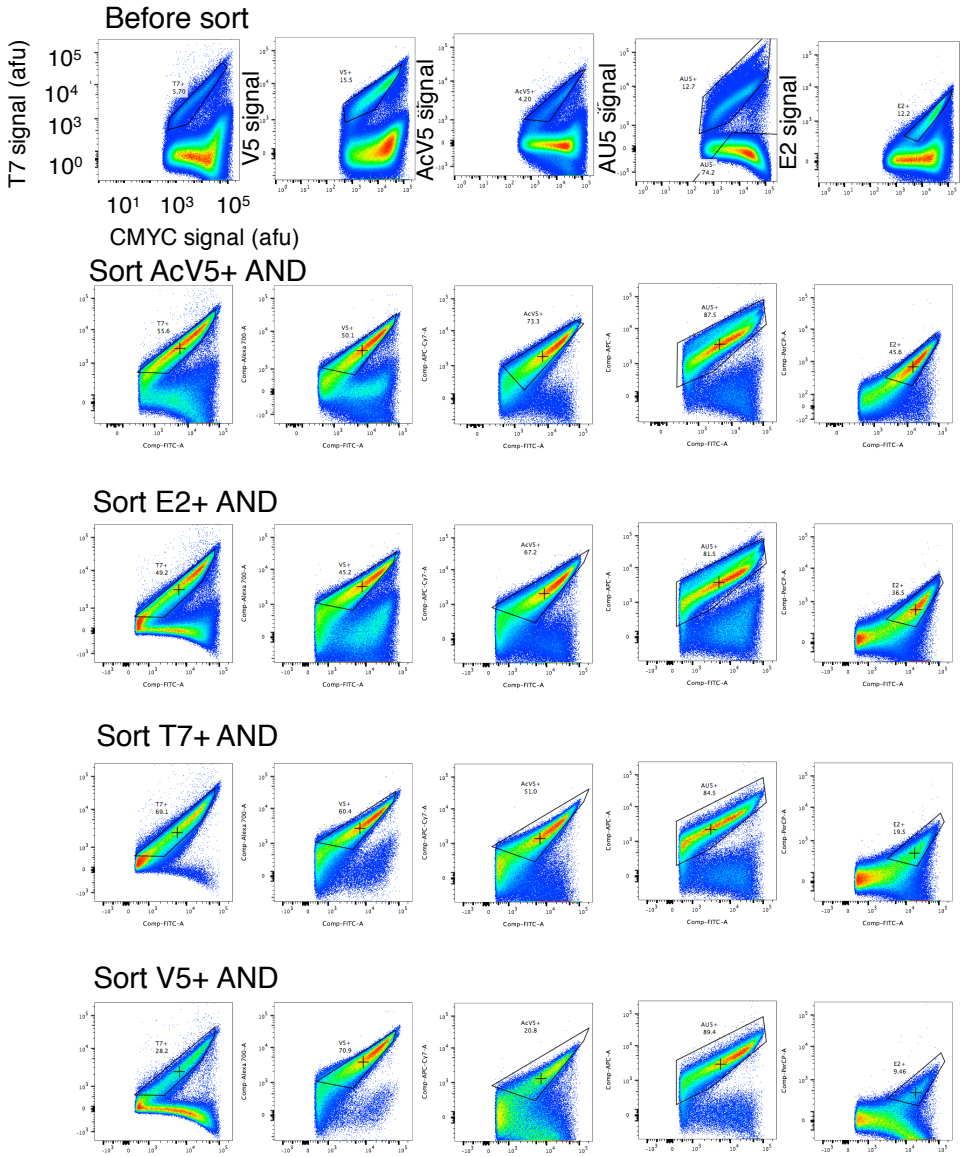


Figure 5.2: Enrichment of combination barcodes by FACS. The barcode library generated by homologous recombination contained combinations of T7, V5, E2, AcV5, and AU5 epitope tags. This subset, which comprised 10% of the expressing cells, was enriched by FACS.

Similarly, the barcode library generated by overlap extension PCR was assessed by immunolabeling and flow cytometry (**Figure 5.3**). This analysis revealed that 0.3% of the expressing cells contained at least one of the T7, V5, AcV5, AU5, or E2 tags. Given the rarity of barcodes containing epitope tags other than cmyc, this subset was enriched in one round of sorting regardless of epitope tag combinations. After one round, 5.5% of the expressing cells contained combinations of two or more epitope tags. Specific barcode combinations ranging from 0.1% to 1% of expressing cells were enriched by a second round of sorting.

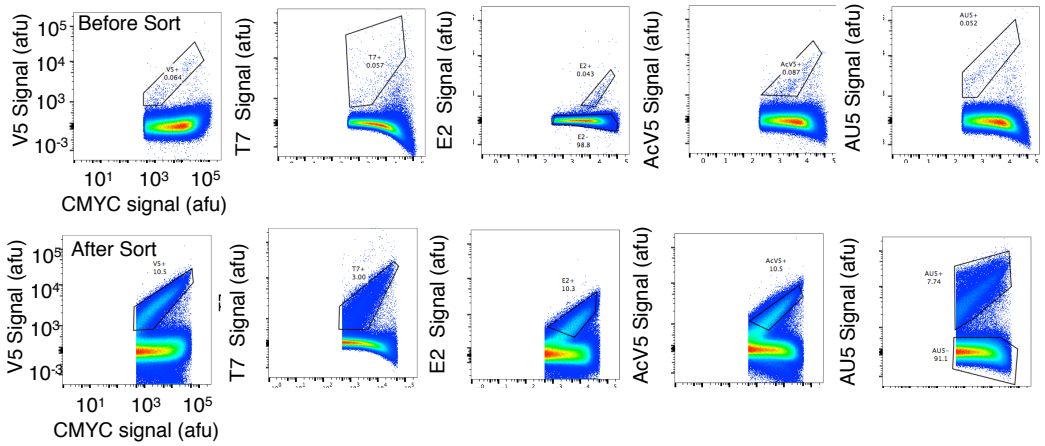


Figure 5.3: FACS enrichment of combination barcodes in overlap PCR library. The unsorted library contained only 0.3% of cells that had epitope tags other than cmyc. After one round of FACS, this subset was enriched to 25%. In the sorted library, 5.5% of the cells expressing barcodes contained combinations of two or more epitope tags. These were isolated using an additional round of FACS.

In total we were able to recover 18 of the 32 possible barcode combinations (**Table 5.2**). After specific barcode combinations were sorted into tubes, the yeast cells

were streaked out onto agar plates to select for single colonies. Colonies were analyzed by immunolabeling and flow cytometry to confirm barcode expression and epitope tag combination were as expected. Then, plasmid DNA was isolated from yeast and transformed into *E coli* for amplification and purification. Recovered plasmids were confirmed for the correct combination of epitope tags by Sanger sequencing.

Table 5.1: T7, V5, AcV5, AU5, and E2 barcode combinations recovered.

T7	V5	AU5	AcV5	E2
0	0	1	1	1
1	0	0	1	1
0	1	0	1	1
1	1	1	0	0
1	1	0	0	1
0	0	1	1	0
1	0	0	1	0
0	1	0	1	0
0	0	1	0	1
0	1	1	0	0
0	1	0	0	1
1	1	0	0	0
0	0	1	0	0
0	0	0	0	1
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	0	0	0	0

5.4 Generation of Unique 11-Epitope Tag Barcode Plasmids and Analysis of Barcode Plasmid Instability

Next, existing barcode plasmids were combined using subcloning in order to expand the number of unique barcodes from hundreds to thousands (**Figure 5.4**). We

created 18 separate libraries, with each library containing a plasmid with a specific combination of T7, V5, AcV5, AU5, and E2 epitope tags as the vector fragment. The 5 barcode libraries containing combinations of HA, HSV, HIS, AU1, GLU, and FLAG epitope tags in different proportions were used as inserts.

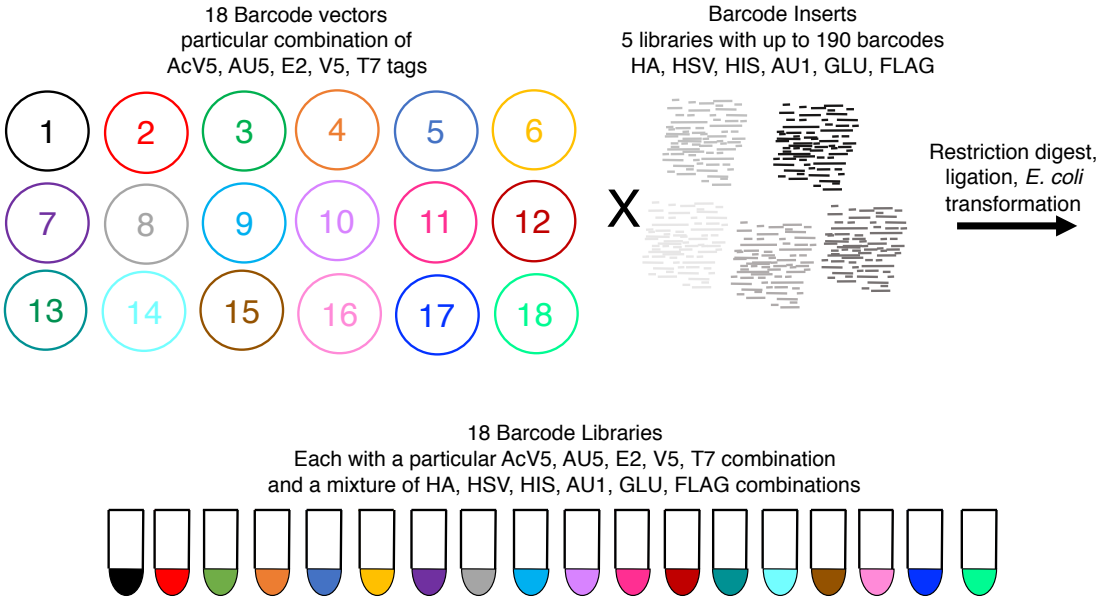


Figure 5.4: Generation of thousands of barcode plasmids. 18 libraries containing up to 190 barcodes each were created using subcloning. A plasmid containing a specific combination of T7, V5, AcV5, AU5, and E2 epitope tags was used as the vector fragment. Five libraries containing different proportions of up to 190 barcodes composed of HA, HSV, HIS, AU1, GLU, and FLAG epitope tags were combined and used as insert fragments.

SpeI and XhoI restriction enzymes were used to generate the DNA fragments, containing the vector fragments with T7, V5, AcV5, AU5, and E2 epitope tag combinations, and the inserts with HA, HSV, HIS, AU1, GLU, and FLAG

combinations and the alpha-agglutinin protein. However, restriction digest and agarose gel electrophoresis of the resulting barcode library DNA showed no full-length barcode plasmids were present. Agarose gel electrophoresis of the HA, HSV, HIS, AU1, GLU, and FLAG libraries showed a heterogeneous mixture of plasmid sizes, ranging from approximately 1-8 kb. Analytical check digest of these libraries showed only a subset of the library DNA contained full-length barcode plasmids, which are expected to be 7-9 kb in length, while a significant portion of the DNA was made up of smaller DNA, which we term mini-plasmids (**Figure 5.5**).

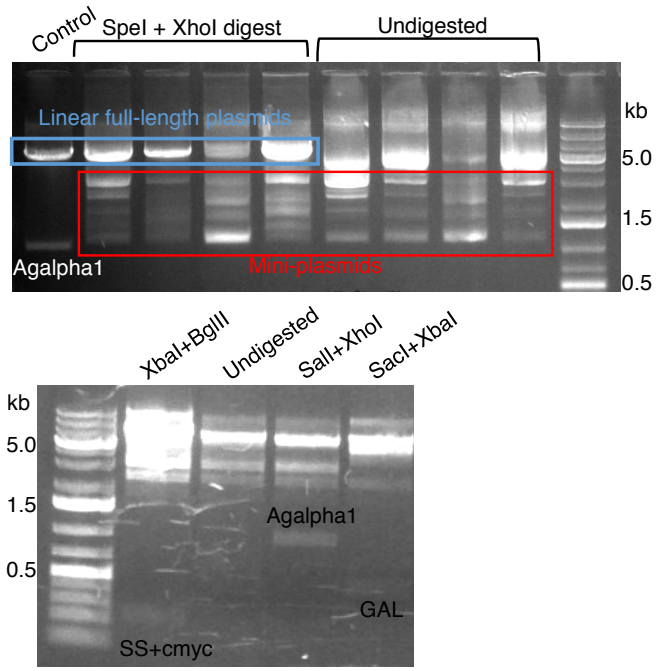


Figure 5.5: Barcode library contains DNA plasmids with heterogeneous sizes. Check digest of the HA, HSV, HIS, AU1, GLU, and FLAG libraries showed only a subset of DNA was composed of full-length barcode plasmids (top panel). Analytical restriction digest (bottom panel) showed a subset of DNA contains elements necessary for barcode expression including secretion signal and cmyc tag, alpha-agglutinin protein, and GAL promoter.

A cocktail of restriction enzymes that digests backbone DNA but not insert DNA containing barcodes was used to prevent mini-plasmids from preferentially transforming *E. coli*, thereby increasing the number of transformants containing new barcode plasmids. Specifically, HA, HSV, HIS, AU1, GLU, and FLAG barcode fragments were excised by SpeI and XhoI digest. Then, the insert DNA was digested with AclI, BstNI, EciI, DraI, and NdeI restriction enzymes to cut up the backbone and mini-plasmids so that they would not be capable of transforming *E. coli*. After ligation of vector and insert fragments and *E. coli* transformation, plasmid DNA was purified and checked for full-length plasmids by analytical check digest (**Figure 5.6**). Analytical check digest of the 11-epitope tag barcode libraries showed that 14 out of 18 libraries contained a significant amount of full-length plasmids, suggesting subcloning of new barcode combinations was successful. Moreover, mini-plasmids were present despite our efforts to eliminate them from the libraries during subcloning.

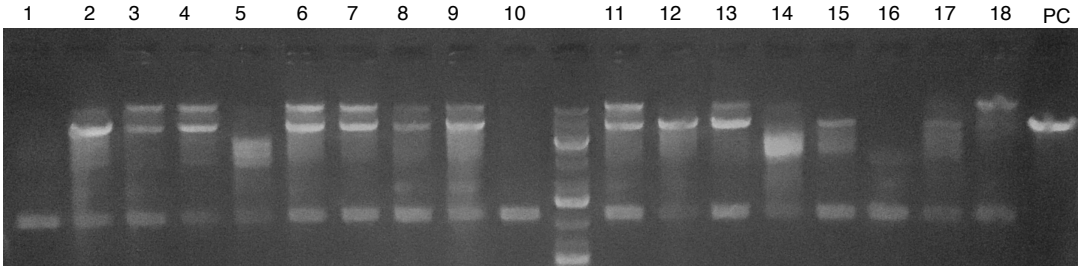


Figure 5.6: Analytical restriction digest of 11-epitope tag barcode plasmid libraries. 18 barcode libraries were checked for full-length DNA by restriction digest with XbaI and agarose gel electrophoresis. This analysis showed that 14 out of 18 libraries (all except libraries 1, 5, 10, and 14) contained a significant amount of full-length plasmids, suggesting that the libraries contain barcodes with new combinations of up to 11 epitope tags. Mini-plasmids also formed in these libraries, despite our efforts to eliminate them during subcloning.

In order to obtain additional insight into the identity and origin of the mini-plasmids, *E. coli* clones containing mini-plasmids were isolated from the 11-epitope tag libraries and their plasmids were analyzed by restriction digest and agarose gel electrophoresis (**Figure 5.7**). The results show that AclI and EciI restriction enzymes cut mini-plasmids in at least two places, resulting in major bands at 2kb and 400bp, and 800bp and 150bp respectively. AclI can cut in up to 5 locations for a full-length barcode plasmid, creating bands containing *E coli* origin and barcode (3.5kb), uracil cassette (URA), CEN/ARS, and partial ampicillin resistance gene (Amp) (2kb), CYC1 terminator and F1 origin (800bp), partial Amp gene (400bp), and partial URA gene (200bp). Also, EciI cuts in up to four places, generating DNA fragments containing barcodes, F1 origin, and partial URA sequences (4kb), partial URA and partial Amp (2kb), partial Amp and *E coli* replication origin (800bp), and partial *E coli* replication origin (150bp). Based on the analytical restriction digest, this suggests mini-plasmids are composed of URA, CEN/ARS, *E coli* origin, and ampicillin genetic elements from the barcode plasmid backbone.

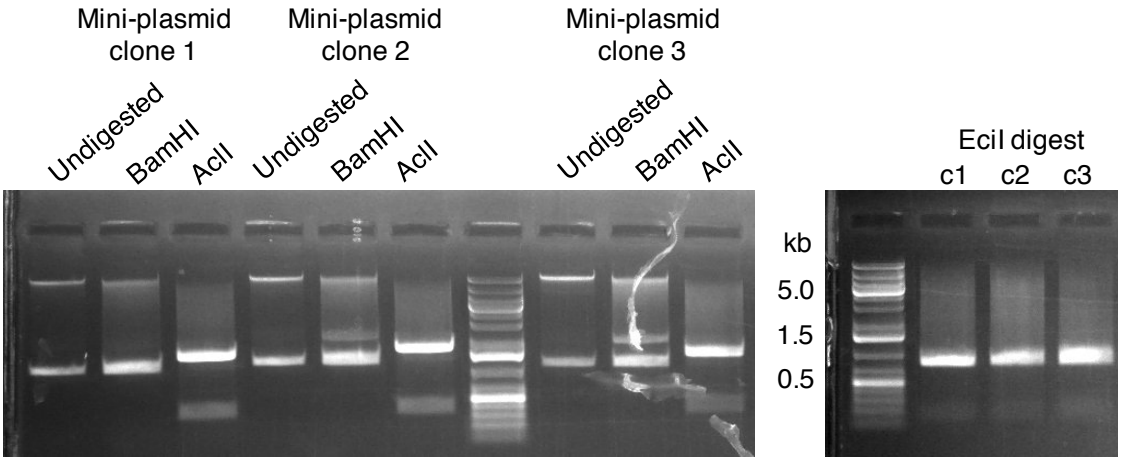


Figure 5.7: Analytical restriction digest of mini-plasmids. Restriction digest with EciI and AclI enzymes, which cut the barcode backbone in multiple locations, suggests miniplasmids are composed of URA, Amp, CEN/ARS, and *E. coli* origin genetic elements.

5.5 Flow Cytometry Analysis of 12-Color Barcode Libraries

The 18 new barcode libraries were assessed by flow cytometry to determine the barcode expression level and abundance of new barcodes containing combinations of up to 11 epitope tags. Each plasmid library was transformed into yeast, barcode expression was induced, and cells were immunolabeled with either a 6-color panel containing CMYC, T7, V5, AcV5, AU5, and E2 antibodies, or a 7-color panel containing CMYC, HA, HSV, HIS, AU1, GLU, and FLAG antibodies (**Figure F.1**). Analysis of the barcode libraries with the 6-color panel shows that 14 out of 18 libraries had a significant fraction (25-50%) of cells with expressed barcode plasmids (**Table 5.3**). Also, all expressing cells contained the expected combination of T7, V5, AcV5, AU5, and E2 epitope tags. The non-expressing cell fraction could be due to a combination of factors including a yeast surface display artifact [64], and mini-

plasmids containing URA and CEN/ARS elements without or with an out-of-frame barcode.

Table 5.2: Analysis of barcode expression in 18 libraries with combinations of up to 11 epitope tags.

Library	% CMYC+ Cells
AcV5	7.5
AU5	41
T7	59.5
E2	50.9
pBC2	1.33
V5	51.3
AU5/V5	45.7
AU5/V5/T7	28
T7/V5/E2	38.6
AcV5/AU5/E2	7.49
T7/V5	52.4
T7/AcV5/E2	42.1
V5/E2	52.8
AU5/E2	4.41
AcV5/V5	26.9
AcV5/V5/E2	46.5
AcV5/AU5	28.5
AcV5/T7	44.8

Flow cytometry analysis of barcode libraries with the 7-color antibody panel against HA, HSV, HIS, AU1, GLU, and FLAG tags allowed quantification of the percentage of barcodes in the libraries containing barcodes with new combinations of up to 11-epitope tags (**Table 5.4**). We found that 85-92% of cells expressing barcodes contained new combinations of up to 11 epitope tags for 13 out of 14 of the libraries, while the AcV5/T7 library contained only 60% new barcodes. This is likely due to incomplete restriction digestion and dephosphorylation of the vector during subcloning.

Table 5.3: Quantification of new barcodes with combinations of up to 11 epitope tags.

Library	% New Barcodes
AU5	89.5
T7	89
E2	88
V5	85.5
AU5/V5	89.5
AU5/V5/T7	85.2
T7/V5/E2	86.1
T7/V5	86.9
T7/AcV5/E2	90.1
V5/E2	87.9
AcV5/V5	86.4
AcV5/V5/E2	91.4
AcV5/AU5	87.4
AcV5/T7	60.9

In addition, barcode identities and their relative abundance were determined for the AU5 library using FlowJo software by manual gating analysis as described earlier (**Table 5.5**). We found that 90 unique barcodes, or 47% of all possible barcodes, were present at 0.01% (100 cells) or more, with relative abundances ranging from 0.01 to 13%. There were 25 barcodes present at 1% or greater in the library, and the barcode containing only T7, V5, AcV5, AU5, and E2 epitope tags was the most abundant.

Table 5.4: Barcodes with the highest abundance present in the AU5 library.

HA	HSV	HIS	AU1	GLU	FLAG	Percent
0	0	0	0	0	0	13.580
0	1	0	0	0	0	8.143
0	0	0	0	1	0	7.240
0	1	0	0	0	1	5.756
0	0	0	1	0	0	5.570
0	1	0	1	0	1	5.024
0	0	0	1	1	0	4.879
4	0	0	0	0	0	3.909
0	0	0	0	0	1	3.756
0	0	0	1	0	1	3.011
0	0	1	0	0	0	2.486
0	4	0	0	0	0	2.475
0	1	0	0	1	1	2.410
0	0	1	0	1	0	2.374
0	0	0	0	1	1	1.925
1	0	0	0	1	0	1.598
0	1	1	1	0	0	1.375
0	0	1	1	0	1	1.362
1	0	1	0	1	0	1.314
0	0	1	1	0	0	1.231
0	1	1	0	0	0	1.167
1	0	0	0	0	0	1.126
0	1	0	1	1	0	0.988
0	1	0	1	0	0	0.964
0	4	0	0	0	1	0.951

5.6 Optimization of a 13-Color Flow Cytometry Panel and Error Analysis

Next, a 13-color flow cytometry panel was developed in order to simultaneously analyze barcodes from all 14 libraries (**Table 5.6 and Table F.1**). Notably, CMYC was analyzed using the AF647 instead of AF488 so that barcodes could be multiplexed with yeast-GFP clones. For applications not requiring the use of this channel, CMYC can be used with AF488 and an additional readout, such as binding affinity, could be measured using AF647.

Table 5.5: 13-color panel used to analyze barcodes and additional variables of interest.

Epitope	Flurophore	Laser	Detector
GLU	Marina Blue	355	450/50
V5	QD525	355	530/30
HIS	QD705	355	670 LP
	GFP	488	530/30
FLAG	PerCP	488	685/35
HA	PE	532	575/25
T7	PE-TexasRed	532	610/20
AU1	PE-Cy5	532	660/20
HSV	PE-Cy5.5	532	710/50
E2 tag	PE-Cy7	532	780/60
cmyc	Alexa Fluor 647	633	660/20
AU5	Alexa Fluor 700	633	730/45
AcV5	APC-Cy7	633	780/60

Initially, saturating antibody concentrations (100nM) CMYC was used for labeling. However, the bright signal from AF647 obscured the signals for AcV5 APC-CY7 and AU5 AF700 due to high spillover, such that positive and negative populations contained too much overlap to be identified accurately (**Figure F.1**). Titration of α-CMYC showed that 1-10 nM gave the most separation between positive and negative AU5 and AcV5 populations. Similarly, spillover from PE fluorophore obscured the PE-Cy5 signal such that AU1 positive events could not be distinguished from the negative. Therefore, HA-PE was titrated as to maximize the distinguishability of HA populations while minimizing the spillover into PE-Cy5 such that AU1 positive populations were not obscured (**Figure 5.7**). It was found that 10nM HA-PE provided the maximum HA-PE signal while allowing 1AU1 populations to be separable from the negative population. In addition, we could not find an appropriate concentration at which HIS-QD705 did not obscure signals in other fluorescent channels, and it was excluded from further analyses.

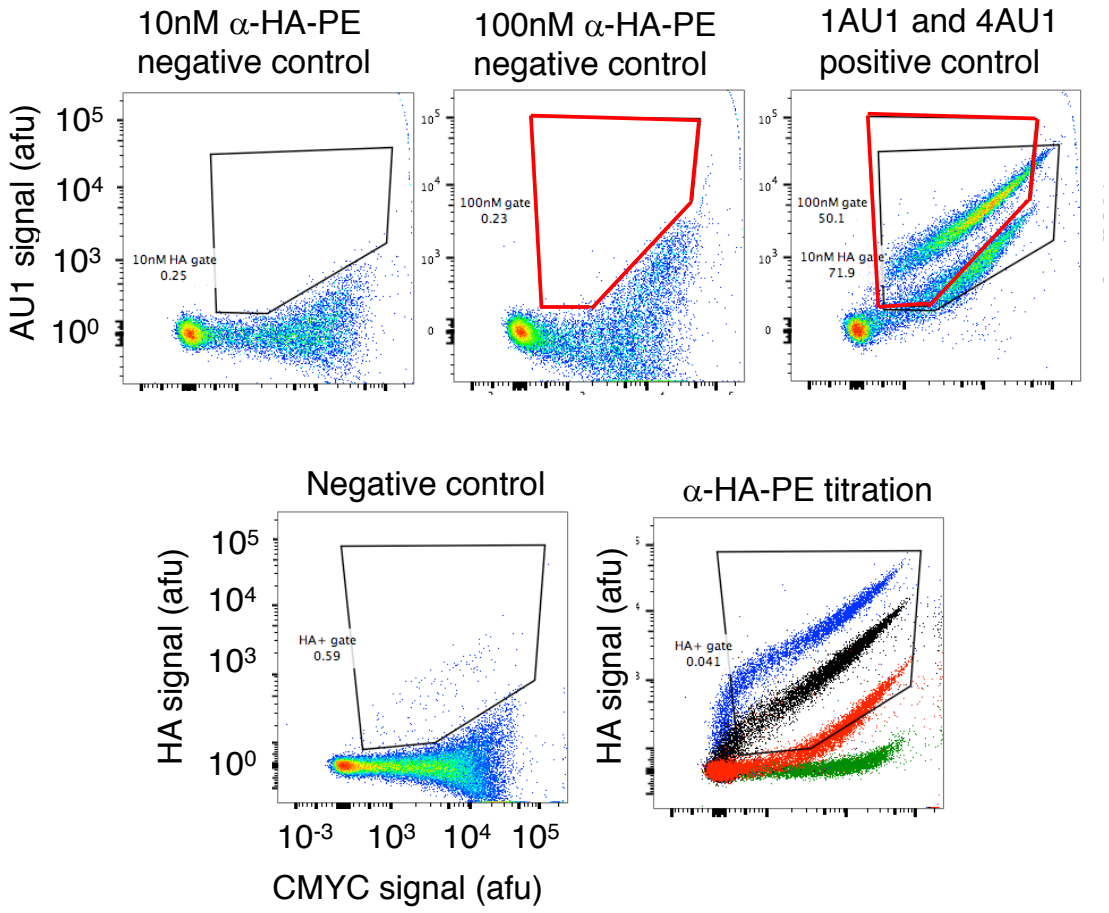


Figure 5.8: Titration of HA-PE to optimize barcode distinguishability for HA and AU1 epitope tags. HA-PE was titrated from 0.1 nM to 100 nM in order to maximize the separation of HA+ and HA- events while minimizing the spillover into PE-Cy5 channel. At 100 nM HA-PE (red box), it was not possible to capture any 1AU1+ cells. When HA-PE was used at 10nM (black box), spillover into PE-Cy5 was lessened, allowing the distinction of AU1+ and AU1- populations.

Next, we sought to assess the distinguishability of positive and negative epitope tag populations with the new antibody fluorophore panel. Yeast expressing known barcode controls were immunolabeled with α-CMYC and one additional epitope tag antibody. Gating analysis was used to quantify the percentage of

incorrectly identified cells and the percentage of cells captured (**Table 5.7**). In general, 15-98% of a barcoded population was captured with 0.6% or fewer false events.

Table 5.6: False positive and cells captured analysis for 12-color panel.

Population	% cells captured	% false positive
AcV5+	59	0.6
AcV5-	14-60	0.4
AU5+	78	0.4
AU5-	38-80	0.2
AU1+	83.5	0.2
AU1-	28-65	0.27
E2+	100	0
E2-	100	0
FLAG+	95	0.1
FLAG-	83-95	0.3
HA+	95	0.3
HA-	75-95	0.5
HSV+	88	0.2
HSV-	50-86	0
T7+	65	0.2
T7-	38-80	0.1
V5+	88	0
V5-	80-95	0.27
GLU+	80	0
GLU-	30-98	0

5.7 Discussion

Homologous recombination of glycine-serine linkers with less than 80% homology to each other were used to create barcodes with combinations of T7, V5, AcV5, AU5, and E2 epitope tags. 18 out of 32, or 50% of all possible epitope tag combinations, were obtained using this approach. Specifically, 6 out of 6 possible barcodes containing single epitope tags were created, as well as 7 out of 10 double, 5 out of 10 triple, 0 out of 5 quadruple, and 0 out of 1 quintuple barcodes. One possible

explanation for why all 32 barcode combinations were not obtained could be because the homologous recombination approach did not work as intended. Although 80% homology was shown previously to be sufficient to prevent unwanted homologous recombination events [87], our results show that this level of homology was insufficient to prevent unwanted recombination events in this case.

We found that glycine-serine linkers recombined regardless of their homology in most plasmids that were sequenced. This could be because the homology was not distributed evenly across the DNA due to glycine only being encoded by GGN. It is possible that glycine-serine linker DNA recombined, regardless of their dissimilarity because of the large stretches of guanine nucleotides. In addition, the high number of crossover events required to generate barcode plasmid combinations hindered barcode creation. It is likely that we obtained few barcodes with three epitope tags, and no barcodes with four or more epitope tags because of the requirement for more crossover events to occur.

Homologous recombination could be a more successful approach to barcode generation if the number of epitope tags used and thus crossovers required were lessened. Our results support this idea, because a greater variety of barcode combinations were obtained when partially digested epitope tag linker fragments were used instead of fully digested fragments to create a barcode library. Only barcodes containing zero or single epitope tags were found when DNA containing single epitope tags surrounded by linkers, and thus requiring nine crossovers, were used. However, diverse barcode combinations were found when partially digested inserts were used, which likely required 2-4 crossovers to generate a full plasmid.

It may be possible to achieve a higher rate of successful combinations by altering the method used to generate combinations. For example, Golden Gate cloning has been used to assemble up to nine fragments simultaneously with 85-90% efficiency [90], [91] and a similar efficiency for 5 to 6 tandem repeat fragments [92]. In addition, Gibson assembly may lead to greater success for assembling epitope tag combinations than an overlap extension PCR approach because dsDNA fragments could be assembled without random reannealing of ssDNA fragments [93].

The probability of successful epitope tag combinations could also be improved by designing a system that relies on specific recombination. One approach could be to use different types of flexible linkers instead of only glycine-serine linkers for a homologous recombination, Gibson assembly, or overlap extension PCR approach [99]. Alternatively, Golden Gate assembly could be used for specific recombination by designing unique 4-base pair overhangs to combine epitope tags together.

Full-length plasmids containing barcodes with combinations of up to 11-epitope tags libraries were successfully created after mini-plasmids were prevented from transforming *E coli* cells by restriction digest of barcode backbone fragments. Mini-plasmid removal from subcloning reactions was essential to permit subcloning of new barcodes, likely due to their increased transformation efficiency and small size. Our results show that mini-plasmids were made up of DNA fragments from barcode backbone plasmids and can contain yeast and *E. coli* replication elements as well as genes or gene fragments for selective growth in ampicillin and uracil dropout media. Another interesting observation is that mini-plasmids persisted in the new barcode libraries despite being fragmented during subcloning. This suggests that mini-plasmids form during cell growth due to barcode plasmid replication instability in *E. coli* cells,

likely due to the long tandem repeat DNA regions. Interestingly, plasmid instability was not observed in yeast cells, which could be due to increased fidelity of eukaryotic transcription machinery [100].

There are a number of potential solutions to remove mini-plasmids from the barcode library or prevent their formation. If mini-plasmids are not removed by selection, it could also be possible to use FACS or magnetic-bead sorting to remove yeast cells that are not expressing barcodes due to harboring of a mini-plasmid. In order to prevent the formation of mini-plasmids, barcodes could be redesigned so that other flexible linkers aside from $(G_4S)_3$ are used to decrease the length of DNA repeats or a shorter linker could be used. It is also possible that mini-plasmids will be less likely to form if epitope tag repeats are not used, as we did not observe any mini-plasmids with T7, V5, AcV5, AU5, and E2 barcodes which only contained single repeats. Alternatively, different recombinase deficient *E. coli* strains could be tested to see if mini-plasmid formation is lessened. This approach has shown to be successful in some cases [71].

Flow cytometry analysis of barcode distribution in the 11-epitope tag libraries showed that barcodes ranged in abundance over four orders of magnitude. Some possible explanations for the wide distribution include experimental error in handling small amounts of DNA, a greater propensity for smaller barcode plasmids to transform cells, and a higher likelihood for larger barcode plasmids with long repeats to be deleted due to erroneous transcription. Bias in barcode abundances could be corrected by FACS as demonstrated earlier. Also, barcodes could be redesigned with spacer regions such that their length and plasmid size is more uniform.

Chapter 6

DEVELOPMENT OF SOFTWARE FOR RAPID BARCODE IDENTIFICATION AND ABUDANCE QUANTIFICATION

6.1 Introduction

Flow cytometry can provide informative, high-dimensional data about cellular structure function, as current advanced flow cytometers can measure up to 20 parameters at once [65]. Manual analysis of multidimensional flow cytometry data involves gating or grouping cells into discrete populations based on their fluorescence and size characteristics. This approach suffers from a number of drawbacks due to its cumbersome and time-consuming nature for complex data sets. For example, analysis of 16 markers from 130 patent leukocyte samples took more than 15 hours to complete [101].

The primary advantage of computational approaches to flow cytometry data analysis is that they can decrease analysis time in some cases from hours to minutes. Dozens of open-source software packages for computationally assisted analysis of flow cytometry data are available [101]–[105]. These software packages apply mathematical models to segment data, of which the three most common approaches are centroid based clustering (k-means), density based clustering, and distribution model-based clustering. K-means clustering is very fast but requires a priori information about the central location and number of clusters, and can only cluster populations with spherical shapes [106]. Distribution based mixture modeling, the most commonly used being Gaussian, typically are more robust than k-means, but

their utility is limited due to necessity of making assumptions about the distribution of the data and the number of mixture components [105], [107]. Bayesian Information Criterion or Akaike Information Criterion can be used to determine the best mixture model fit and estimate the number of clusters without prior information [108]. Some methods use both k-means to partition the data and distribution-based models for clustering [109]. Density based clustering can be used to overcome limitations requiring a priori information about the location and number of clusters, or about the underlying distribution of the data, but in some cases cannot be applied to high-dimensional data and require use of a subspace [108], [110].

In this chapter, new software for rapid identification and quantification of barcode abundances from flow cytometry data is presented. Density-based clustering using the DBSCAN algorithm [111] and Gaussian kernel density estimation [112] were used to cluster cells with similar fluorescence properties and estimate barcode abundances. Barcode identities were elucidated by comparing minimum and maximum cellular fluorescence values within a cluster. In addition, the accuracy of our approach was calculated using many data sets, and the total number of barcodes created and their relative abundances were estimated computationally.

6.2 Overview of Computational Barcode Identification Method

Analysis of barcode flow cytometry data for identification and quantification of ~200 populations took approximately four hours to complete for one sample and required more than 200 unique gates. Therefore, it would be cumbersome to analyze a large number of samples using a manual approach. To automate and hasten data analysis, we developed a computational method to identify and quantify barcodes from flow cytometry data (**Figure 6.1**). First, data was gated in FlowJo to exclude

cells whose fluorescence could not be assigned as belonging to a particular population for a given color. This process required a maximum of 23 gates and took less than 30 minutes for approximately ten samples. After gating, samples were analyzed with a Python script to identify and quantify barcode populations. First, fluorescence data was transformed from linear to log space and cells with negative fluorescence values were assigned a positive fluorescence value based on the median fluorescence of the positive events in the negative population for a particular fluorophore. Cells with negative fluorescence have little to no fluorescence, always belong to the negative population, and arise due to instrument baseline subtraction and compensation error [113].

The software uses the DBSCAN clustering algorithm and Gaussian kernel density estimation to perform the analysis. DBSCAN is a density-based clustering algorithm which clusters together nearest neighbors, and relies on two parameters, the maximum search distance to look for additional nearest neighbors and the minimum number of nearest neighbors required to assign a core point [111]. Using this approach, DBSCAN can cluster groups together regardless of their shape, which is necessary in this case. We also first tried implementing k-means because it is faster than DBSCAN, but the approach did not work based on its requirement for data to be centroid. Gaussian kernel density estimation (KDE) is a method that can estimate the probability density function of data based on a Gaussian model [114]. KDE assigns a density score to each data point, and in this case lower density areas were excluded from the analysis to decrease noise in the data.

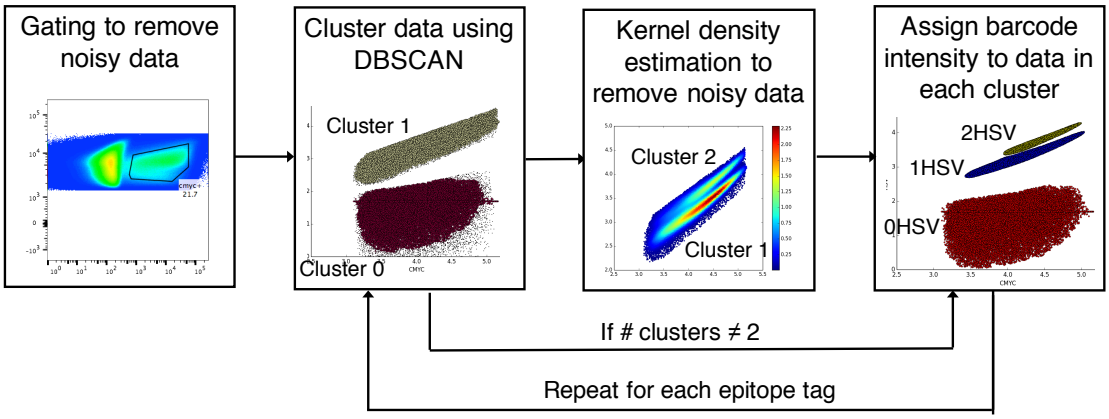
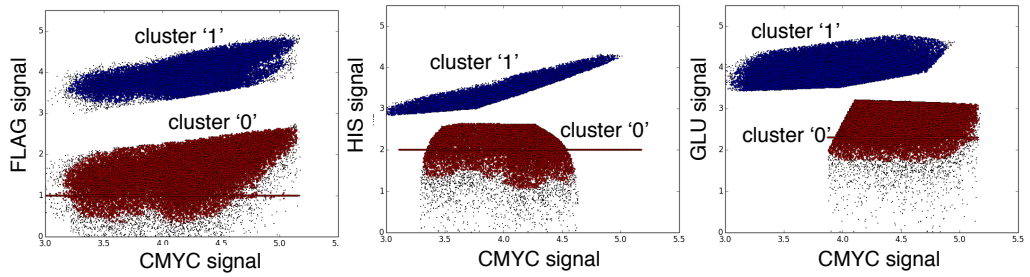


Figure 6.1: Computational approach to barcode identification and abundance quantification. Flow cytometry data was gated to exclude cells whose fluorescence could not be assigned to a particular population for any given color. Then, data was analyzed using a Python script which used DBSCAN clustering to assign cells to one of two clusters based on the density of nearest neighbors. This assignment process was repeated for each epitope tag that produced 'binary' intensities when immunolabeled. Barcodes were partitioned based on their binary fluorescence intensities, and these groups were analyzed by DBSCAN for epitopes that when immunolabeled produced up to three distinct populations. DBSCAN assigned cells in to up to three clusters (one negative and two positive). For cases in which two clusters were assigned, KDE was used to remove outlying cells and DBSCAN was re-run to assess if one or two populations with positive fluorescence intensities were present. Finally, cells were grouped by an 11-digit identifier corresponding to their barcode (0 for negative, 1 for low positive, and 2 for high positive) and the number of cells belonging to that barcode was quantified and normalized to calculate relative abundances. Information was exported to Excel for further analysis if required.

The DBSCAN algorithm was used to identify cells as belonging to one of two groups or clusters for epitope tags which produced binary populations when immunolabeled (binary epitope tag). For each epitope tag or fluorophore, cells were assigned a number, either '0' or '1' which indicated which cluster they were assigned to (**Figure 6.2**). Then, minimum and maximum fluorescence values were used to

calculate which cluster and cells should be assigned ‘0’ indicating negative fluorescence and ‘1’ indicating positive fluorescence. This approach was repeated for each binary epitope tag and then cells were grouped together based on their binary barcode ID. A binary barcode ID is an eight-digit identifier that represents a cell’s assignment to eight epitope tag populations. For example, the binary barcode ID ‘01010101’ represents an assignment to the negative population for T7, AU5, E2, and GLU epitope tags and to the positive population for V5, AcV5, HIS, and FLAG epitope tags.



Example statistics

epitope is: GLU
cluster ID is: 1
minimum value: 3.409
maximum value: 4.798
mean value: 3.989
standard deviation: 0.275
cluster abundance is: 26.827 %
cluster ID is: 0
minimum value: 1.284
maximum value: 3.355
mean value: 2.789
standard deviation: 0.292
Cluster abundance is: 72.75 %

Example output

‘Binary’ Barcode ID : row numbers corresponding to data points (cells) in that population
“00000101":[0,8,20,25,61,122,126,154,178,180,215,226,250...]
"00000100":[34,38,41,55,70,72,109,135,137,148,160,169,184...]
"00000011":[30,43,59,63,78,91,107,145,156,192,230,243,245...]

Figure 6.2: Assignment of cells to one of two clusters using DBSCAN. DBSCAN uses a nearest neighbor density based approach to group cells together. In this case, it was used to identify cells as belonging to one of two populations for each binary fluorophore. The python script also calculated statistics for each cluster which were subsequently used to determine which cluster should be assigned ‘0’ and ‘1’. After all binary epitope tags were assigned, cells were grouped by their binary barcode identity.

Next, cell subpopulations grouped by their binary ID were analyzed by DBSCAN for the three epitope tags that produced multiple intensities when immunolabeled, specifically HSV, AU1, and HA. Binary ID clusters were first segmented based on their HSV population, and then by HA and AU1 in parallel because HSV had lower fluorescence variability than HA or AU1 as determined by manual analysis. DBSCAN was used to cluster cells into one, two, or three groups labeled '0', '1', and '2'. Then, minimum and maximum statistics were used to determine the correct cluster labels for each cell, representing negative fluorescence or '0', low positive fluorescence or '1', and high positive fluorescence or '2'. In the case in which two clusters were found, it was possible that there were two positive barcode populations present. However, cellular fluorescence variability may be obscuring the populations from DBSCAN as the algorithm requires there to be a significant decrease in density between two clusters to identify them as separate groups (**Figure 6.3**).

Thus, cells assigned to the '1' cluster were subjected to KDE which mapped them to an approximate probability density function and cells located in regions of low density were excluded. Exclusion criterion was based on a fraction of the maximum density value calculated by KDE and were determined empirically. The exclusion cutoff was dependent on the particular epitope tag, total number of data points in the cluster, and the maximum relative density value assigned by KDE. For example, clusters with a low number of cells required a less strict filter than clusters a large number of cells (>10,000). Then, DBSCAN was executed again on this filtered data subset in order to cluster the cells into two groups. Statistics were used as before to assign cells to either a '1' or low positive or '2' or high positive cluster.

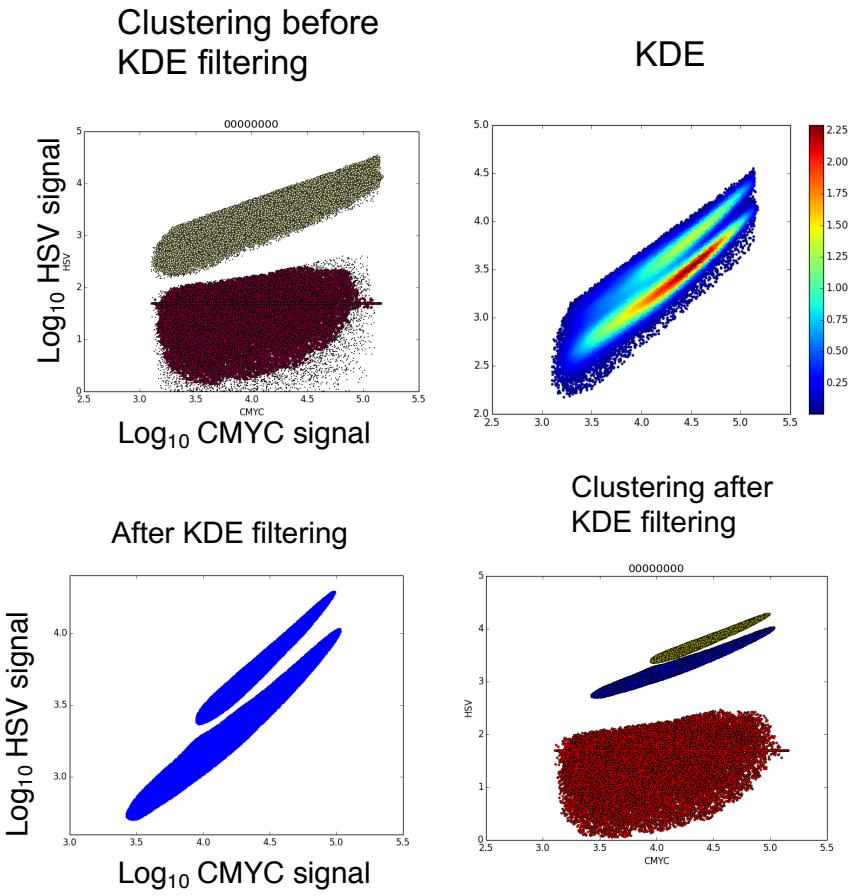


Figure 6.3: Clustering cells for epitopes with multiple fluorescence intensities by DBSCAN and KDE. Cells were clustered with DBSCAN to determine negative and positive populations. If two populations were found, KDE was applied to filter out cells in areas of low relative density. Lastly, DBSCAN was used again to assign cells to low positive and high positive clusters.

Lastly, cells were grouped by their 11-digit barcode ID identifier representing their assignment to each epitope tag. For example, the barcode ID '1110021011' represents cells with positive T7, V5, and AU5 fluorescence, negative AcV5 and E2 fluorescence, high positive HA fluorescence, low positive HSV fluorescence, negative

HIS fluorescence, and positive GLU and FLAG fluorescence. The relative abundance of each barcode was determined by counting the number of cells assigned to each barcode ID and normalizing by the total number of cells in all clusters.

6.3 Establishment of Filtering Criteria and Analysis of Software Accuracy

Control samples containing barcodes with known identities and abundances were used to assess the accuracy of the software and to determine filtering criteria for excluding false barcodes. For example, a sample containing 10 barcodes was analyzed using the approach described above, and the results of this analysis are shown in **Table 6.1**. The algorithm returned 16 barcodes, meaning in this case six of them were false. After analyzing the software results versus those obtained by manual inspection, we determined that a criterion of excluding barcodes present at abundances 100-fold below the expected value could be applied to remove false barcodes, or below 0.1% abundance in this case.

Table 6.1: Barcode identities and abundances found by software for control sample containing 10 barcodes. Note that false barcodes are highlighted in red.

Barcode	Number points	Abundance
00000000000	44753	50.84
00000000100	8638	9.81
00001000000	7964	9.05
00010000000	7536	8.56
00100000000	6603	7.50
10000000000	4052	4.60
00000000001	3650	4.15
00000000010	2190	2.49
00000010000	1882	2.14
00000100000	686	0.78
10000100000	34	0.04
00100000001	15	0.02
00010000100	8	0.01
00100000100	7	0.01
00100010000	2	0.00
10001000000	1	0.00

In addition, barcode identification software accuracy and associated types of error were quantified by manual inspection of DBSCAN clustering outputs. Specifically, data for each DBSCAN clustering was graphically displayed and inspected manually for errors. The types of errors we found were missed barcodes due to insufficient cell density, mislabeled 'high intensity' barcodes due to the absence of a lower positive intensity cluster, and falsely identified barcodes (**Figure 6.4**). False barcodes encompassed those clusters that were identified as negative or '0' but were actually '1' or positive, populations that were erroneously clustered together, and segmentation of one population into two or more clusters.

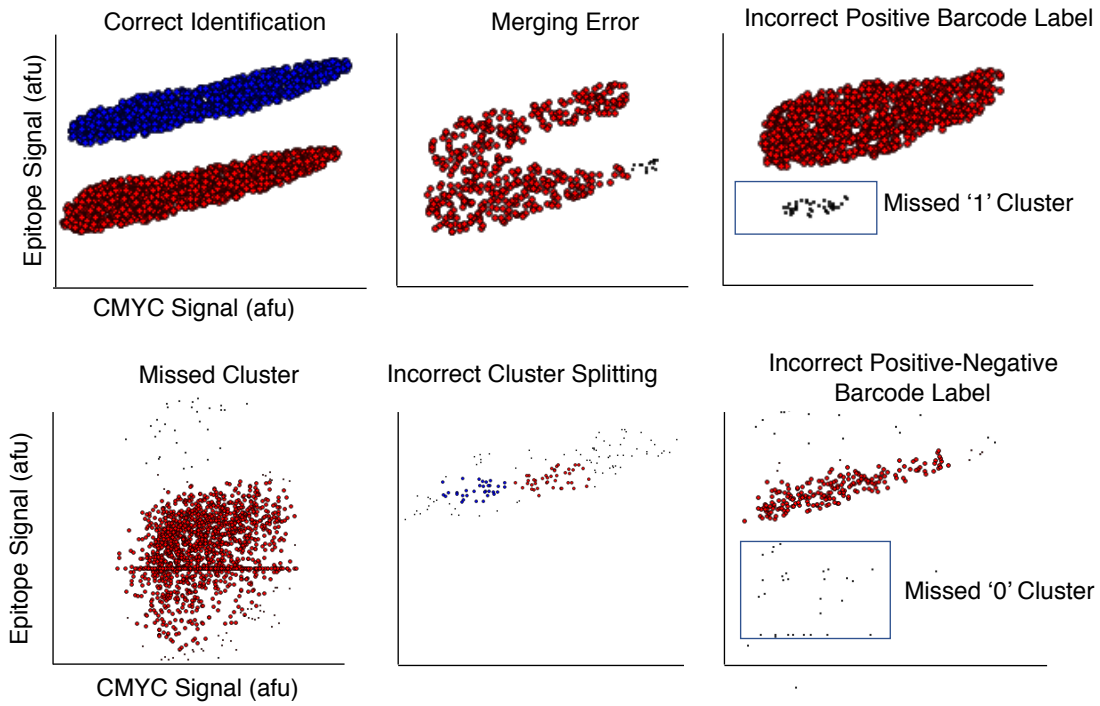


Figure 6.4: Types of errors encountered during barcode identification and clustering. DBSCAN clustering was used to identify populations for each epitope tag present in the sample. This information was compiled for all tags to determine the number of barcodes present in the samples. Manual inspection of DBSCAN outputs showed five types of errors for cluster identification, including merging of two populations, incorrect labeling of positive clusters, missed clusters, incorrectly partitioned clusters, and positive clusters incorrectly identified as negative or vice versa.

For each of the 14 libraries, error analysis of the number of falsely identified barcodes before and after filtering criterion were applied, as well as quantification of mislabeled or missed barcode populations are shown in **Table 6.2**. The number of barcodes found by the software in each library after filtering ranged from 35-79 with an average of 57 barcodes. Of the barcodes found, on average 2 barcodes were mislabeled, 4-5 were falsely identified before filtering and 1 was falsely identified

after filtering, indicating filtering criterion were effective in eliminating most false barcodes from the analysis.

Table 6.2: Analysis of barcode identification software accuracy and error assessment.

Library	# False Barcodes Before Filtering	# False Barcodes After Filtering	# Mislabeled Barcodes	# Missed Barcodes	# of Barcodes After Filtering
00100	9	4	2	5-21	74
10000	4	0	2	14-48	79
01000	9	1	2	10-42	64
00001	7	2	2	17-57	63
11100	4	3	2	13-69	41
01100	10	4	3	8-36	64
11001	7	2	3	8-36	53
11000	5	0	3	14-48	55
10011	1	0	0	5-21	55
01001	2	0	0	5-27	71
01010	2	1	5	13-45	56
01011	0	0	4	17-72	36
00110	4	1	2	13-39	44
10010	2	1	1	12-48	35

An overall summary of barcode software accuracy and an estimation of the total number of barcodes is shown in **Table 6.3**. We estimate that 16-43% (100-600) barcodes were missed based on manual inspection of the number of clusters missed by the software. On average, we had a low rate of misidentification of barcodes after filtering criterion were applied, with 2.4% falsely identified barcodes and 4% mislabeled barcodes. Based on the number of barcodes found by the software and an estimation of the number of missed barcodes, we estimate that between 1,115 and 1,570 barcodes were created in total.

Table 6.3: Summary of barcode identification software accuracy and estimation of total barcodes

	Count	Percentage
Min barcodes missed (software)	154	16.31
Max barcodes missed (software)	609	43.53
False barcodes	19	2.41
Mislabeled barcodes	31	3.92
Total barcodes found	**980**	
Estimated total barcodes	**1115 - 1570**	

6.4 Discussion

We developed software to identify barcodes and quantify their relative abundances from flow cytometry data. The major advantage to using a computational approach to analyze multicolor flow cytometry data, in this case, is that the time taken to analyze samples was decreased from hours to minutes. Specifically, one sample took about four hours to analyze manually, but only ~25 minutes with a computational approach. Overall, it is worthwhile to use computational approaches for large flow cytometry data sets with multiple colors and complex gating schemes, as they are faster than manual approaches.

However, the computational approach to barcodes identification has some drawbacks including the time-consuming optimization of DBSCAN and KDE parameters and less accuracy than the manual approach. There are a number of improvements that can be made to the software which could decrease the error rate and the number of missed or misidentified barcodes, such as the use of more robust barcode exclusion criteria. In addition, analysis could be further automated by not require manual pre-filtering. Currently, the software sacrifices identification of all present barcode populations for a low error rate of false barcode identification.

In order to improve the accuracy of the software, one approach could be to use data sets with larger amounts of cells, and thus would be amenable to more stringent filtering criteria without rare barcode population loss. Acquiring larger data sets is feable as flow cytometry can measure ~10^8 cells per hour. This could have multiple positive effects including a decreased number of missed or mislabeled barcode populations and a decreased number of barcode splitting events due to insufficient cell density within a cluster. More stringent filtering could also decrease the number of false barcodes by enhancing separation between clusters and thus lower the probability of cluster merging.

One drawback of the software is that it cannot label a cluster as a low positive or high positive cluster in the absolute case, meaning without the presence of other clusters. Rather, the software calculates the maximum value present in each cluster and then compares them to assign labels. In most cases, mislabeled clusters are not an issue because negative clusters are the most abundant in the barcode libraries, followed by low positive and high positive clusters. An alternative approach to assigning cluster labels could be used including simply inputting a range of fluorescence values in which negative, low positive, and high positive clusters could occur. This approach would work well for epitope tags that produce binary populations when immunolabeled. However, this is not a simple task for epitope tags that produce multiple intensities when immunolabeled due to the interdependencies between other epitope tags and the fluorescence as discussed in Chapter 3. Ideally, machine learning could be used with training data to determine the optimal parameters for multiple intensity epitope tags, taking into account the presence or absence of other epitope tags.

We have identified a critical exclusion criterion, the relative abundance of a barcode, as essential to computational barcode identification with a low error rate. In addition, it would be advantageous to identify additional exclusion criteria in order to improve the accuracy of the software. For example, statistics on cluster fluorescence intensity and deviation could be used to exclude cells in addition to the abundance criterion. Also, additional control samples with defined numbers of barcodes should be analyzed computationally to determine if the abundance exclusion criterion is sufficient to exclude false barcodes in all cases or to help identify additional exclusion criteria. Lastly, a disadvantage of our computational approach compared to others is that it requires pre-filtering data using manual gating to accurately identify barcode populations. If KDE is applied to all epitope tags instead of only those that produce barcodes with multiple intensities, it could eliminate the need for manual gating and decrease the time needed to analyze flow cytometry data as well as the need for manual analysis.

Chapter 7

APPLICATION OF FLUORESCENT BARCODING FOR MULTIPLEXED ANALYSIS OF BIOMOLECULAR AND CELLULAR LIBRARIES

7.1 Introduction

In this chapter, application of the fluorescent barcoding system to the multiplexed analysis of biomolecular and cellular libraries is presented. First, application of barcoding scFvs for the multiplexed, quantitative analysis of protein-protein interactions is discussed. Towards this goal, we determined the effect of barcode expression on the affinity of the α-prion scFv ICSM18 2.6.1 for recombinant prion protein, and discuss the production and characterization of recombinant prion protein.

In a second application, the fluorescent barcoding system was applied to study the dynamic behavior of yeast GFP protein fusion clones in different environments. Specifically, we assigned genetically-encoded unique fluorescent barcodes to yeast GFP fusion clones and evaluate possible factors influencing barcode expression. In addition, we simultaneously examined the single-cell protein expression dynamics of barcoded yeast GFP clones in response to ten different environmental conditions and over a range of times. We found interesting responses including changes in protein abundance, variation, and distribution, which may suggest that cells use a bet-hedging strategy for enhanced fitness in fluctuating environments. In addition, proteins of unknown function were studied, and changes in their expression profiles due to environmental perturbation potentially indicate functional significance.

7.2 Prion Diseases and Potential Antibody Therapeutics

Prion diseases are invariably fatal neurodegenerative diseases that cause abnormal protein folding, which results in dementia and other debilitating symptoms [115]. Prion diseases can be inherited in 15% of cases, sporadic, or acquired through tainted bovine consumption, cannibalism, or contaminated surgical equipment [116]. Prion disease is caused by the introduction of the disease-causing, misfolded prion protein (PrP^{Sc}). Misfolded prion protein converts the native, membrane bound alpha-helical prion protein (PrP^{C}) into a disease causing, beta-sheet rich form (PrP^{Sc}) [117]. The conversion process from an alpha-helical form to a beta-sheet rich form is unique in that it is not genetic in nature, but rather is caused by interaction of the PrP^{Sc} protein with PrP^{C} protein [118], [119]. In support of this, knockout mice that do not express normal prion protein do not develop prion disease [120]. Misfolded prion protein often forms aggregate and amyloid structures, which cause neuronal death and can be seen as deposits in the brains of affected patients [121].

Cell culture and mouse model studies have shown that antibodies targeting the normally folded prion protein may be able to interrupt the conversion process [122]–[125]. While antibodies have shown limited therapeutic efficacy in mouse models by injection or viral delivery [126], [127], none have been successful in human trials, possibly due to difficulty of accumulation in the brain due to the blood brain barrier. Engineering techniques can be used to improve affinity and stability, and could result in α-prion antibodies with increased therapeutic efficacy. Engineering approaches for improved antibody therapeutics have shown promise in treatments for other disease including Alzheimer's disease [128], rheumatoid arthritis [129], and respiratory syncytial virus [130].

In our laboratory, a number of α-prion single-chain variable fragments (scFvs) have been engineered for increased affinity and stability. scFvs are protein fusions composed of the variable heavy (VH) and variable light (VL) portions of an antibody, connected by a designed flexible linker [131]. They are advantageous for engineering and surface-display approaches because they are small (~25 kDa) and composed of a single domain. Variants of the α-prion antibody ICSM18 [126] were engineered in our laboratory using random mutagenesis and yeast surface display.

The clone ICSM18 2.6.1 was found by other lab members to have increased stability and soluble CHO expressed yields as compared to wild type ICSM18. Improvements in the stability of ICSM18 2.6.1 can be attributed to three amino acid substitutions in framework regions of the immunoglobulin domains (R91G, M77V, M21I), a mutation in the flexible linker (G116D), and four silent mutations leading to improved yeast codon usage. The R91G mutation may have reduced steric hindrance near the binding pocket, allowing for increased contact between the scFv and PrP. The methionine substitutions to valine and isoleucine are smaller and higher on the hydrophobic index. These properties may improve stability by permitting greater burial into the hydrophobic protein core. The linker amino acid substitution to a charged residue may have helped improve linker stability and solubility [99].

7.2.1 Protein-Protein Interactions

Protein-protein interactions, including receptor-ligand interactions, underlie many important biological processes including signal transduction and membrane transport. Aberrant protein-protein interactions play a central role in many diseases, including Creutzfeld-Jakob disease (CJD), Alzheimer's disease, and Huntington's disease [132]. Protein-protein interaction strength is governed by electrostatic forces

including hydrogen bonding and Van der Waals interactions. Protein-protein interactions are stabilized by interaction of hydrophobic, complementary faces with at least 600 square angstroms of buried surface area [133]. Studies on the specific contribution of amino acids to protein interaction interfaces found that tryptophan, tyrosine, and arginine occurred most frequently, possibly due to their ability to form multiple types of interactions including pi-interactions, hydrogen bonding, hydrophobic interactions, and salt bridges [134].

The strength of a protein-protein interaction is related to the free energy of binding:

$$\Delta G = -RTlnK_d$$

where R is the ideal gas constant, T is temperature, and K_d is the equilibrium dissociation constant. The dissociation constant for equilibrium interaction is dictated by the ratio of the on and off rates of binding, k_{off}/k_{on}. There are many methods to estimate the strength of a protein-protein interaction. Techniques are based on measurements of kinetic parameters, which include surface-plasmon resonance [135] and isothermal calorimetry [136], and equilibrium measurements, such as antigen titration [51]. Many quantitative and non-quantitative methods to screen for protein-protein interactions exist, including affinity purification and mass spectrometry [137], FRET [138], proximity ligation assay [139], [140], and yeast two hybrid [141].

7.2.2 Single-Cell analysis of the *S. cerevisiae* Proteome in Fluctuating Environments

Cellular heterogeneity within cell isogenic populations is caused by noise in gene expression due to the stochasticity of biochemical processes involving small numbers of molecules [4], [32]. This heterogeneity is often masked using traditional

methods such as Western blotting or mass spectrometry. Phenotypic heterogeneity in cell signaling and protein expression has been observed in a variety of cell types and affects many important phenomena including cellular fitness improvements during environmental fluctuations [9], [10], [142], differential response of cancer cell subpopulations to drugs [45], [143], [144] , stem-cell lineage [145], and bacterial persistence [7]. For example, over 1000 clones derived from lung carcinoma, with each clone expressing a different endogenous fluorescent protein fusion, were examined during response to the drug camptothecin by time-lapse microscopy [46]. Some of the protein fusions exhibited bimodal expression patterns, which suggest they may play a role in cell subpopulation's escape from drug action. In another study, time-lapse microscopy of clonal yeast populations exhibited a range of growth rates, with slow growing phenotypes correlating with resistance to heat killing and higher expression of the trehalose biosynthesis protein Tsl1 [9].

Typically, single-cell proteomic analysis studies use high-throughput tools such as automated fluorescence microscopy or flow cytometry to measure the response of cells expressing fluorescent protein fusions under endogenous promoters. To facilitate studies of proteome dynamics, a collection of over 4,000 yeast clones representing 75% of the yeast proteome was created, with each clone expressing a GFP fusion from the native open reading frame [55]. Using this collection, 70% of proteins with unknown localization were assigned into 22 distinct subcellular compartments. Since this study, the GFP fusion collection has been widely used to assess the yeast proteome in normal environments [41], and in response to perturbations including DNA damage agents such as hydroxyurea, methyl

methanesulfonate, and UV irradiation [13], [67], [68], osmotic stress [146], and reducing, oxidizing, and heat stress [10].

A study of the yeast proteome in rich and minimal media using flow cytometry found that proteins expressed at the same median level can have different amounts of noise, or variance in protein expression from cell to cell [41], and that protein expression noise is likely predominantly affected by the stochastic production and destruction of mRNA [32]. In addition, noise levels were associated with different functional groups of genes. For example, genes involved in protein synthesis are quiet whereas genes involved in production of proteins that respond to environmental changes are noisy. The authors suggest that imprecise protein expression regulation could be beneficial in that it could allow more rapid adaptations to fluctuating environments.

Studies have also examined the behavior of the yeast proteome in response to environmental perturbations with single-cell resolution, providing additional insight into protein localization and abundance changes that are masked by population averaging. For example, yeast proteome dynamics studies have uncovered that proteins can change localization or abundance in response to stress, and some proteins respond in a stress specific manner [13], [67], [68]. One study found that hydroxyurea, which slows DNA replication by limiting dNTP pools and biosynthesis enzymes, was associated with localization changes in proteins involved in mRNA decapping. Methyl methanesulfonate, which causes DNA damage that cannot be repaired, is associated with localization changes in genes associated with the cell cycle and DNA repair [67].

In addition, another study uncovered a bet-hedging mechanism which was employed by yeast cells grown in a low nitrogen environment [10]. A subset of yeast

GFP fusion clones exhibited bimodal expression profiles in response to low nitrogen stress, which may reflect a survival strategy. Specifically, cells expressing the PRE3 GFP fusion protein partitioned into high and low expressing subpopulations after a day of nitrogen starvation. Investigation of these subpopulations growth rate showed that the PRE3 high expressing subpopulation had a fitness advantage over 24h time scale but the low expressing subpopulation outcompeted the high expressing subpopulation over 4 days.

7.3 Application of Fluorescent Barcoding for the Study of Recombinant Prion Protein-Antibody Interactions

Fluorescent cell barcoding could be used to screen biomolecular libraries for protein-protein interactions and quantify binding affinities. In principle, members of a biomolecular library such as a cDNA library could be assigned to unique fluorescent barcodes by co-transformation or genetic fusion. Barcoded yeast also expressing surface displayed proteins of interest could be titrated with soluble antigens to screen for interaction partners. For example, yeast surface display and next-generation DNA sequencing were used to identify hundreds of peptides with affinity for mouse and human T-cell receptors [147]. Antigen titration could be conducted to estimate the dissociation constant and quantify the strength of the protein-protein interaction.

Towards this goal, we investigated the interaction between recombinant alpha-helical prion protein (PrP^{α}) and yeast surface displayed ICSM18 2.6.1 scFv, which was genetically fused to barcodes of varying lengths. Specifically, we subcloned the nucleotide sequence encoding for the ICSM18 2.6.1 variant into pCTCON2 and pBC2 plasmids, a 5-eptiope tag barcode library, and an 11-epitope tag barcode library. Expression of the scFv in pCTCON2 adds an N-terminal CMYC tag and a C-terminal

HA tag connected by flexible linkers, and a fusion to the AGA2 yeast mating protein. Expression of the scFv in pBC2 adds an N-terminal CMYC tag and linker, and a C-terminal fusion to a domain of the alpha-agglutinin yeast mating protein. scFv barcode library plasmids are composed of an N-terminal CMYC tag and linker, the scFv, the barcodes which are composed of 1 to 11 epitope tags connected by $(G_4S)_3$ linkers, and finally the alpha-agglutinin domain.

Recombinant PrP^{α} was produced in a 30L bioreactor fermentation of *E. coli* cells. Protein was isolated from inclusion bodies by ultracentrifugation and purified from host cell proteins using size-exclusion chromatography (SEC) (**Figure 7.1**). SEC fractions were tested for the presence of PrP by Western blotting, and purity was assessed to be more than 90% using Coomassie and silver stains. Then, protein was oxidized by exposure to atmospheric air for two weeks, followed by reverse-phase HPLC to isolate the oxidized protein fraction. After lyophilization, PrP was resolubilized in water and labeled with Alexa Fluor 647 using succinimidyl ester chemistry. Fluorescently conjugated protein was diluted into 4M Urea with Tris and oxidized glutathione at pH 8, allowing formation of an alpha-helical structure [54].

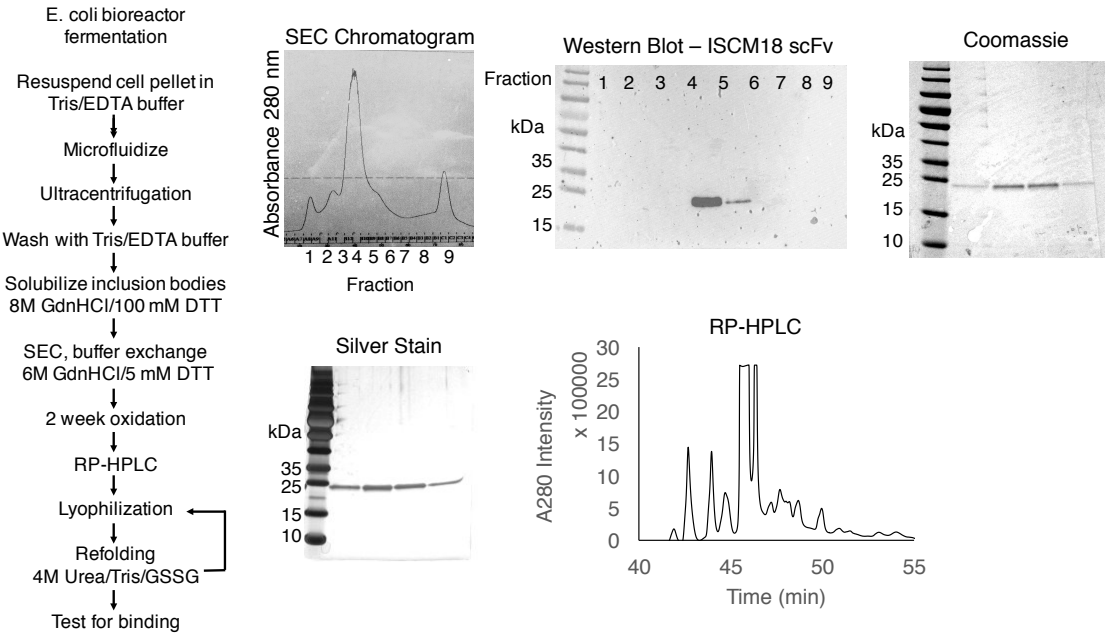


Figure 7.1: Production and purification of recombinant mouse prion protein. Recombinant prion protein was produced in a 4 hour biofermentation using *E. coli* cells. Inclusion bodies were isolated by ultracentrifugation and solubilized. SEC was used to purify prion protein from host cell proteins and fractions were tested for the presence of prion protein by Western blotting. Purity was assessed using Coomassie and silver staining. Prion protein was oxidized for two weeks by exposure to air and oxidized protein was purified from reduced by RP-HPLC and lyophilized. Finally, oxidized protein was resuspended in buffer to form an alpha-helical structure and conjugated to Alexa Fluor 647.

Fluorescently labeled PrP^{α} was titrated with ICSM18 2.6.1 scFv displayed on the yeast surface with or without barcodes in PBS 0.1% BSA pH 7.4 (**Figure 7.2**). Prior to immunolabeling, cells were induced for 24h in galactose media at 20°C or 30°C. Previous experiments by other lab members showed the binding affinity of ICSM18 2.6.1 for PrP^{α} to be ~2nM when expressed in the pCTCON2 vector. However, the titration did not result in a characteristic saturated binding curve for any

of the constructs tested, possibly due to aggregation of PrP. PrP could not be titrated above 100nM due to insolubility. Notably, all expressing cells appeared to interact with PrP, and no nonspecific interaction of PrP with the non-expressing cell population was observed.

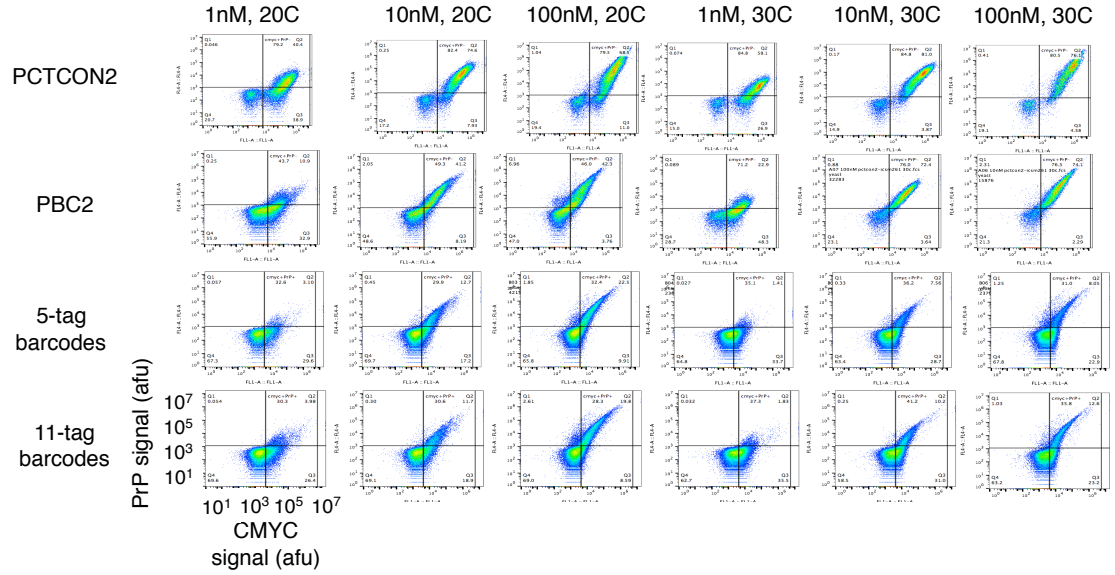


Figure 7.2: Titration of PrP with surface-displayed barcoded ICSM18 2.6.1. Yeast cells displaying ICSM18 2.6.1 as a fusion to AGA2 (pCTCON2) or alpha-aggutinin with (5-tag and 11-tag) or without (pBC2) barcodes were titrated with fluorescently labeled recombinant PrP^{α}. Saturation did not occur as expected in all cases, possibly due to PrP^{α} aggregation, and PrP^{α} did not bind nonspecifically to non-expressing yeast cells. Importantly, all cells expressing recombinant proteins on the surface bound PrP^{α}.

In order to evaluate if barcodes have an effect on the affinity of the PrP ICSM18 2.6.1 interaction, total PrP signal, which is proportional to the amount of PrP bound, was normalized by the median total CMYC fluorescence, which is related to the expression level of the surface displayed protein (**Figure 7.3**). We found that

ICSM18 2.6.1 barcode fusions that were induced at 20°C for 24h had the highest normalized PrP signal as compared to non-barcoded ICSM18 2.6.1 expressed at 20°C or 30°C or barcoded ICSM18 2.6.1 fusions induced at 30°C. This suggests that barcodes do not affect the apparent affinity of ICSM18 2.6.1 PrP interactions when scFV is expressed at 20°C.

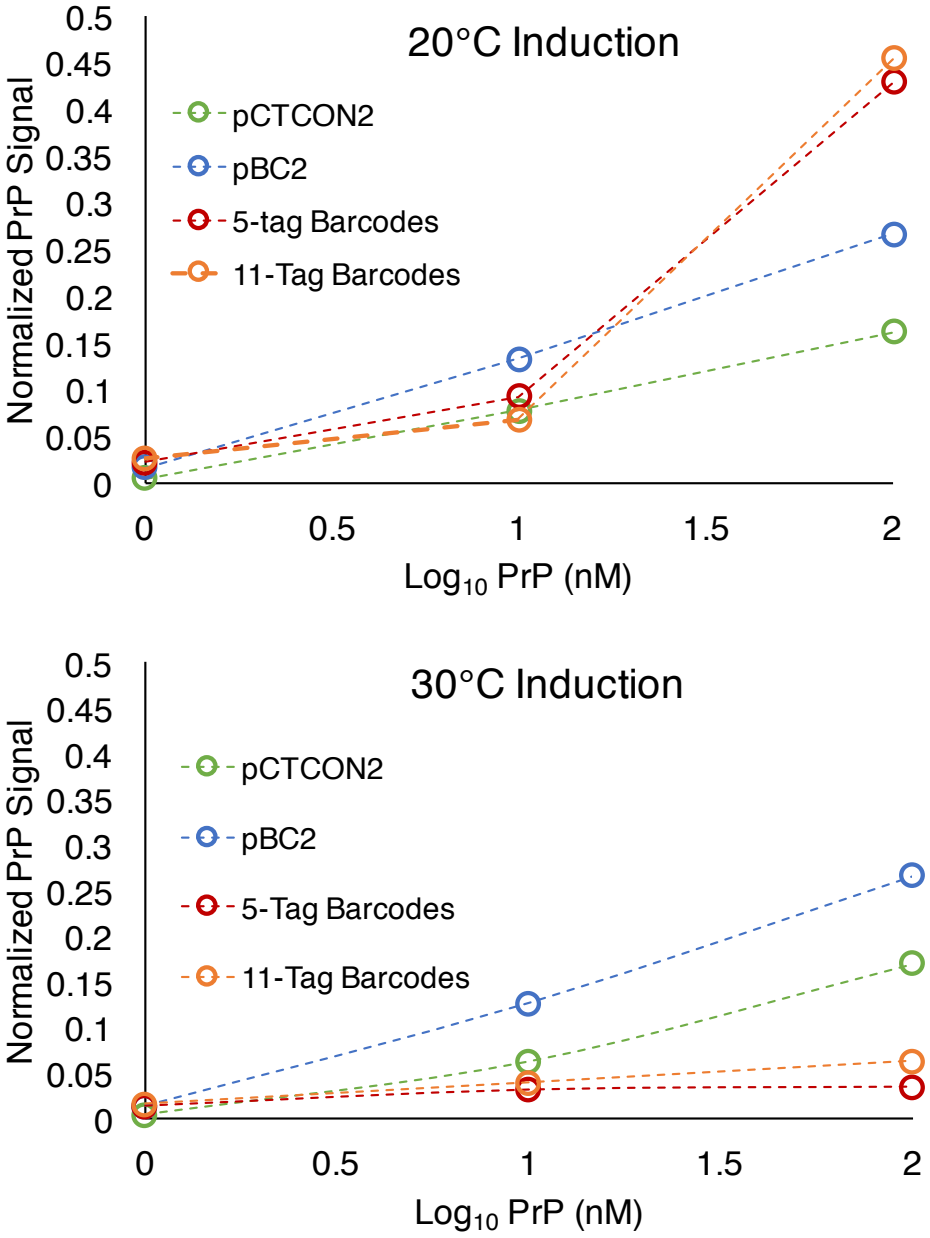


Figure 7.3: Effect of barcodes on the affinity of PrP ICSM18 2.6.1 interaction. The apparent affinity of yeast surface displayed ICSM18 2.6.1 PrP interaction was not affected when barcoded scFv was expressed at 20°C. This is suggested by the similar or higher median normalized PrP signal for barcoded ICSM18 2.6.1 as compared to ICSM18 2.6.1 without barcodes (pCTCON2 and pBC2) at multiple PrP concentrations.

Notably, ICSM18 2.6.1 exhibited lower CMYC signal when fused to alpha-agglutinin with or without barcodes as compared to the AGA2 fusion, suggesting lower expression levels (**Figure 7.4**). The ICSM18 2.6.1 alpha-agglutinin fusion without barcodes exhibited a 7-fold decrease in expression at 20°C and an 8.5-fold decrease in expression at 30°C compared to the AGA2 fusion. An additional 4-fold decrease in expression was observed when ICSM18 2.6.1 was fused to barcodes and expressed at 20°C. Expression of barcoded ICSM18 2.6.1 at 30°C exhibited a 11-fold decrease in expression with 5-tag barcodes and a 6-fold decrease in expression with 11-tag barcode fusions. Interestingly, the scFv had higher median expression at 30°C when barcodes were not present, but 20°C was favored for expression of barcode scFv fusions. In all cases, 30°C induction resulted in a higher percentage of cells expressing yeast surface displayed proteins.

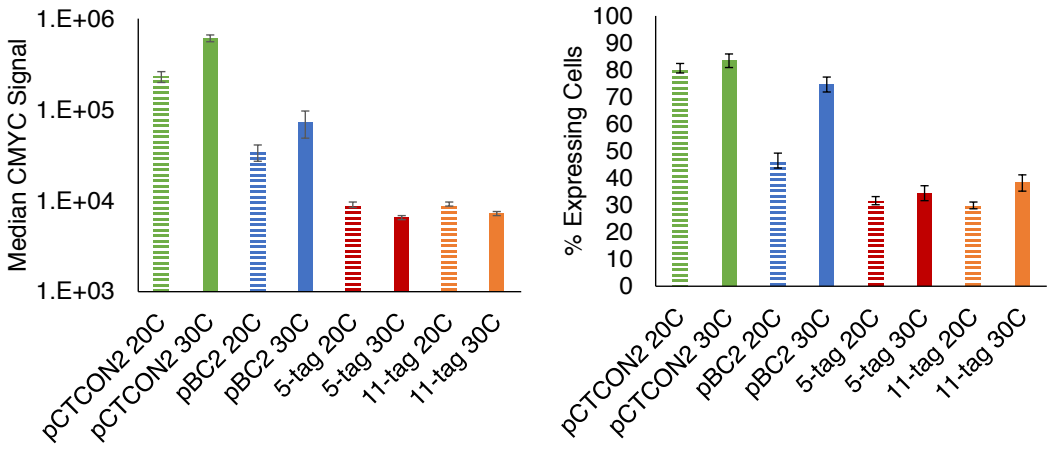


Figure 7.4: Expression of ICSM18 2.6.1 scFv is affected by barcode fusion. Yeast surface displayed ICSM18 2.6.1 barcode fusions exhibited 4 to 11-fold lower median expression as compared to ICSM18 2.6.1 alone. Expression levels were higher at 20°C for barcoded scFv and 30°C for scFv only. The percentage of expressing cells was higher in all cases when protein expression was induced at 30°C.

After optimizing the induction temperature of barcoded ICSM18 2.6.1 expression, we hypothesized that barcoded scFv expression could be improved by increasing the induction time for protein expression. Previous studies have shown that induction of heterologous yeast surface displayed proteins can be improved by varying induction time up from 20-48 hours [148]. Yeast surface displayed barcoded scFv expression levels were measured as a function of induction time at 20°C by immunolabeling with antibodies against the N-terminal CMYC tag and additionally with antibodies against the barcode epitope tags (**Figure 7.5**). Results show that barcoded ICSM18 2.6.1 expression improved approximately 3-fold after an additional 24h of induction. Further studies could be conducted to determine if longer induction times would be beneficial.

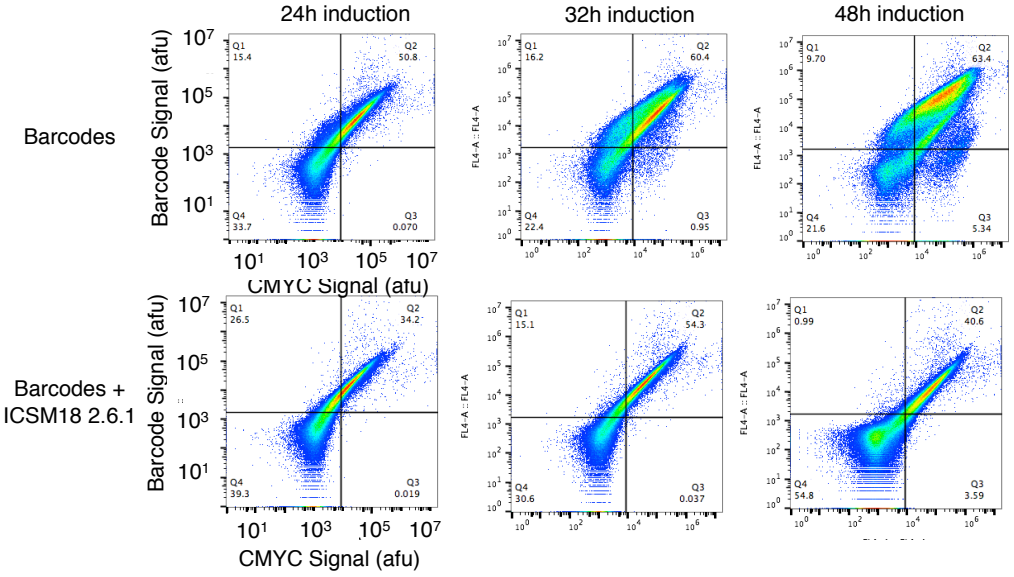


Figure 7.5: Effect of induction time on barcoded ICSM18 2.6.1 expression. Barcoded scFv expression improved approximately 3-fold after an addition 24h of induction at 20C, suggesting that longer induction times may be beneficial for higher expression.

7.4 Assignment of Unique Barcodes to Yeast GFP Fusion Clones and Investigation of Factors Affecting Barcode Expression

Systems biology is a research area in which a large amount of data, i.e. genomic, transcriptomic, metabolomics, or proteomic scale, is used to assess the behavior of cells, tissues, and organisms and create predictive models [2]. In addition, single-cell analysis provides greater resolution of information than methods that rely on population averaging, and it can reveal interesting phenomena that are masked by population averaging approaches. Previous single-cell studies of the yeast proteome have used time-consuming methods including robotics-based flow cytometry or automated fluorescence microscopy. Moreover, cells have only been investigated in a limited number of environments due to the large number of samples required. The fluorescent cell barcoding tool we have developed is a powerful multiplexing method for single-cell analysis that can greatly decrease the number of samples needed for a study by approximately 100 to 1000-fold. Therefore, fluorescent cell barcoding can enhance yeast proteomics studies by decreasing the amount of time and samples needed, and also allowing more replicates and perturbations to be studied.

We applied the fluorescent cell barcoding system to study the dynamic response of yeast proteins to environmental perturbations using the yeast GFP fusion collection. This collection is composed of over 4,000 yeast clones with each one expressing a different GFP fusion protein from the native open reading frame [55]. Clones were chosen based on those which were previously reported to have large coefficients in variation (CV) [41], which is defined as the standard deviation normalized by the mean, or those which were reported to have bimodal expression patterns in certain environmental conditions [10]. Additional yeast GFP clones that express proteins of unknown function, representing 17.6% of the yeast proteome, were

chosen for this study. We hypothesized that investigating changes in protein expression profiles in different environmental conditions could lead to insight into protein function.

Each of the fifteen barcode libraries, which are defined by a unique combination of 5 epitope tags, were transformed into 12 or 13 yeast GFP clones. Also, plasmids with known combinations of 5 epitope tags were transformed into clones. Single clones harboring a barcode plasmid an a unique endogenous GFP fusion protein were picked from selective agar plates. Then up to 15 barcoded yeast GFP clones, each containing a plasmid with a known combination of 5 epitope tags and an unknown combination of the other 6 epitope tags, were mixed together in a single tube for screening by immunolabeling and flow cytometry.

The results of the barcode screening experiments are shown in **Table 7.1**. 49 out of 81 (60.49%) of the barcoded yeast GFP clones were unique, and the probability of unique assignment rose to 72.41% when GFP clones assigned to specific 5-tag plasmids were excluded. In addition, the most commonly observed barcodes were those with 5-epitope tag combinations, namely AcV5/V5, V5, and T7/V5/E2. These results are not surprising because the 5 epitope tag plasmids were the most abundant barcodes present in the libraries (**Appendix H**). In general, the majority of barcodes in the libraries were observed once (56%) or two times (35.09%), indicating barcode libraries contain diverse combinations.

Table 7.1: Frequency of barcode observations during one-by-one assignment to yeast GFP clones.

Including transformants with 5-tag plasmids

# observations	# barcodes	% barcodes	# unique barcodes
1	28	31.65	28
2	28	32.91	14
3	12	22.78	4
4	8	5.06	2
5	5	7.59	1
		total # unique barcodes	**49**
		total % unique barcodes	**60.49**

Excluding transformants with 5-tag plasmids

# observations	# barcodes	% barcodes	# unique barcodes
1	28	56.14	28
2	24	35.09	12
3	6	8.77	2
		total # unique barcodes	**42**
		total % unique barcodes	**72.41**

Interestingly, more than 93% of attempted barcode plasmid transformations were successful, but only 54% of these successful transformants had barcode expression as observed by immunolabeling and flow cytometry (**Table 7.2**). This suggested that barcoded yeast clones were acquiring the selective marker during plasmid transformation but not a functional barcode. In addition, we calculated the probability of observing a yeast GPF clone with barcode expression given that the clone was transformed with either a 5-epitope tag plasmid or a mixture of barcode library plasmids (**Table 7.3**). Statistical comparison of the samples showed that the probabilities were significantly different ($p = 0.11$). Taken together, these data suggest that the mini-plasmids present in the barcode library were likely the underlying cause of the discrepancy between the number of successful transformants and the number of transformants expressing functional barcodes.

Table 7.2: Summary of yeast GFP barcode transformation and screening results.

Result	Count
Attempted GFP Clone Barcode Transformation	161
Successful GFP Clone Barcode Transformation	150
GFP Clones with Barcode Expression	84
GFP Clones with Unique Barcode Expresssion	49

Table 7.3: Barcode expression probability comparison between yeast GFP clones transformed with 5-tag plasmids and 11-tag libraries.

Sample	# transformants without barcode expression	# transformants with barcode expression	% transformants with barcode expression
SC 1	7	11	61.11
UK 1	4	12	75.00
		average	**67.65**
SC 2	4	9	69.23
SC 3	10	4	28.57
SC 4	8	6	42.86
SC 5	7	7	50.00
SC 6	6	5	45.45
UK 2	3	4	57.14
UK 3	5	5	50.00
UK 4	8	5	38.46
UK 5	3	8	72.73
UK 6	4	5	55.56
		average	**50.00**

Barcoded yeast GFP clone screening was conducted twice with the same transformants but different proportions of each clone. We observed that barcoded mixtures of yeast GFP clones had heterogeneous barcode expression, with an average expression of approximately 20% (**Figure 7.6**). Typically, cells expressing barcodes without the presence of mini-plasmids exhibited 50-70% expression, suggesting removal of cells containing mini-plasmids would restore expression levels. We

observed a recovery of barcode expression levels to 55.4% of cells after immunolabeling a mixture of barcoded yeast-GFP clones that had been screened previously for barcode expression.

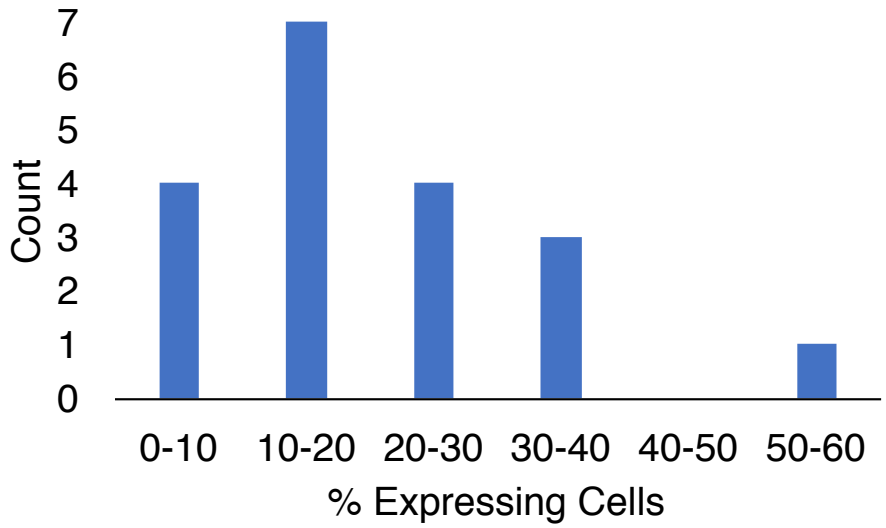


Figure 7.6: Barcoded yeast GFP mixtures exhibit a range of low expression percentages. Barcoded mixtures of yeast GFP clones exhibited expression heterogeneity, with an average expression level of ~20%. Typical barcode expression percentages ranged from 50-70%. The approximately two-fold lower expression percentage observed is consistent with the number of transformants expressing barcodes observed.

An alternative hypothesis for the low percentage of expression observed is that induction conditions are non-optimal for barcoded yeast GFP clones. Therefore, barcode expression was monitored over 64 hours of induction at two temperatures for four different mixtures of barcoded yeast GFP clones (**Figure 7.7**). Again, we observed that mixtures had a range of expression percentages. The expression level

and percentage of cells expressing barcodes was not improved for any of the mixtures at the times and temperatures tested. Additionally, the 24h and 30°C condition used for all other experiments was sufficient to induced barcode expression. These results suggest that low percentages of expressing cells are not caused by suboptimal barcode induction conditions.

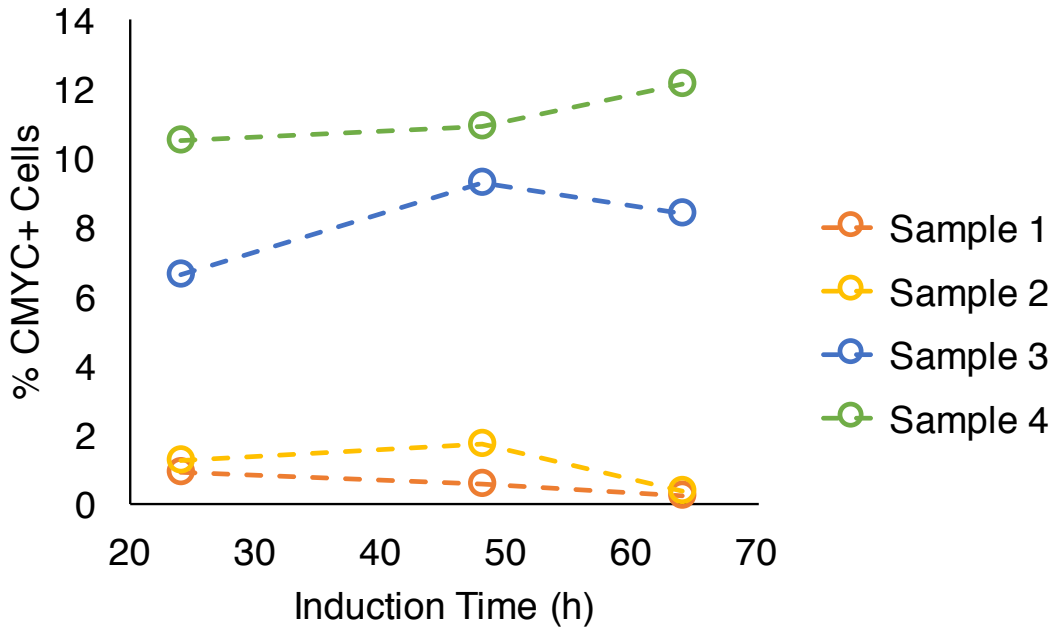


Figure 7.7: Barcoded yeast GFP clone mixture expression levels are unchanged in different induction conditions. Four mixtures containing different barcoded yeast GFP clones were tested for expression levels by immunolabeling with an antibody against the CMYC tag at two different temperatures and 64h of expression. Mixtures exhibited a range of expression percentages that did not vary with the induction conditions tested. Therefore, the low percentage of expressing cells is likely not caused by suboptimal induction conditions.

7.5 Improvement of GFP Signal to Background Ratio in Fixed Yeast Cells

Fixation of barcoded GFP cells is necessary for studies involving proteome dynamics because of the amount of processing time required for immunolabeling and flow cytometry measurements. Therefore, we investigated the effect of formaldehyde fixation on GFP fluorescence. A GFP fusion clone with highly fluorescent GFP expression and a negative control strain with the same genetic background as the GFP clones were grown overnight and fixed using either 1% OR 4% formaldehyde in PBS pH 7.4 for up to one hour (**Figure 7.8**). Fixation caused a decrease in GFP fluorescence, as also observed by others [149], with almost 2-fold losses after 10 minutes in the 1% condition. Over the time-course, 1% formaldehyde had less of an effect on GFP fluorescence than 4% formaldehyde as expected. Cells fixed at all conditions were checked for a lack of growth after 24h. GFP chromophore formation is accomplished by the folding, cyclization, and atmospheric oxidation of three amino acid side chains, T65, Y66, G67 [150]. These results suggest that formaldehyde may alter the structure of GFP, effectively lowering chromophore signal.

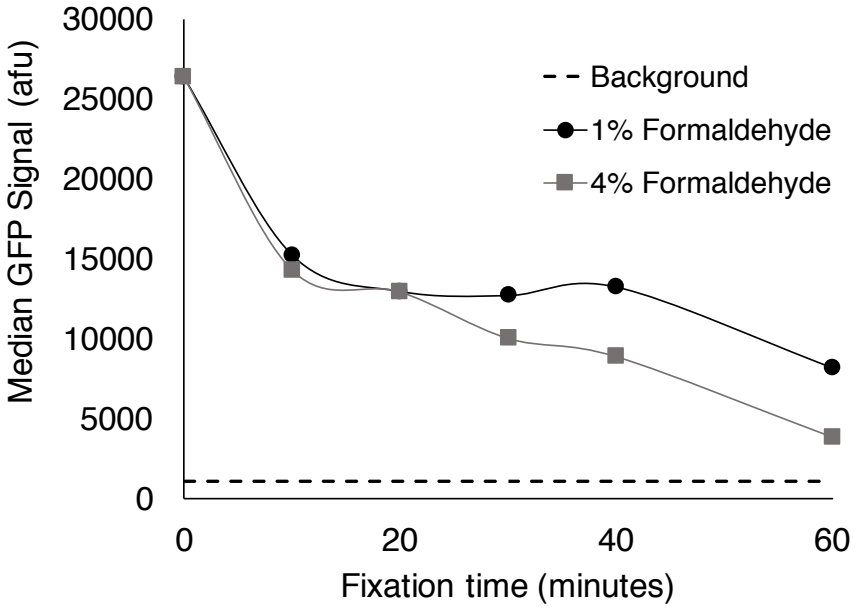


Figure 7.8: Effect of formaldehyde fixation on GFP fluorescence. GFP fluorescence of a highly expressed yeast GFP fusion clone was monitored over time during fixation with either 1% or 4% formaldehyde in PBS pH 7.4. At both conditions tested, formaldehyde lowered the GFP signal by almost 2-fold after only 10 minutes. As expected, 1% formaldehyde had less of a detrimental effect than 4% at longer times.

In addition, we wanted to improve the GFP detection sensitivity by increasing the signal to background ratio. It should be possible to achieve this by lowering cellular autofluorescence, which is ubiquitous in the 500-600nm range due to the presence of flavins, which are small molecule redox cofactors [151]. We hypothesized that permeabilization of the cell membrane would allow diffusion of flavins outside of the cell, effectively lowering autofluorescence. Autofluorescence and GFP fluorescence were measured by flow cytometry after fixed cells were exposed to different permeabilization reagents including detergents and alcohols (**Figure 7.9**).

The signal to background ratio was calculated by dividing the median fluorescence value for yeast cells expressing GFP by yeast cells not expressing GFP. Mild detergent permeabilization methods mildly improved detection sensitivity by lowering autofluorescence, and this effect was more pronounced for alcohols. In addition, when mild fixation conditions were used (1% formaldehyde for 10 minutes), alcohol permeabilization restored the GFP signal to background ratio to levels seen in unfixed cells. Effectively, alcohol lowered cellular autofluorescence enough to overcome the decrease in GFP signal caused by formaldehyde.

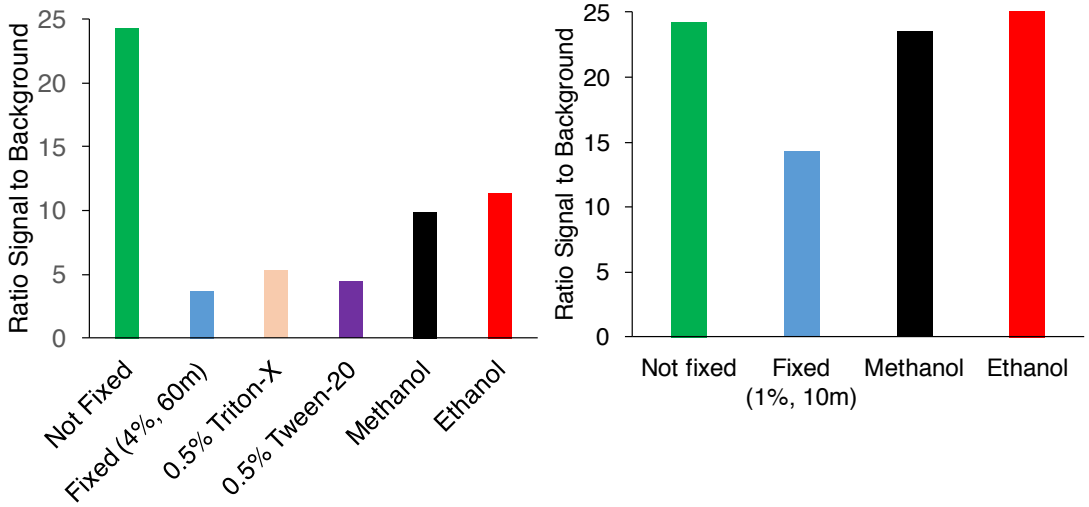


Figure 7.9: Permeabilization improves the signal to background ratio for fixed cells expressing GFP by lowering autofluorescence. Only mild improvements in GFP signal to background were achieved when detergents were used (left panel). Alcohols were more successful in improving the detection sensitivity by lowering autofluorescence. Alcohol fixation improved the signal to background ratio for GFP to unfixed cell levels by lowering autofluorescence, effectively overcoming the decrease in GFP signal due to fixation.

7.6 Dynamic Behavior of *S. cerevisiae* Protein Expression in Response to Environmental Perturbations

We examined the single-cell protein expression profiles of barcoded yeast GFP fusion clones in response to environmental perturbations as a first demonstration of barcode technology applied to studies of single-cell yeast proteomics. The barcoded GFP fusion clone mixtures contained different number of barcoded clones, GFP fusion proteins, and assigned barcodes. Two mixtures, containing four and seven barcoded yeast GFP clones, were studied under four different environmental conditions and at two different time points after perturbation. One mixture contained barcoded yeast GFP fusion clones that express proteins known to be upregulated in response to certain environmental stresses (**Table 7.4**). The other sample contained barcoded yeast GFP fusion clones whose fusion proteins are known to be upregulated in response to certain stresses, as well as some clones with unknown function (**Table 7.5**).

Table 7.4: Mixture of four barcoded yeast GFP clones with known stress responses.

GFP Clone ORF	GFP Clone Name	Barcode	Related Stress Response	Description
YDR513W	GRX2	10011000000	oxidative stress	Cytoplasmic glutaredoxin
YBR126C	TPS1	00001000000	heat shock	Synthase subunit of trehalose-6-P synthase/phosphatase complex
YJR104C	SOD1	11000000000	oxidative stress	Cytosolic copper-zinc superoxide dismutase
YER103W	SSA4	00100000000	heat shock	Heat shock protein that is highly induced upon stress

Table 7.5: Mixture of seven barcoded yeast GFP clones with either known stress responses or unknown function.

GFP Clone ORF	GFP Clone Name	Barcode	Description
YIL127C	RRT14	10011000000	Putative protein of unknown function
YDR099W*	BMH2	11001000000	14-3-3 protein, minor isoform
YNR014W		01001010000	Putative protein of unknown function
YGL108C		01011000000	Protein of unknown function, predicted to be palmitoylated
YGR012W	MCY1	10000000010	Putative cysteine synthase
YER062C*	GPP2	11000000000	DL-glycerol-3-phosphate phosphatase involved in glycerol biosynthesis
YCR016W		01100100000	Putative protein of unknown function

*control

Specifically, barcoded yeast GFP clones were mixed together in approximately equal proportions in a single tube and exposed to either no stress, heat stress at 37°C, oxidative stress induced by 1mM H_2O_2, or alcohol stress with 5% ethanol for up to one hour. Cells were washed with PBS and fixed with formaldehyde to preserve cellular structure at the time points indicated. Before barcode immunolabeling, cells were permeabilization with methanol to lower autofluorescence. Analysis of single-cell protein expression profiles for barcoded yeast GFP clones with known stress responses revealed interesting and in some cases unexpected behavior (**Figure 7.10**).

GRX2 is a cytoplasmic glutaredoxin involved thiol oxidoreduction and has been reported to have increased gene expression in response to oxidative, osmotic, and heat stress as well as stationary phase growth [152]. We found that GRX2 had a bimodal expression profile, with 2-6% of cells expressing GRX2 at a 50-fold higher level on average. cells expressing high levels of GRX2 were statistically significantly larger than those in the low expressing population ($p = < 10^{-7}$). Also, SSA4 is a heat shock protein that is highly induced under heat stress conditions, and has also been reported to be induced during oxidative and ethanol stress [153]–[155]. We found that SSA4 on average increased 2-fold after 30 minutes of heat stress at 37°C and expression rose to 3-fold after 60 minutes. In addition, the expression distribution of SSA4 was significantly smaller than during non-stressed conditions. CV decreased 1.7-fold on average from 101 in non-stress conditions to 60 at 30 minutes and 56 at 60 minutes. SSA4 expression was not induced during ethanol or oxidative stress, and upregulation of the cytosolic copper/zinc superoxide dismutase SOD1 was also not observed. One possible explanation for no SOD1 change is that the perturbation

conditions were too mild to induce significant increases in protein expression, or that protein expression was not upregulated during the examined time points [156].

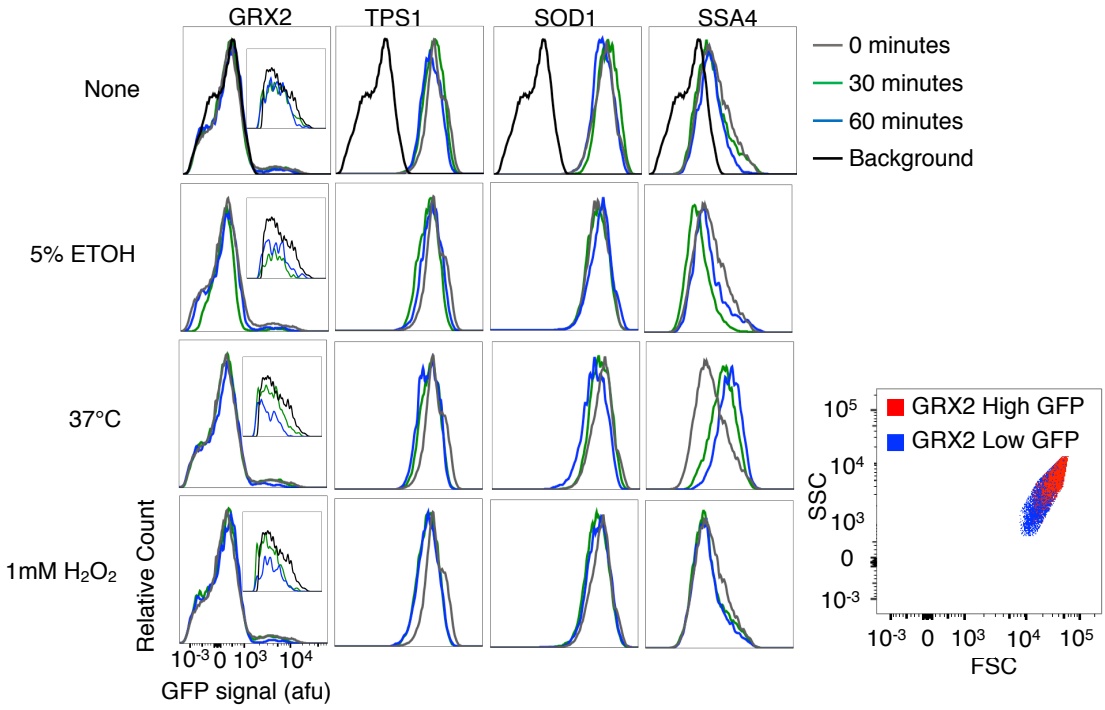


Figure 7.10: Single-cell dynamic protein expression response to environmental perturbations. GRX2, a thiol oxidoreductase, exhibited a bimodal expression profile in all conditions tested. Cells in the high expressing GRX2 population were larger than those in the low expressing population. SSA4, a heat shock protein, was upregulated in heat stress and a contraction in protein expression deviation was observed. SSA4 also had decreased expression after 30 minutes of heat shock. Interestingly, SOD1 is known to increase expression during oxidative stress, but remained unchanged in this case.

In addition, a second mixture of seven barcoded yeast GFP clones containing proteins of unknown function and control proteins with reported stress responses was studied under the same stress conditions (**Figure 7.11**). The protein expression levels of the five proteins with unknown functions remained unchanged under all conditions tested, with the exception of YGL108C. After 30 minutes of heat stress, cells expressing a GFP YGL108C fusion exhibited a bimodal expression pattern with 11% of cells expressing the protein at ~100-fold higher levels. YGL108C expression returned to basal levels after an additional 30 minutes, which is reasonable because protein half lives in yeast can be as low as < 4 minutes with an average half-life of ~40 minutes [157]. In addition, it is not entirely surprising that most of the unknown function protein expression profiles remained unchanged, as only three stress conditions were tested and these proteins are likely only upregulated in specific circumstances.

The proteins of known function, BMH2 and GPP2, remained unchanged in the conditions tested. BMH2 is a regulatory protein important in RAS/MAPK signaling and vesicle transport, and has been reported to be upregulated during DNA replication stress [67]. GPP2 is a phosphatase involved in glycerol biosynthesis and has been shown to be upregulated in response to the oxidant paraquat, which produces superoxide anions [158]. It is likely we did not observe induction of GPP2 with hydrogen peroxide because it may be specifically induced in response to agents that generate superoxide such as menadione and paraquat.

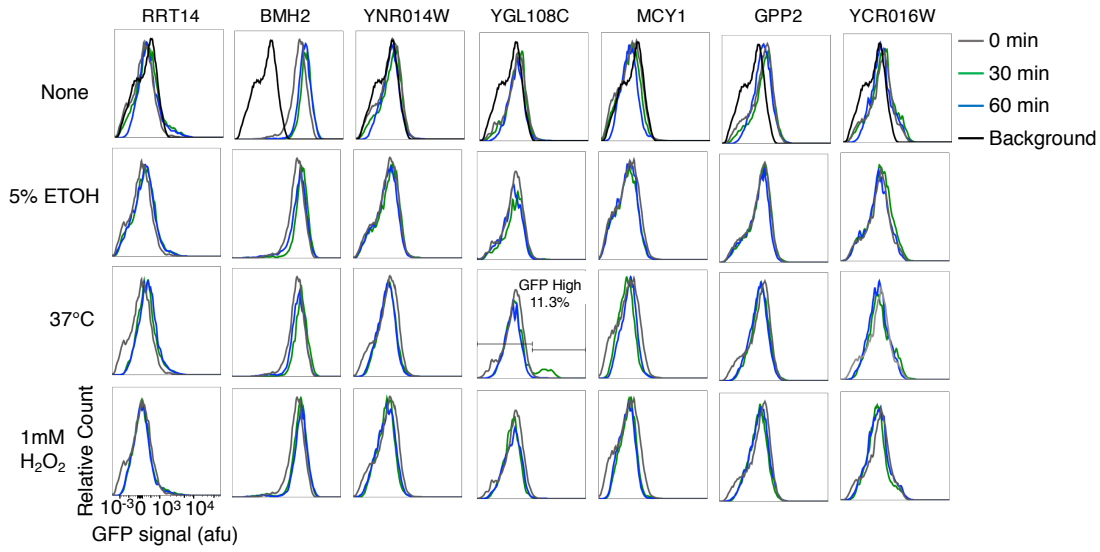
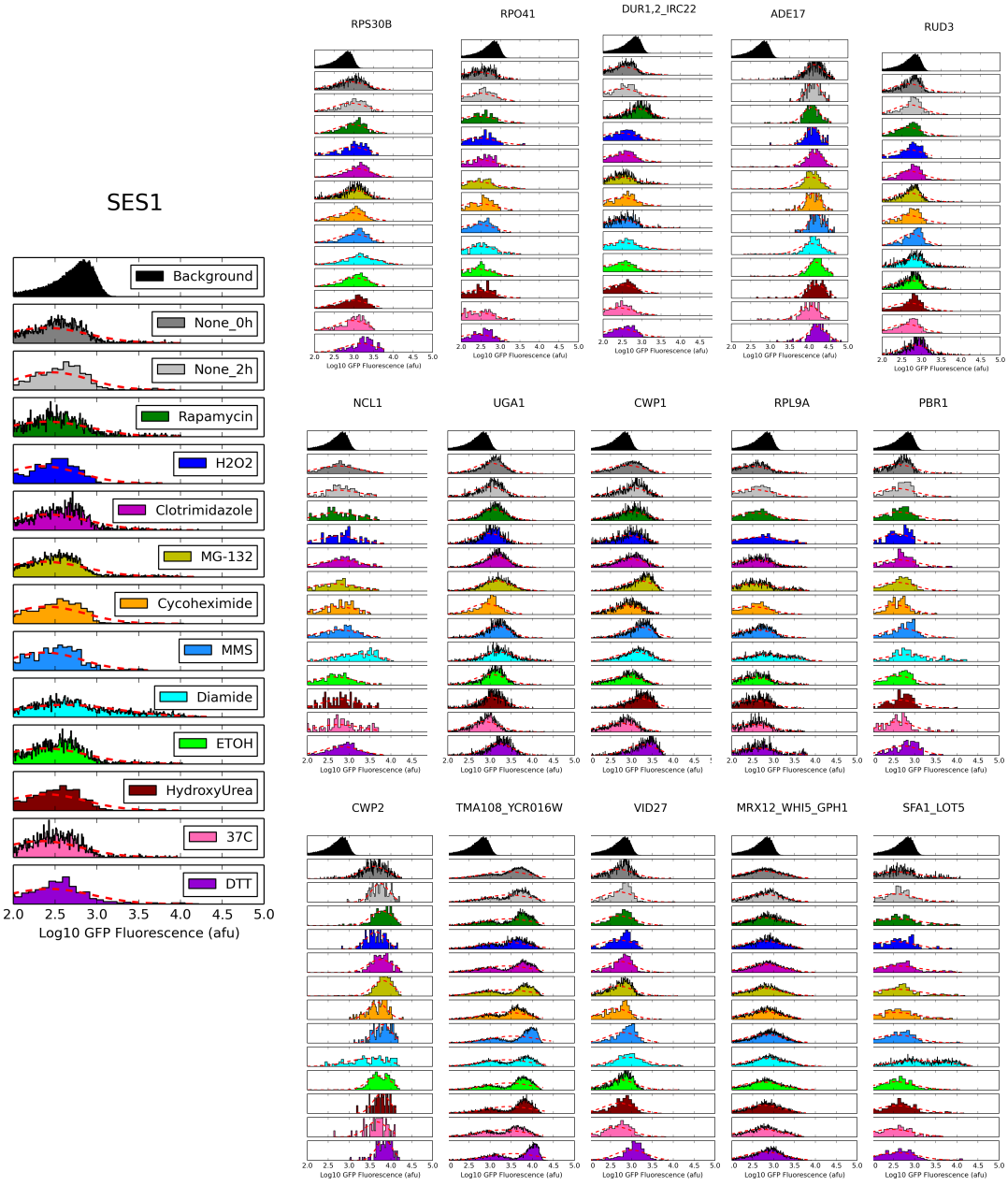


Figure 7.11: Dynamic, single-cell response of endogenous yeast GFP fusion proteins with unknown function to environmental perturbations. The five proteins with unknown function (RRT14, YNR014W, YGL108C, MCY1, and YCR016W) remained unchanged in response to the stress conditions tested, with the exception of the 30 minute heat shock condition for YGL108C. During this condition, YGL108C had a bimodal expression profile with 11.3% of yeast cells exhibiting high expression levels.

Next, we applied fluorescent barcoding to simultaneously study the response of a larger set of endogenous yeast GFP fusion proteins (**Table I.1**). 46 barcoded yeast GFP clones were mixed together in a single tube and exposed to eleven different stress conditions for two hours (**Table I.2**). After immunolabeling and flow cytometry, barcodes were used to identify the yeast GFP clones, and their single-cell expression profiles before and after perturbations were elucidated (**Figure 7.12**). To elucidate GFP distributions and quantify abundance changes, manual analysis was used to de-convolve barcoded populations and a Python script was written to analyze GFP expression profiles (**Appendix J**). Due to an immunolabeling error, specifically the

addition of insufficient HA antibody, only 32 of the 46 barcoded clone were distinguished.

This result demonstrates that fluorescent barcoding can enable massively parallel analysis of cellular libraries, permitting the study of cellular behavior in a wider array of different conditions and potentially gaining a more multifaceted view of the cell. Specifically in this case, the use of 32 fluorescent barcodes decreased the number of samples required from 416 to 13. The GFP expression distributions elucidated by single-cell analysis show that protein abundance varies from cell-to-cell, showing that average measurements are insufficient to capture protein abundance. Moreover, these results show that variation in protein expression levels changes on a protein to protein basis, and could indicate pathway specific regulatory mechanisms or differences in promoter noise [32], [41]. One interesting finding was that the distribution of the ribosomal subunit proteins, namely RPS30B, RPL1B, RPL9A, RPL20, and RPL21, had increased variation in response to diamide. This observation could suggest that translation is upregulated in a subset of cells in response to diamide stress, as diamide is known to damage proteins by thiol oxidation, which causes the formation of disulfide bonds.


SES1
Background
None_0h
None_2h
Rapamycin
H2O2
Clotrimidazole
MG-132
Cycoheximide
MMS
Diamide
ETOH
HydroxyUrea
37C
DTT
2.0 2.5 3.0 3.5 4.0 4.5 5.0
Log10 GFP Fluorescence (afu)
RPS30B
RPO41
DUR1,2_IRC22
ADE17
RUD3
NCL1
UGA1
CWP1
RPL9A
PBR1
CWP2
TMA108_YCR016W
VID27
MRX12_WHI5_GPH1
SFA1_LOT5

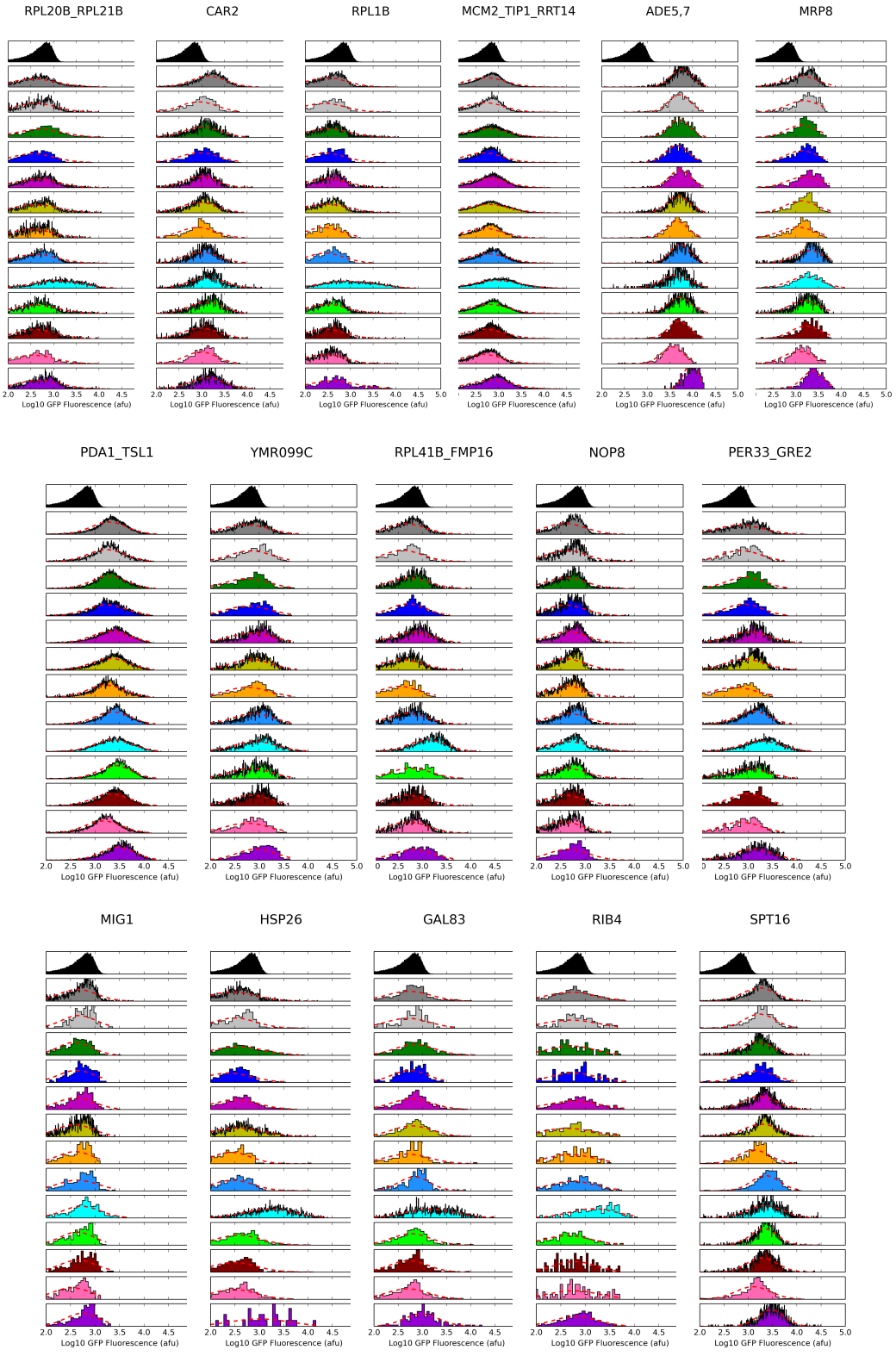
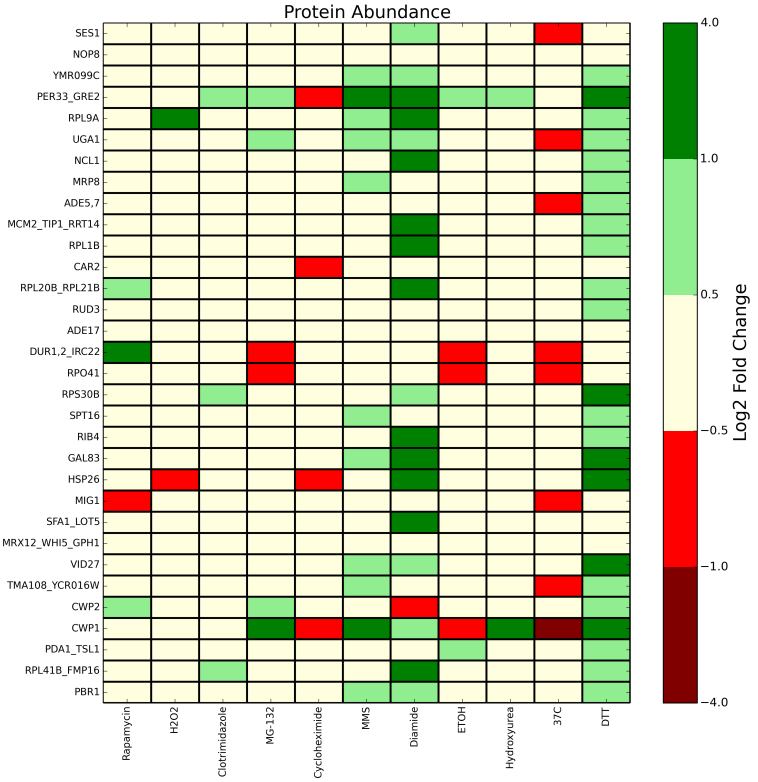

RPL20B_RPL21B
CAR2
RPL1B
MCM2_TIP1_RRT14
ADE5,7
MRP8
PDA1_TSL1
YMR099C
RPL41B_FMP16
NOP8
PER33_GRE2
MIG1
HSP26
GAL83
RIB4
SPT16
Log10 GFP Fluorescence (afu)

Figure 7.12: Barcodes enable multiplexed analysis of 32 single-cell protein expression distributions in 12 environmental conditions. Barcoded yeast GFP clones were pooled in a single sample and exposed to stress for two hours. After immunolabeling and barcode deconvolution, their GFP protein expression profiles were elucidated. Note that the clones that could not be deconvolved are indicated by an underscore. The lognormal fits are indicated by the dashed red line.

In addition, the protein abundance and variation changes in response to different environmental conditions were quantified (**Figure 7.13**). In general, we found that protein abundance increased in response to diamide and DTT, and decreased in response to heat stress. One interpretation for this observation is that cells increase protein expression in response to protein misfolding caused by oxidative or reductive damage. Decreases in the variation of protein expression was more commonly observed than abundance changes, with dimaide causing the most changes significantly affecting 20 out of the 32 proteins studied. In response to stress conditions, most proteins decreased their expression variation, suggesting a bet-hedging response in which cells express a wider array of protein expression levels in order to more rapidly combat adverse environments.


Protein Abundance
SES1
NOP8
YMR099C
PER33_GRE2
RPL9A
UGA1
NCL1
MRP8
ADE5,7
MCM2_TIP1_RRT14
RPL1B
CAR2
RPL20B_RPL21B
RUD3
ADE17
DUR1,2_IRC22
RPO41
RPS30B
SPT16
RIB4
GAL83
HSP26
MIG1
SFA1_LOT5
MRX12_WHI5_GPH1
VID27
TMA108_YCR016W
CWP2
CWP1
PDA1_TSL1
RPL41B_FMP16
PBR1
Rapamycin
H2O2
Clotrimidazole
MG-132
Cycloheximide
MMS
Diamide
ETOH
Hydroxyurea
37C
DTT
4.0
1.0
0.5
−0.5
−1.0
−4.0
Log2 Fold Change

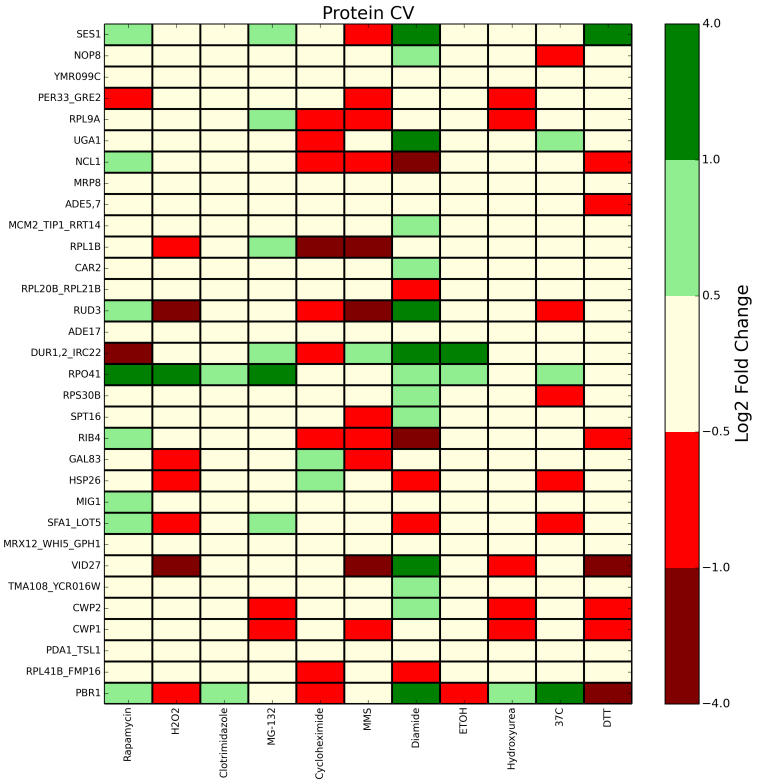

Protein CV
SES1
NOP8
YMR099C
PER33_GRE2
RPL9A
UGA1
NCL1
MRP8
ADE5,7
MCM2_TIP1_RRT14
RPL1B
CAR2
RPL20B_RPL21B
RUD3
ADE17
DUR1,2_IRC22
RPO41
RPS30B
SPT16
RIB4
GAL83
HSP26
MIG1
SFA1_LOT5
MRX12_WHI5_GPH1
VID27
TMA108_YCR016W
CWP2
CWP1
PDA1_TSL1
RPL41B_FMP16
PBR1
Rapamycin
H2O2
Clotrimidazole
MG-132
Cycloheximide
MMS
Diamide
ETOH
Hydroxyurea
37C
DTT
4.0
1.0
0.5
−0.5
−1.0
−4.0
Log2 Fold Change

Figure 7.13: Protein abundance and variation changes in response to environmental stress. Fold change was calculated as the ratio of the protein abundance or CV after stress versus before stress. Proteins that were upregulated or had wider variation after stress are shown in green and those that were downregulated or had narrower distributions are shown in red. Fold changes greater than 1.4 times the average abundance or CV of the non-stress condition replicates were considered significant.

7.7 Discussion

In this chapter, we present proof of concept applications to illustrate the use of fluorescent barcoding for multiplexed analysis of biomolecular and cellular libraries. Towards the demonstration of fluorescent cell barcoding for massively-parallel analysis of biomolecular libraries including protein-protein interactions studies, we determined the binding affinity of a yeast surface displayed, barcoded α-prion scFv, ICSM18 2.6.1 for recombinant prion protein. Specifically, we discuss the production, characterization, and use of recombinant prion protein in antigen titrations to determine the affinity of barcoded and non-barcoded ICSM18 2.6.1. Our preliminary results indicate that barcodes do not have a detrimental effect on the apparent interaction affinity, as shown by equivalent normalized binding signal for barcoded and non-barcoded ICSM18 2.6.1. We also demonstrate that barcoded ICSM18 2.6.1 has full-length expression on the yeast surface.

However, the expression level of barcoded ICSM18 2.6.1 was significantly lower with barcodes, which is surprising since this mutant was engineered to have higher stability on the yeast surface. Low expression of barcode ICSM18 2.6.1 fusions could be caused by a variety of factors, including the presence of mini-plasmids (for 11-tag barcodes), improper folding of the scFv, or interaction of the scFv with the cell surface when it is attached to barcodes which can be substantially lengthier than the

typically used one or two epitope tag fusions. In addition, the yeast surface displayed barcoded ICSM18 2.6.1 could have poor expression due to the need to fold the ICSM18 2.6.1 into the correct structure before secretion. Along the same lines, it is likely that long barcodes do not exhibit a decrease in expression level and have higher expression levels at 30°C because they do not require folding. In support of the folding hypothesis, the apparent binding affinity for barcoded ICSM18 2.6.1 was restored to levels similar to non-barcoded scFv when protein expression was induced at 20°C, suggesting that slower specific growth rates may improve barcoded ICSM18 2.6.1 folding via slower kinetics [159].

In order to further improve the expression of barcoded ICSM18 and possibly other scFvs, dual promoter expression vectors could be used. In this scheme, the scFv mutant would be expressed in one transcript and the fluorescent barcode on a separate one. Elimination of a direct fusion may improve ICSM18 2.6.1 expression levels to those exhibited for pCTCON2 or pBC2. In addition, using a more stable yeast surface displayed protein could lead to improvements in protein-barcode expression. Additional improvements in barcoded ICSM18 2.6.1 expression could be obtained by fusion to AGA2 instead of alpha-agglutinin, as ICSM18 2.6.1 alone fused to AGA2 had ~7-fold lower expression than when fused to alpha-agglutinin.

Immunolabeling of barcoded ICSM18 2.6.1 fusions over 2.5 days of induction at 20°C showed expression increased over time. However, the ICSM18 2.6.1 barcode profile at all induction times examined was characteristic of that observed for barcodes alone at 24h, whereas barcodes exhibited multiple subpopulations after 2.5 days. This could indicate that either expression of barcoded ICSM18 2.6.1 is too low to observe multiple populations, or that only certain, likely smaller barcodes, are expressed when

fused to ICSM18 2.6.1. Further experiments in which specific epitope tags are labeled could shed light on this observation. Although we have shown preliminarily that barcodes do not affect the ICSM18 prion interaction affinity, the effect of specific barcodes on binding affinity should be elucidated by further experiments.

In the second part of this work, we discuss the assignment of fluorescent barcodes to yeast GFP clones for the multiplexed study of single-cell protein dynamics in response to environmental perturbations. To assign barcodes to yeast GFP clones, two approaches could be used, namely a one-by-one approach or a library approach. In the one-by-one approach, plasmids harboring unique barcodes could be assigned using a multi-well plate transformation, or a library of plasmids could be used followed by clonal screening. We chose to use a one-by-one approach due to its ease of execution, and successfully assigned 49 unique barcodes to yeast GFP clones. However, the approach is most suitable for smaller studies on the order of 100 clones, and is limited by the highest abundant barcodes in the libraries. Currently, one-by-one assignment would likely only result in on the order of 100 unique barcoded yeast GFP clones, as there are approximately 300 barcodes present in the top 1-10% of barcode libraries (**Appendix H**).

Normalization of barcode libraries by FACS could greatly improve the number of unique barcodes assigned to yeast GFP clones using a one-by-one approach. Specifically, FACS could be used to normalize the distribution of barcodes in the libraries, as demonstrated in Chapter 4, and to lessen the amount of background 5-epitope tag barcodes in the libraries, as illustrated by the increase in unique barcode assignment from 60-72% when barcode plasmids were not specifically assigned to yeast GFP clones. Also, a high rate of successful barcode transformation was observed

(>93%) but there was a low rate of successful yeast GFP clones expressing barcodes (50%), likely due to the presence of mini-plasmids in the barcode library DNA. In order to improve the likelihood of yeast GFP clones expressing functional barcodes, sorting of barcoded yeast GFP clones could be used to remove cells harboring mini-plasmids. An alternative approach would be to separate full-length barcode plasmids from mini-plasmids by gel electrophoresis prior to yeast transformation.

Although we have used a one-by-one approach to assign barcodes to yeast GFP clones, random assignment using a library approach could also be used. Specifically, yeast GFP clones could be pooled and transformed with a mixture of barcode plasmids followed by limiting dilution to achieve an expected value of one to one assignment. This approach is advantageous because it is limited by a particular range of barcode abundance, likely one order of magnitude, instead of by the most abundant barcodes. A library barcode assignment approach would likely lead to a higher number of uniquely barcoded yeast GFP clones, as the majority of barcodes in our libraries range in abundance from 0.1-1%. Also, with a library approach FACS could be used to easily remove transformants harboring mini-plasmids. One disadvantage of this approach is that barcode GFP clone assignment is not known a priori, and would require additional experiments such as gene amplification and DNA sequencing in order to ascertain the identity of the GFP fusion.

After assigning unique fluorescent barcodes to yeast GFP clones, we studied the dynamic single-cell response of GFP fusion proteins to a variety of environmental perturbation. Our experiments revealed interesting dynamic behavior, including bimodal expression profiles for GRX2 under all conditions and for YGL108C during heat stress. The bimodal expression profile of GRX2 was ubiquitous under all

conditions tested, and the high expressing fraction was composed of larger cells on average. Taken together, these results could suggest that the high expressing fraction is in stationary phase, as GRX2 has been shown to be upregulated during stationary phase, and stationary phase cells are larger than log phase cells [81]. This hypothesis can be further elucidated by cell cycle analysis and growth rate studies. Also, the bimodal expression of YGL108C GFP fusions could indicate that this protein in somehow involved in a rapid and transient stress response regime [160]. In addition, we found that yeast cells increase expression of the SSA4 heat shock protein 2-3 fold after yeast were incubated for one hour at 37°C. Furthermore, the variation in protein expression within the population decreased by approximately two-fold. This behavior, which is revealed by single-cell analysis, could be indicative of a bet-hedging mechanism in which cells express a wider range of protein expression levels under non-stress conditions as to more quickly adapt to sudden environmental changes.

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The main objective of this work was to develop a fluorescent barcoding system that can be used for massively parallel single-cell analysis of bimolecular and cellular libraries. The work described here discusses the approaches used to engineer, to the best of our knowledge, the largest fluorescent barcoding system to date consisting of over 1000 unique, genetically-encoded barcodes. In addition, we have illustrated the power of this system by applying the method to analyze protein-protein interaction affinities and high-throughput, single-cell protein expression dynamics in response to environmental perturbations. The barcoding system developed throughout this work can be further expanded upon for greater multiplexing capability, and can be used in a variety of research applications that would greatly benefit from massively-parallel single-cell analysis, including systems biology, computational modeling of protein-protein interactions and rational antibody design, protein-protein interaction screening, and antibody engineering and epitope/paratope mapping.

In the first part of this work, we discuss the development of single-color barcodes composed of up to 16 repeating epitope tags connected by flexible linkers, using a new, generally-applicable method for exponential expansion of tandem DNA repeats. This method could be used for subcloning any sequence of interest, such as CRISPR guide RNA or poly-glutamine sequences. Epitope tag repeat proteins up to 34 kDa exhibited full-length expression that did not decrease with increasing repeat

length, and can improve immunodetection by more than 100-fold for yeast-surface displayed protein fusions and more than 40-fold for endogenous protein fusions. Epitope tag repeats can be used to enhance immunodetection for low abundance proteins or for proteins in which no antibody is available, as well as improve immunopurification due to avidity effects. We demonstrate that long epitope fusions can enable flow cytometry detection of endogenous low abundance proteins in yeast, thus enabling the study of more than 1,600 low abundance proteins using single-cell analysis. In addition, we discovered that plasmids containing long epitope tag repeats are unstable in *E. coli* and developed a computational approach to analyze next-generation sequencing data in support of this observation.

In the second part of this work, engineering of a multi-color fluorescent barcoding system is presented. We demonstrate that up to four distinct fluorescence intensities can be achieved by fluorescence normalization and expression of epitope tag repeats of different lengths, and that the number of distinct intensities is heavily influenced by fluorophore brightness. To our knowledge, this is the first example of a genetically-encoded fluorescent barcoding system with the capability to generate barcodes with four fluorescence intensities using a single fluorophore. 190 out of 216 possible barcodes were created by combining 6 different types of epitope tags with varying repeat lengths, resulting in greater than 2-fold more barcodes than any other published system. In addition, we identified critical epitope tag interdependencies causing differences in the fluorescence of barcoded subpopulations, and show that there is a bias for smaller barcodes with fewer repeats to have higher library abundances, suggesting transformation efficiency is affected by repeat length.

Furthermore, constitutive barcode expression was found to impact cell growth rate in a size-dependent manner, and an inducible promoter was used to minimize this bias, highlighting the importance of inducible promoters for production of toxic proteins or in for cellular libraries with rare members that could be lost due to growth differences. To expand the barcode library from hundreds to thousands of members, 18 barcodes made of 5 additional epitope tag combinations were constructed by homologous recombination or PCR. We found that, contrary to previous reports [87], nucleotide sequences with less than 80% homology were capable of recombination. This finding may suggest that homologous recombination can occur regardless of exact homology or that the distribution of non-homologous residues can affect the probability of recombination.

Fourteen barcode libraries consisting of combinations of up to 11 epitope tags were constructed, resulting in the creation of more than 1100 barcodes by computational estimation. Our barcoding system has 10-fold more barcodes than any other published to date and 20-fold more than any other genetically-encoded barcoding system. In addition, mini-plasmids composed of plasmid backbone without barcode regions spontaneously formed as a result of unstable plasmids caused by epitope tag repeats. This highlights the importance of minimizing repeating sequence length if possible and using recombinase deficient cell lines for cloning. Also, software was developed to rapidly analyze high-dimensional flow cytometry data, resulting in a decrease in time required for analysis from 4 hours to approximately 25 minutes per data set. The software we have developed could be used to cluster, identify, and quantify other multi-color flow cytometry data sets. We found that the

software had a low error rate (2.5% false identification, 4.5% mislabeled barcode positive intensities) and modest accuracy (16-44% of barcodes missed).

In the final part of this work, we applied the fluorescent barcoding system to illustrate its use for multiplexed, single-cell analysis of biomolecular and cellular libraries. Towards the application of fluorescent barcodes for studying protein-protein interactions, we barcoded the α-prion scFv ICSM18 2.6.1 and produced and characterized recombinant prion protein. We found that barcode expression did not impact the apparent binding affinity of ICSM18 2.6.1 for prion protein, but did observe a decrease in expression for barcoded scFv compared to non-barcoded scFv. These results suggest that barcodes could be used for multiplexed analysis of biomolecular libraries composed of protein mutants, which are useful for studies involving protein-protein interactions, epitope/paratope mapping, or rational antibody design applications.

In a second application, we applied the fluorescent barcoding system to examine the dynamic, single-cell response of yeast proteins to environmental changes using yeast clones that express a particular protein GFP fusion from the endogenous promoter. Barcode plasmid transformation into yeast GFP fusion clones was very successful (94% of the 150 clones attempted), but only 50% of clones had barcode expression and only 30% contained unique barcode combinations. In addition, we studied the dynamic response of yeast GFP fusion clones to different environmental stresses including heat, oxidative stress, and ethanol, and found interesting changes in protein abundance, distribution, or bimodality for three proteins (SSA4, GRX2, and YGL108C). This illustrates the utility of fluorescent barcoding for systems biology applications, as it permits more replicates, time points, and conditions to be studied

more easily by decreasing the fold-number of samples required by the number of barcodes used. Furthermore, this application highlights that single-cell analysis can provide additional insight into proteomics studies that are masked with population averaging methods.

8.2 Future Work

In the first part of this work, we discuss the development of single-color barcodes composed of different lengths of repeating epitope tags connected by flexible linkers. Long repeats were used to greatly improve immunodetection by flow cytometry, enabling detection of low abundance endogenous fusion proteins in yeast. Future studies could focus on a more expansive study of epitope fusion proteins in single-cells, and their behavior could be studied in different environmental conditions or cell types. For example, many low abundance proteins are not well studied, have unknown functions, or have important roles in cellular function such as transcription factors.

In addition, deep sequencing and gel electrophoresis were used to investigate the instability of plasmids containing long epitope tag repeats. This work could be expanded upon by investigating methods to decrease repeat instability, including using different *E. coli* strains, varying linker sequences or shortening linkers to decrease the number of repeats or repeat length, and developing an *E. coli* free cloning method such as homologous recombination. In addition, the deep sequencing data could be further analyzed to potentially elucidate patterns of plasmid deletion and could suggest mechanisms underlying this phenomenon or approaches to decease recombination frequency.

In the second part of this work, we developed a fluorescent barcoding system for multiplexed, single-cell analysis, resulting in the creation of over 1,100 unique fluorescent barcodes. A number of approaches could be used to further expand the number of unique barcodes, as up to 2048 unique binary combinations are possible with 11-epitope tags and up to 3420 combinations are possible with the current libraries. The number of multiple intensity barcodes could potentially be expanded by increasing the number of 'bright' fluorophores using a different flow cytometry setup with capability for quantum dot detection, or brightness could be enhanced using species specific antibodies and secondary detection. Additionally, it is likely that more than 200 barcodes could be recovered by successful cloning of the four remaining barcode libraries. Furthermore, new barcode combinations could be created if additional libraries were constructed with a focus on missing or rare epitope tag combinations from the 6-epitope tag library, and 18-fold more barcodes could be constructed if the missing 5-epitope tag combinations are created.

Future experiments focusing on the construction of additional barcodes may benefit from using a different subcloning method such as Golden Gate cloning or homologous recombination. It is possible to clone up to nine inserts at once using Golden Gate cloning, which could be advantageous from a time perspective, although efficiency has been shown to decrease for repetitive sequences [92]. Spacer sequences could be incorporated to create plasmids with homogenous sizes, possibly lessening the observed transformation bias but potentially increasing the likelihood of plasmid instability. Homologous recombination could be advantageous, as repeat plasmids are unstable in *E. coli*. This method is also rapid and could be conducted in one step

assuming barcodes were constructed with unique linker sequences that would permit specific recombination.

In the third part of this work, we developed software for the rapid identification and quantification of barcodes from flow cytometry data using the DBSCAN clustering algorithm and kernel density estimation. Although the software greatly decreased the time required to analyze multicolor flow cytometry data and had a low error rate, many barcodes were missed by the algorithm. In order to improve the accuracy of the software, larger data sets could be used, which would allow the use of more stringent filtering criteria and in turn increased density and regularity of barcoded subpopulations. In addition, manual preprocessing of data could be eliminated if kernel density estimation were used to filter out noisy data points in all channels instead of only those with multiple fluorescence intensities. Furthermore, future experiments should focus on identification of additional filtering criteria to eliminate false positive barcodes by analysis of more control data sets. Finally, the software could be improved if fluorophore intensity identification could be accomplished in the absence of populations with lesser intensities. In order to accomplish this, data sets would have to be analyzed for identification criteria such as minimum and maximum fluorescence values for a certain intensity, taking into account epitope tag interdependency effects.

In the final part of this work, we illustrate the utility of the fluorescent barcoding system for multiplexed analysis of biomolecular and cellular libraries. In a first application, we studied the interaction of recombinant mouse prion protein with the barcoded α-prion scFv ICSM18 2.6.1. Further applications involving barcoded biomolecular libraries should explore the impact of other scFvs or proteins of interest

on surface-displayed protein expression. Protein expression levels may also be improved using a dual promoter vector or AGA2 fusion, and the effect of specific barcodes on the apparent binding affinity of ICSM18 2.6.1 prion interactions should be elucidated. In addition, DNA barcodes could be used to uniquely assign members of bimolecular libraries to fluorescent barcodes. DNA barcodes can be created using type IIS restriction enzyme sites and incorporated randomly into fluorescent barcodes. After deep sequencing to pair DNA and fluorescent barcodes, members of a biomolecular library can be uniquely DNA barcoded by PCR or gene synthesis and assigned to a fluorescent barcode via subcloning with a unique pair of DNA barcode sticky ends.

Furthermore, in a second fluorescent barcoding application, we simultaneously examined the dynamics of endogenous yeast GFP fusion proteins in single-cells in response to environmental perturbations. Follow up experiments should be conducted to explore the reproducibility and noise associated with GFP fusion protein dynamics, and results should be verified in the absence of barcodes. Further experiments can be conducted to explore the interesting protein expression dynamics found in this work, including additional time points, conditions, and fluorescent readouts such as cell cycle analysis, as well as other types of experiments such as fitness measurements. In addition, studies of barcoded yeast GFP fusion clones could be expanded to hundreds to thousands of proteins after improvements to the barcode libraries have been made.

A number of improvements can be made to the barcode libraries in order to improve the likelihood of unique barcode assignment to yeast GFP clones. We found that mini-plasmids and highly abundant barcodes affected the number of transformants with expressed barcodes and unique barcodes respectively. To decrease the number of

mini-plasmids using a one-by-one transformation approach, barcode DNA could be purified by gel extraction before transformation to remove mini-plasmids. In addition, barcode libraries could be sorted using FACS to lessen the number of highly abundant barcodes and enrich rare barcodes. If barcodes are transformed into a library of yeast GFP clones instead of with a one-by-one approach, FACS could be used to eliminate transformants with no barcode expression and simultaneously lessen the number of transformants expressing highly abundant barcodes.

In this work, a one-by-one barcode assignment approach was used in which individual yeast GFP clones were transformed with a mixture of barcodes and individual transformants were screened for unique barcode assignment. This approach is advantageous because assignment is known prior to experimentation, but it is somewhat low throughput. Alternatively, more high-throughput library methods could be used for barcode assignment. In a library approach, barcodes could be assigned to libraries of interest randomly in a one pot transformation. Limiting dilution could be used to isolate a specific set of barcodes with an expected value of one. For example, barcodes with 1% abundance would appear once on average for a limiting dilution of 100 clones. For biomolecular libraries in which pairings are genetically linked on a plasmid, deep sequencing can be used to determine barcode assignment. For cellular libraries such as the yeast GFP library, clones that exhibit interesting behavior could be isolated with FACS and the fusion protein could be amplified by PCR for sequencing to determine barcode clone parings.

Fluorescent barcoding is a powerful tool that can be used for massively-parallel analysis of biomolecular and cellular libraries of interest, and information derived from these studies can be applicable to many research areas including systems

biology, biochemistry, molecular dynamics, and protein engineering. In addition to the applications described in this work, it may be possible to apply fluorescent barcoding to many applications including high-throughput screens for protein-protein interactions, assessment of gene function using knockout libraries, and quantitative analysis of the effect of point mutations on binding affinity for protein engineering [161], [162], paratope/epitope mapping [163], and protein docking model applications [164]. Fluorescent barcoding may be particularly useful when it is desirable to assess interactions with multiple targets, such as bispecific or broadly neutralizing antibodies, and to evaluate for the absence of binding which can be used to improve antibody specificity. Fluorescent barcoding may also be extended to other cell types such as mammalian and *E. coli* cells.

REFERENCES

[1] T. Ideker *et al.*, "Integrated Genomic and Proteomic Analyses of a Systematically Perturbed Metabolic Network," *Science (80-.).*, vol. 292, no. May, pp. 929–935, 2001.

[2] H. Chuang, M. Hofree, and T. Ideker, "A Decade of Systems Biology," *Annu. Rev. Cell Dev. Biol.*, vol. 26, pp. 721–44, 2010.

[3] L. Hood, J. R. Heath, M. E. Phelps, and B. Lin, "Systems Biology and New Technologies Enable Predictive and Preventative Medicine," *Science (80-.).*, vol. 306, no. October, pp. 640–644, 2004.

[4] J. M. Raser and E. K. O'Shea, "Control of Stochasticity in Eukaryotic Gene Expression," *Science (80-.).*, vol. 304, no. 5678, pp. 1811–1814, 2004.

[5] M. B. Elowitz, A. J. Levine, and E. D. Siggia, "Stochastic Gene Expression in a Single Cell," *Science (80-.).*, vol. 297, no. August, pp. 1183–1186, 2002.

[6] L. S. Weinberger, J. C. Burnett, J. E. Toettcher, A. P. Arkin, and D. V Schaffer, "Stochastic Gene Expression in a Lentiviral Positive-Feedback Loop : HIV-1 Tat Fluctuations Drive Phenotypic Diversity," *Cell*, vol. 122, pp. 169–182, 2005.

[7] N. Q. Balaban, J. Merrin, R. Chait, L. Kowalik, and S. Leibler, "Bacterial Persistence as a Phenotypic Switch," *Science (80-.).*, vol. 305, no. September, pp. 1622–1626, 2004.

[8] E. Kussell, R. Kishony, N. Q. Balaban, and S. Leibler, "Bacterial Persistence : A Model of Survival in Changing Environments," *Genetics*, vol. 1814, no. April, pp. 1807–1814, 2005.

[9] S. F. Levy, N. Ziv, and M. L. Siegal, "Bet Hedging in Yeast by Heterogeneous , Age-Correlated Expression of a Stress Protectant," *PLoS Biol.*, vol. 10, no. 5, 2012.

[10] M. Breker, M. Gymrek, and M. Schuldiner, "A novel single-cell screening platform reveals proteome plasticity during yeast stress responses," *J. Cell Biol.*, vol. 200, no. 6, pp. 839–850, 2013.

[11] M. Acar, J. T. Mettetal, and A. Van Oudenaarden, “Stochastic switching as a survival strategy in fluctuating environments,” *Nat. Genet.*, vol. 40, no. 4, pp. 471–475, 2008.

[12] X. Liu *et al.*, “Analysis of Cell Fate from Single-Cell Gene Expression Profiles in C . elegans,” *Cell*, vol. 139, pp. 623–633, 2009.

[13] N. Dénervaud, J. Becker, R. Delgado-gonzalo, P. Damay, A. S. Rajkumar, and M. Unser, “A chemostat array enables the spatio-temporal analysis of the yeast proteome,” *PNAS*, vol. 110, no. 39, 2013.

[14] J. R. Heath, A. Ribas, and P. S. Mischel, “Single-cell analysis tools for drug discovery and development,” *Nat. Rev. Drug Discov.*, vol. 15, no. 3, pp. 204–216, 2015.

[15] D. Wang and S. Bodovitz, “Single cell analysis : the new frontier in ‘ omics ,’” *Trends Biotechnol.*, vol. 28, pp. 281–290, 2010.

[16] G. P. Irish, J.M., Kotecha, N. and Nolan, “Mapping normal and cancer cell signalling networks : towards single-cell proteomics,” *Nat. Rev. Cancer*, vol. 6, no. February, pp. 146–155, 2006.

[17] B. Bodenmiller *et al.*, “Multiplexed mass cytometry profiling of cellular states perturbed by small-molecule regulators,” *Nat. Biotechnol.*, vol. 30, no. 9, pp. 857–866, 2012.

[18] P. O. Krutzik and G. P. Nolan, “Fluorescent cell barcoding in flow cytometry allows high-throughput drug screening and signaling profiling,” *Nat. Methods*, vol. 3, no. 5, 2006.

[19] L. C. Mattheakis *et al.*, “Optical coding of mammalian cells using semiconductor quantum dots,” *Anal. Biochem.*, vol. 327, pp. 200–208, 2004.

[20] C. Lin *et al.*, “Submicrometre geometrically encoded fluorescent barcodes self-assembled from DNA,” *Nat. Chem.*, vol. 4, no. September, 2012.

[21] B. Akkaya *et al.*, “A Simple, Versatile Antibody-Based Barcoding Method for Flow Cytometry,” *J. Immunol.*, vol. 197, pp. 2027–38, 2017.

[22] C. Kuo *et al.*, “Optically Encoded Semiconducting Polymer Dots with Single-Wavelength Excitation for Barcoding and Tracking of Single Cells,” *Anal. Chem.*, 2017.

[23] R. L. Mccarthy, D. H. Mak, J. K. Burks, and M. C. Barton, “Rapid

monoisotopic cisplatin based barcoding for multiplexed mass cytometry," *Sci. Rep.*, pp. 1–6, 2017.

[24] T. Maetzig *et al.*, "A Lentiviral Fluorescent Genetic Barcoding System for Flow Cytometry-Based Multiplex Tracking," *Mol. Ther.*, vol. 25, no. 3, pp. 606–620, 2017.

[25] R. Chen *et al.*, "A Barcoding Strategy Enabling Higher-Throughput Library Screening by Microscopy," *ACS Synth. Biol.*, 2015.

[26] M. Mohme *et al.*, "Optical Barcoding for Single-Clone Tracking to Study Tumor Heterogeneity," *Mol. Ther.*, vol. 25, no. 2, 2017.

[27] G. M. Mali, P., Aach, J. Lee, J., Levner, D., Nip, L. and Church, "Barcoding cells using cell-surface programmable DNA-binding domains," *Nat. Methods*, vol. 10, no. 5, pp. 403–6, 2013.

[28] C. A. Smurthwaite *et al.*, "Fluorescent Genetic Barcoding in Mammalian Cells for Enhanced Multiplexing Capabilities in Flow Cytometry," *Cytom. Part A*, vol. 1, no. 2, pp. 105–113, 2014.

[29] J. M. Raser and E. K. and O'Shea, "Noise in Gene Expression : Origins, Consequences, and Control," *Science (80-.).*, vol. 309, pp. 2010–2014, 2005.

[30] Y. Taniguchi *et al.*, "Quantifying E. coli Proteome and Transcriptome with Single-Molecule Sensitivity in Single Cells," *Science (80-.).*, no. 533, 2011.

[31] W. J. Blake, M. Kærn, C. R. Cantor, and J. J. Collins, "Noise in eukaryotic gene expression," *Nature*, vol. 422, no. 2003, pp. 633–7, 2003.

[32] A. Bar-even *et al.*, "Noise in protein expression scales with natural protein abundance," *Nat. Genet.*, vol. 38, no. 6, pp. 636–643, 2006.

[33] L. Cai, N. Friedman, and X. S. Xie, "Stochastic protein expression in individual cells at the single molecule level," *Nature*, vol. 440, no. March, 2006.

[34] E. M. Ozbudak, M. Thattai, I. Kurtser, A. D. Grossman, and A. Van Oudenaarden, "Regulation of noise in the expression of a single gene," *Nat. Genet.*, vol. 31, no. April, pp. 69–73, 2002.

[35] N. Friedman, L. Cai, and X. S. Xie, "Linking Stochastic Dynamics to Population Distribution : An Analytical Framework of Gene Expression," *Phys. Rev. Lett.*, 2006.

[36] L. Keren *et al.*, "Noise in gene expression is coupled to growth rate," *Genome*

Res., pp. 1893–1902, 2015.

[37] X. Chen and J. Zhang, "The Genomic Landscape of Position Effects on Protein Expression Level and Noise in Yeast," *Cell Syst.*, vol. 2, no. 5, pp. 347–354, 2016.

[38] A. Becskei, B. B. Kaufmann, and A. Van Oudenaarden, "Contributions of low molecule number and chromosomal positioning to stochastic gene expression," *Nat. Genet.*, vol. 37, no. 9, pp. 937–944, 2005.

[39] E. M. Ozbudak, M. Thattai, H. N. Lim, B. I. Shralman, and A. van Oudenaarden, "Multistability in the lactose utilization network of Escherichia coli," *Nature*, vol. 4680, no. 1982, pp. 4677–4680, 2004.

[40] M. Acar, A. Becskei, and A. Van Oudenaarden, "Enhancement of cellular memory by reducing stochastic transitions," *Nature*, vol. 435, no. May, pp. 1–5, 2005.

[41] J. R. S. Newman *et al.*, "Single-cell proteomic analysis of S . cerevisiae reveals the architecture of biological noise," *Nature*, vol. 441, no. June, 2006.

[42] H. B. Fraser, A. E. Hirsh, G. Giaever, J. Kumm, and M. B. Eisen, "Noise Minimization in Eukaryotic Gene Expression," *PLoS Biol.*, vol. 2, no. 6, pp. 834–838, 2004.

[43] P. J. Choi, L. Cai, K. Frieda, and X. S. Xie, "A Stochastic Single-Molecule Event Triggers Phenotype Switching of a Bacterial Cell," *Science (80-.).*, vol. 4142, no. October, pp. 442–447, 2008.

[44] N. Vardi, S. Levy, M. Assaf, and M. Carmi, "Report Budding Yeast Escape Commitment to the Phosphate Starvation Program Using Gene Expression Noise," *Curr. Biol.*, vol. 23, no. 20, pp. 2051–2057, 2013.

[45] S. V Sharma *et al.*, "A Chromatin-Mediated Reversible Drug-Tolerant State in Cancer Cell Subpopulations," *Cell*, vol. 141, no. 1, pp. 69–80, 2010.

[46] A. A. Cohen *et al.*, "Dynamic Proteomics of Individual Cancer Cells in Response to a Drug," *Science (80-.).*, vol. 322, no. December, pp. 1511–1517, 2008.

[47] C. B. Brachmann, A. Davies, G. J. Cost, and E. Caputo, "Designer Deletion Strains derived from Saccharomyces cerevisiae S288C : a Useful set of Strains and Plasmids for PCR-mediated Gene Disruption and Other Applications," *Yeast*, vol. 132, pp. 115–132, 1998.

[48] R. D. Gietz and R. H. Schiestl, “Microtiter plate transformation using the LiAc / SS carrier DNA / PEG method,” *Nat. Protoc.*, vol. 2, no. 1, pp. 5–9, 2008.

[49] D. W. et. al. Colby, “Engineering Antibody Affinity by Yeast Surface Display,” *Methods Enzymol.*, vol. 388, no. 2000, pp. 348–358, 2004.

[50] G. T. Hermanson, *Bioconjugate Techniques*, Second Edi. Rockford, Illinois, USA, 2008.

[51] G. Chao, W. L. Lau, B. J. Hackel, S. L. Sazinsky, S. M. Lippow, and K. D. Wittrup, “Isolating and engineering human antibodies using yeast surface display,” *Nat. Protoc.*, vol. 1, no. 2, pp. 755–769, 2007.

[52] W. Huang, X. and Miller, “A Time-Efficient , Linear-Space Similarity Algorithm,” *Adv. Appl. Math.*, vol. 357, pp. 337–357, 1991.

[53] I. Mehlhorn *et al.*, “High-Level Expression and Characterization of a Purified 142-Residue Polypeptide of the Prion Protein,” *Biochemistry*, vol. 35, pp. 5528–5537, 1996.

[54] B. Lu, P. J. Beck, and J. Chang, “Oxidative folding of murine prion mPrP (23-231),” *Eur. J. Biochem.*, vol. 268, pp. 3767–3773, 2001.

[55] W. Huh *et al.*, “Global analysis of protein localization in budding yeast,” *Nature*, vol. 425, pp. 686–91, 2003.

[56] R. Y. Giepmans, B.N. Adams, S.R., Ellisman, M.H. and Tsien, “The Fluorescent Toolbox for Assessing Protein Location and Function,” *Science (80-.).*, vol. 312, no. April, 2006.

[57] N. C. Shaner, P. A. Steinbach, and R. Y. Tsien, “A guide to choosing fluorescent proteins,” *Nat. Methods*, vol. 2, no. 12, pp. 905–909, 2005.

[58] S. C. Bendall, G. P. Nolan, M. Roederer, and P. K. Chattopadhyay, “A deep profiler ’ s guide to cytometry,” *Trends Immunol.*, vol. 33, no. 7, pp. 323–332, 2012.

[59] B. Brizzard, “Epitope tagging,” *Biotechnqiues*, vol. 44, no. 5, pp. 693–695, 2008.

[60] J. W. Jarvik and C. A. Telmer, “EPITOPE TAGGING,” *Annu. Rev. Genet.*, vol. 32, pp. 601–18, 1998.

[61] B. J. Hackel, A. Kapila, and K. D. Wittrup, “Picomolar Affinity Fibronectin Domains Engineered Utilizing Loop Length Diversity , Recursive Mutagenesis

, and Loop Shuffling," pp. 1238–1252, 2008.

[62] E. V Shusta, M. C. Kieke, E. Parke, D. M. Kranz, and K. D. Wittrup, "Yeast Polypeptide Fusion Surface Display Levels Predict Thermal Stability and Soluble Secretion Efficiency," *JMB*, pp. 949–956, 1999.

[63] K. and Ueda, "Yeast cell-surface display — applications of molecular display," *Appl Microbiol Biotechnol*, vol. 64, pp. 28–40, 2004.

[64] K. D. Boder, E.T. and Wittrup, "Yeast surface display for screening combinatorial polypeptide libraries," *Nat. Biotechnol.*, vol. 15, pp. 553–7, 1997.

[65] S. P. Perfetto, P. K. Chattopadhyay, and M. Roederer, "Seventeen-colour flow cytometry : unravelling the immune system," *Nat. Rev. Immunol.*, vol. 4, no. August, pp. 1160–1163, 2004.

[66] V. P. Zharov, V. V Tuchin, and A. Ta, "In Vivo Flow Cytometry : A Horizon of Opportunities," *Cytom. Part A*, pp. 737–745, 2011.

[67] J. M. Tkach *et al.*, "Dissecting DNA damage response pathways by analysing protein localization and abundance changes during DNA replication stress," *Nat. Cell Biol.*, vol. 14, no. 9, pp. 966–976, 2012.

[68] Y. T. Chong, J. L. Y. Koh, C. Boone, and B. J. Andrews, "Yeast Proteome Dynamics from Single Cell Imaging and Automated Analysis Resource Yeast Proteome Dynamics from Single Cell Imaging and Automated Analysis," *Cell*, vol. 161, no. 6, pp. 1413–1424, 2015.

[69] R. Pepperkok and J. Ellenberg, "Microscopy for systems biology," *Nat. Mol. Cell Biol.*, vol. 7, no. September, pp. 690–696, 2006.

[70] R. et al. Wooster, "Instability of short tandem repeats (microsatellites) in human cancers," *Nat. Genet.*, vol. 6, pp. 152–6, 1994.

[71] F. A. Al-allaf, O. E. Tolmachov, L. P. Zambetti, V. Tchetchelnitski, and H. Mehmet, "Remarkable stability of an instability-prone lentiviral vector plasmid in Escherichia coli Stbl3," *3 Biotech*, vol. 3, pp. 61–70, 2013.

[72] J. P. Jakupciak and R. D. Wells, "Genetic Instabilities in (CTG CAG) Repeats Occur by Recombination," *JBC*, vol. 274, no. 33, pp. 23468–23479, 1999.

[73] D. R. Leach, "Long DNA palindromes, cruciform structures, genetic instability, and secondary structure repair," *BioEssays*, vol. 16, no. 27, pp. 893–900, 1994.

[74] M. Bzymek and S. T. Lovett, "Instability of repetitive DNA sequences : The

role of replication in multiple mechanisms," *PNAS*, vol. 98, no. 15, 2001.

[75] X. Bi, L. F. Liu, and U. Wood, "A Replicational Model for DNA Recombination between Direct Repeats," *JMB*, vol. 256, pp. 849–858, 1996.

[76] J. S. Ghaemmaghami, S, Huh, W., Bower, K., Howson, R.W., Belle, A., Dephoure, N., OShea, E.K., Weissman, "Global analysis of protein expression in yeast," *Nature*, vol. 108, no. 1997, pp. 737–741, 2003.

[77] P. Tompa, J. Prilusky, I. Silman, and J. L. Sussman, "Structural disorder serves as a weak signal for intracellular protein degradation," *Proteins Struct. Funct. Genet.*, vol. 71, no. 2, pp. 903–909, 2008.

[78] N. Baumgarth and M. Roederer, "A practical approach to multicolor flow cytometry for immunophenotyping," *J. Immunol. Methods*, vol. 243, pp. 77–97, 2000.

[79] M. Roederer, "Spectral Compensation for Flow Cytometry : Visualization Artifacts , Limitations , and Caveats," *Cytometry*, vol. 45, pp. 194–205, 2001.

[80] P. K. Chattopadhyay *et al.*, "Quantum dot semiconductor nanocrystals for immunophenotyping by polychromatic flow cytometry," *Nat. Med.*, vol. 12, no. 8, pp. 972–977, 2006.

[81] M. Werner-washburne, E. Braun, G. C. Johnston, and R. A. Singer, "Stationary Phase in the Yeast Saccharomyces cerevisiae," *Microbiol. Rev.*, vol. 57, no. 2, pp. 383–401, 1993.

[82] A. E. Wentz and E. V Shusta, "A Novel High-Throughput Screen Reveals Yeast Genes That Increase Secretion of Heterologous Proteins," *Appl. Environ. Microbiol.*, vol. 73, no. 4, pp. 1189–1198, 2007.

[83] F. M. Klis, "Dynamics of cell wall structure in Saccharomyces cerevisiae," *FEMS Microbiol. Rev.*, vol. 26, pp. 239–56, 2002.

[84] T. L. Orr-weaver, J. W. Szostak, and R. J. Rothsteint, "Yeast transformation : A model system for the study of recombination," *PNAS*, vol. 78, no. 10, pp. 6354–6358, 1981.

[85] M. C. Lorenz, R. S. Muir, E. Lim, J. Mcelver, J. Heitman, and S. C. Weber, "Gene disruption with PCR products in Saccharomyces cerevisiae," *Gene*, vol. 158, pp. 113–117, 1995.

[86] K. R. Oldenburg, K. T. Vo, S. Michaelis, and C. Paddon, "Recombination-

mediated PCR-directed plasmid construction in vivo in yeast," *Nucleic Acids Res.*, vol. 25, no. 2, pp. 451–452, 1997.

[87] J. S. Swers, B. A. Kellogg, and K. D. Wittrup, "Shuffled antibody libraries created by in vivo homologous recombination and yeast surface display," *Nucleic Acids Res.*, vol. 32, no. 3, pp. 1–8, 2004.

[88] D. G. Gibson *et al.*, "One-step assembly in yeast of 25 overlapping DNA fragments to form a complete synthetic Mycoplasma genitalium genome," *PNAS*, vol. 105, no. 51, pp. 20404–20409, 2008.

[89] Z. Shao, H. Zhao, and H. Zhao, "DNA assembler , an in vivo genetic method for rapid construction of biochemical pathways," *Nucleic Acids Res.*, vol. 37, no. 2, pp. 1–10, 2009.

[90] C. Engler, R. Gruetzner, R. Kandzia, and S. Marillonnet, "Golden Gate Shuffling : A One-Pot DNA Shuffling Method Based on Type IIs Restriction Enzymes," *PlosOne*, vol. 4, no. 5, 2009.

[91] C. Engler, R. Kandzia, and S. Marillonnet, "A One Pot , One Step , Precision Cloning Method with High Throughput Capability," *PlosOne*, vol. 3, no. 11, 2008.

[92] E. Weber, R. Gruetzner, S. Werner, C. Engler, and S. Marillonnet, "Assembly of Designer TAL Effectors by Golden Gate Cloning," *PlosOne*, vol. 6, no. 5, 2011.

[93] D. G. Gibson *et al.*, "Enzymatic assembly of DNA molecules up to several hundred kilobases," *Nat. Methods*, vol. 6, no. 5, pp. 12–16, 2009.

[94] P. D. Holler, P. O. Holman, E. V Shusta, S. O. Herrin, K. D. Wittrup, and D. M. Kranz, "In vitro evolution of a T cell receptor with high affinity for peptide ¯ MHC," *PNAS*, vol. 97, no. 10, pp. 5387–5392, 2000.

[95] J. Wrammert *et al.*, "Rapid cloning of high-affinity human monoclonal antibodies against influenza virus," *Nature*, vol. 453, no. May, 2008.

[96] D. Orlic *et al.*, "Bone marrow cells regenerate infarcted myocardium," *Nature*, vol. 410, no. April, pp. 701–705, 2001.

[97] E. T. Boder, K. S. Midelfort, and K. D. Wittrup, "Directed evolution of antibody fragments with monovalent femtomolar antigen-binding affinity," *PNAS*, vol. 2000, no. 20, pp. 1–5, 2000.

[98] L. A. Hulett, H.R., Bonner, W.A., Barrett, J., Herzenberg, "Cell Sorting : Automated Separation of Mammalian Cells as a Function of Intracellular Fluorescence," *Science (80-.).*, vol. 166, no. 3906, pp. 747–749, 1969.

[99] X. Chen, J. L. Zaro, and W. Shen, "Fusion protein linkers : Property , design and functionality," *Adv. Drug Deliv. Rev.*, vol. 65, no. 10, pp. 1357–1369, 2013.

[100] L. A. Wagner, R. B. Weiss, R. Driscoll, D. S. Dunn, and R. F. Gesteland, "Transcriptional slippage occurs during elongation at runs of adenine or thymine in Escherichia coli," *Nucleic Acids Res.*, vol. 18, no. 12, pp. 3529–35, 1990.

[101] C. P. Verschoor, A. Lelic, J. L. Bramson, D. M. E. Bowdish, and H. Maecker, "An introduction to automated flow cytometry gating tools and their implementation," *Front. Immunol.*, vol. 6, no. July, pp. 1–9, 2015.

[102] N. Aghaeepour *et al.*, "Critical assessment of automated flow cytometry data analysis techniques," *Nat. Methods*, vol. 10, no. 3, 2013.

[103] R. O'Neill, K., Aghaeepour, N., Spidlen, J., and Brinkman, "Flow Cytometry Bioinformatics," *Plos Comput. Biol.*, vol. 9, no. 12, 2013.

[104] F. Hahne *et al.*, "flowCore : a Bioconductor package for high throughput flow cytometry," *BMC Bioinformatics*, vol. 10, no. 106, 2009.

[105] K. Lo, F. Hahne, R. R. Brinkman, and R. Gottardo, "flowClust : a Bioconductor package for automated gating of flow cytometry data," *BMC Bioinformatics*, vol. 10, no. 145, 2009.

[106] N. Aghaeepour, R. Nikolic, H. H. Hoos, and R. R. Brinkman, "Rapid Cell Population Identification in Flow Cytometry Data," *Cytom. Part A*, vol. 79, pp. 6–13, 2011.

[107] P. Qiu *et al.*, "Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE," *Nat. Biotechnol.*, vol. 29, no. 10, pp. 886–891, 2011.

[108] I. P. Sugár and S. C. Sealfon, "Misty Mountain clustering : application to fast unsupervised flow cytometry gating," *BMC Bioinformatics*, vol. 11, no. 502, 2010.

[109] Y. Ge and S. C. Sealfon, "flowPeaks : a fast unsupervised clustering for flow cytometry data via K -means and density peak finding," vol. 28, no. 15, pp. 2052–2058, 2012.

[110] Y. Qian *et al.*, “Elucidation of Seventeen Human Peripheral Blood B-Cell Subsets and Quantification of the Tetanus Response Using a Density-Based Method for the Automated Identification of Cell Populations in Multidimensional Flow Cytometry Data,” *Cytom. Part B*, vol. 82, no. May, pp. 69–82, 2010.

[111] M. Ester, H. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[112] U. Naumann and M. P. Wand, “Automation in High-Content Flow Cytometry Screening,” *Cytom. Part A*, vol. 75, pp. 789–797, 2009.

[113] D. R. Parks, M. Roederer, and W. A. Moore, “A New ‘“ Logicle ”’ Display Method Avoids Deceptive Effects of Logarithmic Scaling for Low Signals and Compensated Data,” *Cytom. Part A*, vol. 551, no. April, pp. 541–551, 2006.

[114] M. P. Chacon, J.E., Duong, T., Wand, “Asymptotics for General Multivariate Kernel Density Derivative Estimators,” *Stat. Sin.*, vol. 21, no. 2, pp. 807–840, 2011.

[115] B. Stanley, “Molecular Biology of Prion Diseases,” *Science (80-.).*, vol. 252, pp. 1515–22, 1991.

[116] J. Collinge, “Prion Diseases of Humans and Animals: Their Causes and Molecular Basis,” *Annu. Rev. Neurosci.*, vol. 24, pp. 519–50, 2001.

[117] S. B. Prusiner, “Novel Proteinaceous Infectious Particles Cause Scrapie,” *Science (80-.).*, vol. 216, pp. 136–144, 1982.

[118] S. Prusiner *et al.*, “Transgenetic Studies Implicate Interactions between Homologous PrP lsoforms in Scrapie Prion Replication,” *Cell*, vol. 63, pp. 673–86, 1990.

[119] J. Masel, V. A. A. Jansen, and M. A. Nowak, “Quantifying the kinetic parameters of prion replication,” *Biophys. Chem.*, vol. 77, pp. 139–152, 1999.

[120] H. Biieler, A. Aguui, A. Sailer, and R. Greiner, “Mice Devoid of PrP Are Resistant to Scrapie,” *Cell*, vol. 73, pp. 1339–1347, 1993.

[121] H. Budka, “Neuropathological Diagnostic Criteria for Creutzfeldt-Jakob Disease (CJD) and Other Human Spongiform Encephalopathies (Prion Diseases),” *Brain Pathol.*, vol. 5, no. 4, pp. 459–66, 1995.

[122] C. Song *et al.*, “Effect of intraventricular infusion of anti-prion protein monoclonal antibodies on disease progression in prion-infected mice,” *J. Gen. Virol.*, no. 2008, pp. 1533–1544, 2017.

[123] D. Peretz *et al.*, “Antibodies inhibit prion propagation and clear cell cultures of prion infectivity,” *Nature*, vol. 412, no. August, pp. 739–743, 2001.

[124] M. Enari, E. Flechsig, and C. Weissmann, “Scrapie prion protein accumulation by scrapie- infected neuroblastoma cells abrogated by exposure to a prion protein antibody,” *PNAS*, vol. 98, no. 16, pp. 9295–9299, 2001.

[125] G. Donofrio, F. L. Heppner, M. Polymenidou, C. Musahl, and A. Aguzzi, “Paracrine Inhibition of Prion Propagation by Anti-PrP Single-Chain Fv Miniantibodies,” *J. Virol.*, vol. 79, no. 13, pp. 8330–8338, 2005.

[126] A. R. White, P. Enever, M. Tayebi, and R. Mushens, “Monoclonal antibodies inhibit prion replication and delay the development of prion disease,” *Nature*, vol. 422, no. March, pp. 18–21, 2003.

[127] F. Moda *et al.*, “Brain delivery of AAV9 expressing an anti-PrP monovalent antibody delays prion disease in mice,” *Prion*, no. October, pp. 383–390, 2012.

[128] Y. J. Yu *et al.*, “Boosting Brain Uptake of a Therapeutic Antibody by Reducing Its Affinity for a Transcytosis Target,” *Sci. Transl. Med.*, vol. 3, no. 84, 2011.

[129] T. Igawa *et al.*, “letters Antibody recycling by engineered pH-dependent antigen binding improves the duration of antigen neutralization,” *Nat. Biotechnol.*, vol. 28, no. 11, pp. 1203–1207, 2010.

[130] W. F. Dall’Acqua, P. A. Kiener, and H. Wu, “Properties of Human IgG1s Engineered for Enhanced Binding to the Neonatal Fc Receptor (FcRn),” *JBC*, vol. 281, no. 33, pp. 23514–23524, 2006.

[131] J. S. Huston *et al.*, “Protein engineering of antibody binding sites : Recovery of specific activity in an anti-digoxin single-chain Fv analogue produced in Escherichia coli,” *PNAS*, vol. 85, no. August, pp. 5879–5883, 1988.

[132] M. W. Gonzalez and M. G. Kann, “Chapter 4: Protein Interactions and Disease,” *PLoS Comput. Biol.*, vol. 8, no. 12, 2012.

[133] J. Chothia, C. and Janin, “Principles of protein-protein recognition,” *Nature*, pp. 705–8, 1975.

[134] A. A. Bogan and K. S. Thorn, “Anatomy of Hot Spots in Protein Interfaces,”

JMB, vol. 280, pp. 1–9, 1998.

[135] J. Homola, "Surface Plasmon Resonance Sensors for Detection of Chemical and Biological Species," *Chem. Rev.*, vol. 108, pp. 462–493, 2008.

[136] M. M. Pierce, C. S. Raman, and B. T. Nall, "Isothermal Titration Calorimetry of Protein – Protein Interactions," *Methods*, vol. 221, pp. 213–221, 1999.

[137] T. Berggård, S. Linse, and P. James, "Methods for the detection and analysis of protein – protein interactions," *Proteomics*, vol. 7, pp. 2833–2842, 2007.

[138] E. A. Jares-Erijman and T. M. Jovin, "FRET imaging," *Nat. Biotechnol.*, vol. 21, no. 11, pp. 1387–1395, 2003.

[139] L. Gu, C. Li, J. Aach, D. E. Hill, M. Vidal, and G. M. Church, "Multiplex single-molecule interaction profiling of DNA-barcoded proteins," *Nature*, vol. 515, no. 7528, pp. 554–557, 2014.

[140] A. P. Frei *et al.*, "Highly multiplexed simultaneous detection of RNAs and proteins in single cells," *Nat. Methods*, vol. 13, no. 3, pp. 269–77, 2016.

[141] P. Uetz *et al.*, "A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae," *Nature*, vol. 403, no. February, 2000.

[142] S. Paliwal, P. A. Iglesias, K. Campbell, Z. Hilioti, A. Groisman, and A. Levchenko, "MAPK-mediated bimodal gene expression and adaptive gradient sensing in yeast," *Nature*, vol. 446, no. March, 2007.

[143] S. L. Spencer, S. Gaudet, J. G. Albeck, J. M. Burke, and P. K. Sorger, "Non-genetic origins of cell-to-cell variability in," *Nature*, vol. 459, no. 7245, pp. 428–432, 2009.

[144] R. A. Burrell, N. Mcgranahan, J. Bartek, and C. Swanton, "The causes and consequences of genetic," *Nature*, vol. 501, pp. 338–45, 2013.

[145] H. H. Chang, M. Hemberg, M. Barahona, D. E. Ingber, and S. Huang, "Transcriptome-wide noise controls lineage choice in mammalian progenitor cells," *Nature*, vol. 453, no. May, pp. 4–8, 2008.

[146] R. Zhang, H. Yuan, S. Wang, Q. Ouyang, Y. Chen, and N. Hao, "High-throughput single-cell analysis for the proteomic dynamics study of the yeast osmotic stress response," *Sci. Rep.*, 2017.

[147] M. E. Birnbaum *et al.*, "Deconstructing the Peptide-MHC Specificity of T Cell Recognition," *Cell*, vol. 157, pp. 1073–1087, 2014.

[148] K. D. Boder, E.T. and Wittrup, “Yeast Surface Display for Directed Evolution of Protein Expression , Affinity , and Stability,” *Methods Enzymol.*, vol. 328, no. 1999, pp. 430–444, 2000.

[149] M. G. Paez-segala *et al.*, “Fixation-resistant photoactivatable fluorescent proteins for CLEM,” *Nat. Methods*, vol. 12, no. 3, 2015.

[150] B. G. Reid and G. C. Flynn, “Chromophore Formation in Green Fluorescent Protein,” *Biochemistry*, vol. 97403, no. 97, pp. 6786–6791, 1997.

[151] G. M. Benson, R.C., Meyer, R.A., Zaruba, M.E., and McKhann, “Cellular autofluorescence - Is it due to flavins?,” *J. Histochem. Cytochem.*, vol. 27, pp. 44–48, 1979.

[152] S. Luikenhuis, G. Perrone, I. W. Dawes, and C. M. Grant, “The Yeast Saccharomyces cerevisiae Contains Two Glutaredoxin Genes That Are Required for Protection against Reactive Oxygen Species,” *Mol. Biol. Cell*, vol. 9, pp. 1081–1091, 1998.

[153] C. Godon *et al.*, “The H2O2 Stimulon in Saccharomyces cerevisiae,” *J. Biol. Chem.*, vol. 273, no. 34, pp. 22480–22489, 1998.

[154] R. Boorsteins and A. Craigsv, “Structure and Regulation of the SSA4 HSP70 Gene of Saccharomyces cerevisiae,” *J. Biol. Chem.*, vol. 26, no. 31, pp. 18912–18921, 1990.

[155] D. Stanley, A. Bandara, S. Fraser, P. J. Chambers, and G. A. Stanley, “The ethanol stress response and ethanol tolerance of Saccharomyces cerevisiae,” *J. Appl. Microbiol.*, vol. 109, no. Demain 2009, pp. 13–24, 2010.

[156] L. A. Sturtz, K. Diekert, L. T. Jensen, R. Lill, and V. Cizewski, “A Fraction of Yeast Cu , Zn-Superoxide Dismutase and Its Metallochaperone , CCS , Localize to the Intermembrane Space of Mitochondria,” *J. Biol. Chem.*, vol. 276, no. 41, pp. 38084–38089, 2001.

[157] A. Belle, A. Tanay, L. Bitincka, R. Shamir, and E. K. O. Shea, “Quantification of protein half-lives in the budding yeast proteome,” *PNAS*, vol. 103, no. 35, pp. 13004–9, 2006.

[158] A. Påhlman, K. Granath, R. Ansell, S. Hohmann, and L. Adler, “The Yeast Glycerol 3-Phosphatases Gpp1p and Gpp2p Are Required for Glycerol Biosynthesis and Differentially Involved in the Cellular Responses to Osmotic , Anaerobic , and Oxidative Stress,” *J. Biol. Chem.*, vol. 276, no. 5, pp. 3555–3563, 2001.

[159] B. J. Hackel, D. Huang, J. C. Bubolz, X. X. Wang, and E. V Shusta, "Production of Soluble and Active Transferrin Receptor-Targeting Single-Chain Antibody using Saccharomyces cerevisiae," *Pharmaceultical Res.*, vol. 23, no. 4, pp. 790–797, 2006.

[160] E. P. Miller, M.J., Xuong, N., and Geiduschek, "Quantitative Analysis of the Heat Shock Response of Saccharomyces cerevisiae," *J. Bacteriol.*, vol. 151, no. 1, pp. 311–327, 1982.

[161] S. J. Fleishman *et al.*, "Computational Design of Proteins Targeting the Conserved Stem Region of Influenza Hemagglutinin," *Science (80-.).*, vol. 979, no. May, pp. 816–822, 2011.

[162] T. A. Whitehead *et al.*, "Optimization of affinity , specificity and function of designed influenza inhibitors using deep sequencing," *Nat. Biotechnol.*, vol. 30, no. 6, pp. 543–548, 2012.

[163] K. M. Doolan and D. W. Colby, "Conformation-Dependent Epitopes Recognized by Prion Protein Antibodies Probed Using Mutational Scanning and Deep Sequencing," *J. Mol. Biol.*, vol. 427, no. 2, pp. 328–340, 2015.

[164] R. Moretti *et al.*, "Community-wide evaluation of methods for predicting the effect of mutations on protein–protein interactions," *Proteins*, no. July, pp. 1980–1987, 2013.

Appendix A

ADDITIONAL CONTRIBUTORS

Joe Reynolds constructed the first pBC1 plasmid containing a GPD promoter, multiple cloning site, C-terminal alpha-agglutinin protein, and a CYC1 terminator.

Erin Aho created pCTCON2-1HA, 2HA, 3HA, 4HA, and 5HA constructs using a PCR based expansion method. She found that 1HA and 4HA gave distinct fluorescence intensities using 'ideal' antibodies, i.e. high brightness with secondary labeling

Seth Ritter developed the idea for the exponential expansion of repeating DNA sequences. He also constructed some of the single epitope repeat constructs (**Appendix C, Table C.1**)

Quentin Dubroff created some of the barcode plasmids consisting of epitope tag combinations, that were used in the beginning of this project (**Appendix C, Table C.1**)

Greg Vorsanger implemented the LALIGN algorithm and wrote the Python scripts for analysis of the SMRT data with direction from Stefanie Berges. Stefanie wrote Python scripts to generate graphics from analyzed data. Greg also assisted in the development of the barcode identification software, specifically with the decision to use DBSCAN clustering and Gaussian Kernel Density Estimation.

Olga Morozova and Stefanie Berges both contributed to the production and purification of the recombinant mouse prion protein. Kyle McHugh refolded and labeled the protein with Alexa Fluor 647 NHS ester.

Appendix B

SOFTWARE FOR ANALYSIS OF SMRT SEQUENCING DATA

```
#!/bin/bash

script="fasta/fasta-36.3.8e/bin/lalign36 -d 0 -m 9 -E 0.0001 -n -f 12 -g 4"
echo "Aligning GLUGLU"
time $script 3passes_CCS.fasta ref_sequences/GluGlu_G4S.fasta >
data/3passes_full_GluGlu_G4S
echo "Aligning HA"
time $script 3passes_CCS.fasta ref_sequences/HA_G4S.fasta >
data/3passes_full_HA_G4S
echo "Aligning AU1"
time $script 3passes_CCS.fasta ref_sequences/AU1_G4S.fasta >
data/3passes_full_AU1_G4S
echo "Aligning FLAG"
time $script 3passes_CCS.fasta ref_sequences/FLAG_G4S.fasta >
data/3passes_full_FLAG_G4S
echo "Aligning HIS"
time $script 3passes_CCS.fasta ref_sequences/HIS_G4S.fasta >
data/3passes_full_HIS_G4S
echo "Aligning HSV"
time $script 3passes_CCS.fasta ref_sequences/HSV_G4S.fasta >
data/3passes_full_HSV_G4S
echo "Aligning CMYC"
time $script 3passes_CCS.fasta ref_sequences/CMYC_G4S.fasta >
data/3passes_full_CMYC_G4S
echo "Done Aligning."
```

```
#!/bin/sh

echo "HA"
python process_local_align.py --afile data/3passes_full_HA_G4S  --name HA_G4S --
lfile ref_sequences/length_ref_seqs.json --tfile ref_sequences/ref_len_thresh.json

echo "CMYC"
python process_local_align.py --afile data/3passes_full_CMYC_G4S  --name
CMYC_G4S --lfile ref_sequences/length_ref_seqs.json --tfile
ref_sequences/ref_len_thresh.json

echo "FLAG"
python process_local_align.py --afile data/3passes_full_FLAG_G4S  --name
FLAG_G4S --lfile ref_sequences/length_ref_seqs.json --tfile
ref_sequences/ref_len_thresh.json

echo "GLUGLU"
python process_local_align.py --afile data/3passes_full_GluGlu_G4S  --name
GluGlu_G4S --lfile ref_sequences/length_ref_seqs.json --tfile
ref_sequences/ref_len_thresh.json

echo "HIS"
python process_local_align.py --afile data/3passes_full_HIS_G4S  --name HIS_G4S -
-lfile ref_sequences/length_ref_seqs.json --tfile ref_sequences/ref_len_thresh.json

echo "HSV"
python process_local_align.py --afile data/3passes_full_HSV_G4S  --name HSV_G4S
--lfile ref_sequences/length_ref_seqs.json --tfile ref_sequences/ref_len_thresh.json

echo "AGAlpha1"
python process_local_align.py --afile data/3passes_full_AGAlpha1 --name AGAlpha1
--lfile ref_sequences/length_ref_seqs.json --tfile ref_sequences/ref_len_thresh.json

echo "AU1"
python process_local_align.py --afile data/3passes_full_AU1_G4S  --name AU1_G4S
--lfile ref_sequences/length_ref_seqs.json --tfile ref_sequences/ref_len_thresh.json
```

```
import argparse
import requests
import numpy
import json
import time
import os.path
from collections import defaultdict
from pprint import pprint
def get_ref_dict(lenfile):
   with open(lenfile,'r') as data:
      len_dict = json.load(data)
#     pprint(len_dict)
      return len_dict

def get_matrices_from_file(filename):
   matrix_dict = {}
   curr_matrix = ""
   with open(filename,'r') as f:
      in_matrix = False
      curr_name = ""
      for line in f:
         if ">>>m" in line:
            curr_name = line[6:]
#        print line
         if in_matrix:
            if ">>><<<" in line:
               in_matrix = False
               matrix_dict[curr_name] = curr_matrix
            else:
               curr_matrix += line
         elif "%_id  %_sim  lsw  alen  an0  ax0  pn0  px0  an1  ax1 pn1 px1
gapq gapl  fs" in line:
            in_matrix = True
            curr_matrix = ""
   return matrix_dict

def make_list_matrix(matrix):
   #print "MATRIX"
   #print matrix
   #print "END MATRIX"
   i = 0
   #dynamic based on length returned
   mat_len = len(matrix.split('\n')) -1
   #static based on the lalign program
   mat_width = 6
   mat_list = numpy.zeros((mat_len,18))
   for line in matrix.split('\n'):
      line = line.replace("\t"," ")
      j =0
      split_spaces = line[69:].split(" ")
      for item in split_spaces:

         if item != '':
            mat_list[i][j] = item
            j+=1
      i+=1
#     print split_spaces
```

```
#  print mat_list
   return mat_list

#matrix, matching threshold, dict1, dict2
#default, 0.85
def analyze_matrix(list_matrix,seq_len,len_thresh,match_thresh = 0.85):
   unique_vals = {}
   top_match = 0.0
   top_e_val = 0.0
   match_count = 0
   for row in list_matrix:
      match_val  = row[3]
      e_val = row [2]
      #if e_Val not in dict, add and grab the match
      if e_val not in unique_vals:
         unique_vals[e_val] = match_val

      #if match val greater, set it to highest and grab eval
      if match_val > top_match:
         if row[6] >= seq_len - len_thresh and row[6] <= seq_len +
len_thresh:
            top_match = match_val
            top_e_val = e_val
      if match_val > match_thresh:
         if row[6] >= seq_len - len_thresh and row[6] <= seq_len +
len_thresh:
            match_count+=1
            #print "GOOD len:", row[6], "vs:" , seq_len
         else:
            pass
            #print "BAD len:", row[6], "vs:" , seq_len

   best_match_dict = {"Count": match_count,"Top E Value":  top_e_val, "Best
Match Value":  top_match}
   return unique_vals,best_match_dict


def run_alignment(filename,run_name,ref_len_name,len_thresh_name):
   #length of reference
   ref_len_dict = get_ref_dict(ref_len_name)
   #maximm length error for alingment
   len_thresh_dict = get_ref_dict(len_thresh_name)
   seq_len = ref_len_dict["Length"][run_name]
   len_thresh = len_thresh_dict["Thresh"][run_name]
   info_dict = {}
   info_dict[run_name] = {}
   matrix_dict = get_matrices_from_file(filename)
   outfile = run_name + "_analysis.json"
   print "This file has:", len(matrix_dict.keys()), "matrices present."
   for key in matrix_dict.keys():
      list_matrix = make_list_matrix(matrix_dict[key])

#     print list_matrix
      unique_dict,best_dict =
analyze_matrix(list_matrix,seq_len,len_thresh,0.90)
      #for testing
```

```
#     print unique_dict,best_dict
      if best_dict["Count"] > 0:
         info_dict[run_name][key] = {}
         info_dict[run_name][key]["Best"] = best_dict
         info_dict[run_name][key]["Unique"] = unique_dict
   output_str = json.dumps(info_dict, sort_keys=True, indent=4,
separators=(',', ': '))
   with open(outfile,'w') as f:
      f.write(output_str)

#for testing only
if __name__ == "__main__":
   parser = argparse.ArgumentParser(description="Give a file to process")
   parser.add_argument('--afile', metavar="f",type = str)
   parser.add_argument('--lfile', metavar="l",type = str)
   parser.add_argument('--name', metavar="n",type = str)
   parser.add_argument('--tfile', metavar='t',type = str)
   args = parser.parse_args()
   if args.afile and args.name and args.lfile and args.tfile:
      run_alignment(args.afile,args.name,args.lfile,args.tfile)
   else:
      print "Need filename, len file name, threshold file name and name of
run!"

#  debug()
```

```
import json
import os
import argparse
from pprint import pprint
#grabs all json, makes one dict with keys based on reference and data within
def make_dict(json_dir):
   alignment_dict = {}
   for name in os.listdir(json_dir):
      print name
      json_file = os.path.join(json_dir,name)
      with open(json_file,'r') as f:
         data = json.load(f)
         alignment_dict = dict(alignment_dict, **data)

#  print alignment_dict.keys()
   for key in alignment_dict.keys():
      print "Reference:",key, "has: " ,  len(alignment_dict[key].keys()),
"sequences"
   return alignment_dict

#get all seq_ids. For now this is hardcoded to use CMYC!!
def get_sequence_ids(ad,list_mode):
   seq_id_list = []
   print "List mode  = ", list_mode
   #ONLY CMYC
   if list_mode == "CMYC":
      raw_seq_list = ad["CMYC_G4S"].keys()
                for key in raw_seq_list:
                   if ad["CMYC_G4S"][key]["Best"]["Count"] > 0:
                #assume no 0's
                      if key not in ad["AGAlpha1"]: #and
ad["AGAlpha1"][key]["Best"]["Count"] == 0:
                                    seq_id_list.append(key)
   #ONLY AGA
   if list_mode == "AGA":
      raw_seq_list = ad["AGAlpha1"].keys()
      for key in raw_seq_list:
         if ad["AGAlpha1"][key]["Best"]["Count"] > 0:
            if key not in ad["CMYC_G4S"]: # and
ad["CMYC_G4S"][key]["Best"]["Count"] == 0:
               seq_id_list.append(key)

   #BOTH
   if list_mode == "CMYC+AGA":
      raw_seq_list = ad["CMYC_G4S"].keys()
      for key in raw_seq_list:
         #if ad["CMYC_G4S"][key]["Best"]["Count"] > 0 and key in
ad["AGAlpha1"] and ad["AGAlpha1"][key]["Best"]["Count"] > 0:
         if key in ad["AGAlpha1"]:
            seq_id_list.append(key)

   if list_mode == "ALL":
      raw_seq_dict = {}
      for ref in ad.keys():
         for key in ad[ref].keys():
```

```
            raw_seq_dict[key] = 1
      seq_id_list = raw_seq_dict.keys()

   if list_mode == "NEITHER":
      raw_seq_dict = {}
      for ref in ad.keys():
         for key in ad[ref].keys():
            raw_seq_dict[key] = 1
      print "TOTAL KEYS:", len(raw_seq_dict.keys())
      for key in ad["CMYC_G4S"].keys():
         if key in ad["AGAlpha1"]:
            del raw_seq_dict[key]
      for key in ad["AGAlpha1"].keys():
         if key in ad["CMYC_G4S"]:
            if key in raw_seq_dict:
               del raw_seq_dict[key]
      seq_id_list = raw_seq_dict.keys()
   return seq_id_list


#get barcodes by seqid. Takes dict from all json files and Returns dict of
seqid : barcode:
def get_barcode_dict(ad,seq_id_list,key_order):
   barcode_dict = {}
   for seq_id in seq_id_list:
      barcode_list = []
      for key in key_order:
         if seq_id in ad[key]:
            barcode_list.append(ad[key][seq_id]["Best"]["Count"])
         else:
            barcode_list.append(0)
      barcode_dict[seq_id] = barcode_list

   return barcode_dict

#gets barcode frequency in data set.
#expects dictionary of seqid : barcode
#returns dictionary with barcode: frequency, count, seqids/ on off
def get_barcode_frequency_dict(bc_dict,total_barcodes, seq_ids = False):
   bc_freq_dict = {}
   for seq_id in bc_dict.keys():
      raw_s = ""
      for item in bc_dict[seq_id]:
         raw_s += str(item)
         raw_s += "-"
      barcode_string = raw_s[:-1]
      #default dict too awkward here.
      if barcode_string in bc_freq_dict:
         bc_freq_dict[barcode_string]["Count"] += 1
         if seq_ids:
            bc_freq_dict[barcode_string]["Seq_IDs"].append(seq_id)
      else:
         bc_freq_dict[barcode_string] = {}
         bc_freq_dict[barcode_string]["Count"] = 1
         if seq_ids:
            bc_freq_dict[barcode_string]["Seq_IDs"] = [seq_id]
```

```
   #at this point we have a dictionary with all the barcodes and counts, need
to get frequencies.
   for key in bc_freq_dict.keys():
      bc_freq_dict[key]["Frequency"] =
float(bc_freq_dict[key]["Count"])/float(total_barcodes)

#  pprint(bc_freq_dict)
   return bc_freq_dict
#takes alignment dict. Returns dict for each ref sequence with key: ref seq :
Number of matches : freq and seqids/onoff
def
get_best_match_dicts(ad,seq_id_list,key_order,total_count,output_name,seq_ids
= False):
   for key in key_order:
      best_match_dict = {}
      #for seq_id in ad[key].keys():
      for seq_id in seq_id_list:
         if seq_id in ad[key]:
            count = ad[key][seq_id]["Best"]["Count"]
            if count in best_match_dict:
               best_match_dict[count]["Count"] += 1
               if seq_ids:
                  best_match_dict[count]["Seq_IDs"].append(seq_id)
            else:
               best_match_dict[count] = {}
               best_match_dict[count]["Count"] = 1
               if seq_ids:
                  best_match_dict[count]["Seq_IDs"] = [seq_id]
      #do frequency
      for match_num in best_match_dict.keys():
         best_match_dict[match_num]["Frequency"] =
float(best_match_dict[match_num]["Count"])/float(total_count)

#     pprint(best_match_dict)
      #print each to own file
      if seq_ids:
         outfile = output_name + "_" + key +
"best_match_table_w_seq_ids.json"
      else:
         outfile = output_name + "_" + key + "best_match_table.json"
      json_to_file(outfile,best_match_dict)

def json_to_file(name, jdict):
   output = json.dumps(jdict, sort_keys = True, indent=4, separators=(',', ':
'))
   with open(name,'w') as f:
      f.write(output)

def run_analysis(json_dir,output_name,list_mode):
   a_dict = make_dict(json_dir)

   #get sequence IDS
   print "Getting sequence ideas for mode:", list_mode
   seq_id_list = get_sequence_ids(a_dict,list_mode)
   print "This mode returned", len(seq_id_list), "ids"
   #hardcode for now for order reasons
   key_order =
```

```
["HA_G4S","HSV_G4S","HIS_G4S","AU1_G4S","GluGlu_G4S","FLAG_G4S"]
   bc_dict = get_barcode_dict(a_dict,seq_id_list,key_order)
   print "bc_dict has this many barcodes:", len(bc_dict.keys())
   bc_file = output_name + "_barcodes_table.json"
   #write barcode dict to file
   json_to_file(bc_file,bc_dict)

   #Get number of barcodes form CMYC file - NOT GENERIC
   total_barcodes = len(seq_id_list)
   #Get barcode count dictionary w/ and w/out seq ids
   bc_count_dict = get_barcode_frequency_dict(bc_dict,total_barcodes)
   bc_count_dict_seqs =
get_barcode_frequency_dict(bc_dict,total_barcodes,True)
   #print to file w/ and w/o seqs
   bcf_file = output_name +  "_barcode_frequency_table.json"
   json_to_file(bcf_file,bc_count_dict)
   bcf_file_w_seqs = output_name + "_barcode_frequency_table_w_seqs.json"
   json_to_file(bcf_file_w_seqs,bc_count_dict_seqs)
   #this calls its own json_to_file

get_best_match_dicts(a_dict,seq_id_list,key_order,len(seq_id_list),output_nam
e)

get_best_match_dicts(a_dict,seq_id_list,key_order,len(seq_id_list),output_nam
e,True)


if __name__ == "__main__":
   parser = argparse.ArgumentParser(description= "Give a directory of JSON to
eat")
   parser.add_argument ("--list_mode",metavar="l",type=str)
   parser.add_argument("--json_dir",metavar="d",type=str)
   parser.add_argument("--output_name",metavar="o",type=str)
   args = parser.parse_args()
   list_mode = "all"
   if args.list_mode:
      list_mode = args.list_mode
   if args.json_dir and args.output_name:
      run_analysis(args.json_dir,args.output_name,list_mode)
   else:
      print "Error, no directory or output name provided!"
      print "List modes: ALL, AGA_only, CMYC_only, AGA+CMYC, NEITHER"
```

```
import numpy as np
import os
import json
from collections import defaultdict
import csv

file_path = "/Users/Stefanie/PycharmProjects/untitled/Pacbio_tables"

if not os.path.exists(file_path):
    os.makedirs(file_path)
os.chdir(file_path)

json_filename = "/Users/Stefanie/Dropbox/Pacbio
Analysis/1120/1120_CYMC+AGA_barcode_frequency_table.json"

barcodetabledict = {}

with open(json_filename,'r') as f:
    barcode_dict = json.load(f)

for barcodeid in barcode_dict.keys():
    count = barcode_dict[barcodeid]['Count']
    barcodetabledict[barcodeid] = count

fields = ['Barcode ID','Count']
with open('Pacbio barcode table.csv','w') as f:
    w = csv.writer(f,fields)
    w.writerow(fields)
    for row in barcodetabledict.items():
        w.writerow([row[0],row[1]])
```

```
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import json
from collections import defaultdict


file_path = "/Users/Stefanie/PycharmProjects/untitled/Pacbio_graphics"


if not os.path.exists(file_path):
    os.makedirs(file_path)
os.chdir(file_path)

tagnames = ['HA','HSV','HIS','GluGlu','AU1','FLAG']
epcolor = ['blue','red','green','purple','orange','magenta']
i = 1
#[fig,ax] = plt.subplots(2,3,sharex=False,sharey=False)

odd_repeats_dict = defaultdict(lambda: defaultdict(float))

for name in tagnames:
    json_filename = "/Users/Stefanie/Dropbox/Pacbio
Analysis/1120/1120_CMYC_"+name+'_G4Sbest_match_table.json'
    with open (json_filename,'r') as f:
        repeat_freq_dict = json.load(f)
    repeat_length_list = []
    counts_list = []
    freq_list = []
    count_odd_repeat_sizes = 0
    total_reads = 0
    for numrep in repeat_freq_dict.keys():
        #print 'repeat',numrep
        count = repeat_freq_dict[numrep]["Count"]
        #print 'count',count
        freq = repeat_freq_dict[numrep]["Frequency"]
        repeat_length_list.append(int(numrep))
        counts_list.append(count)
        freq_list.append(freq)
        total_reads += count
        if numrep not in ['1','2','4','8','16','32','14']:
            count_odd_repeat_sizes += count
    odd_repeats_dict[name]['Count'] = count_odd_repeat_sizes
    odd_repeats_dict[name]['Percent'] =
float(count_odd_repeat_sizes)/float(total_reads)


    counts_list_log = np.log10(counts_list)
    repeat_counts_list = map(lambda x,y:[x,y],repeat_length_list,counts_list)
    repeat_counts_list_log = map(lambda
x,y:[x,y],repeat_length_list,counts_list_log)
    repeat_counts_list.sort(key=lambda x:x[0])
    repeat_counts_list_log.sort(key=lambda x:x[0])
```

```
    xvals = map(lambda x:x[0],repeat_counts_list)
    yvals = map(lambda x:x[1],repeat_counts_list)

    [fig,ax] = plt.subplots()
    ind = np.arange(0,len(repeat_length_list))
    width = 0.5
    c = colors.cnames[epcolor[i-1]]
    plt.bar(ind,yvals,width,color=c)
    ax.set_xticks(ind+width/2)
    ax.set_xticklabels(xvals)
    plt.title(name,fontsize=16)
    plt.xlabel('Repeat length')
    plt.ylabel('Count')
    plt.tight_layout()
    plt.savefig('Repeat counts CMYC only '+name)
    i = i+1
#plt.show()

with open('Pacbio odd repeats.txt','w') as f:

f.write(json.dumps(odd_repeats_dict,sort_keys=True,indent=4,separators=(',',
':')))
```

Appendix C

BARCODE SMRT LIBRARY DATA

Table C.1: Individually constructed barcodes

HA	HSV	HIS	AU1	GLU	FLAG	Barcode Name	Date Constructed	Constructed By
0	0	0	0	0	0	pBC1		Joe Reynolds
1	0	0	0	0	0	1HA	1/14/13	Seth Ritter
2	0	0	0	0	0	2HA	1/16/13	Seth Ritter
4	0	0	0	0	0	4HA	1/21/13	Seth Ritter
8	0	0	0	0	0	8HA	4/8/13	Seth Ritter
16	0	0	0	0	0	16HA	3/19/13	Seth Ritter
32	0	0	0	0	0	32HA	12/3/13	Stefanie Berges
0	0	1	0	0	0	1HIS	1/22/13	Seth Ritter
0	0	2	0	0	0	2HIS	4/21/13	Stefanie Berges
0	0	4	0	0	0	4HIS	5/15/13	Stefanie Berges
0	0	8	0	0	0	8HIS	6/5/13	Stefanie Berges
0	0	16	0	0	0	16HIS	6/11/13	Stefanie Berges
0	0	16	0	0	0	16HIS	10/24/14	Quentin Dubroff
2	0	1	0	0	0	2HA1HIS	5/22/13	Stefanie Berges
2	0	4	0	0	0	2HA4HIS	5/31/13	Stefanie Berges
2	0	8	0	0	0	2HA8HIS	6/11/13	Stefanie Berges
8	0	1	0	0	0	8HA1HIS	5/22/13	Stefanie Berges
8	0	4	0	0	0	8HA4HIS	6/5/13	Stefanie Berges
8	0	8	0	0	0	8HA8HIS	6/11/13	Stefanie Berges
0	0	0	1	0	0	1AU1	7/25/13	Seth Ritter
0	0	0	2	0	0	2AU1	9/13/13	Seth Ritter
0	0	0	4	0	0	4AU1	9/25/13	Stefanie Berges
0	0	0	8	0	0	8AU1	11/11/13	Stefanie Berges
0	0	0	16	0	0	16AU1	11/11/13	Stefanie Berges
0	0	0	32	0	0	32AU1	12/4/13	Stefanie Berges
0	0	0	0	1	0	1GluGlu	7/25/13	Seth Ritter
0	0	0	0	2	0	2GluGlu	9/25/13	Stefanie Berges
0	0	0	0	4	0	4GluGlu	9/29/13	Seth Ritter
0	0	0	0	8	0	8GluGlu	10/21/13	Seth Ritter
0	0	0	0	16	0	16GluGlu	10/24/13	Stefanie Berges
0	0	0	0	32	0	32GluGlu	11/11/13	Stefanie Berges
0	0	0	0	64	0	64GluGlu	12/4/13	Stefanie Berges
0	1	0	0	0	0	1HSV	10/21/13	Stefanie Berges
0	2	0	0	0	0	2HSV	10/24/13	Stefanie Berges
0	4	0	0	0	0	4HSV	10/25/13	Stefanie Berges
0	8	0	0	0	0	8HSV	11/8/13	Stefanie Berges
0	16	0	0	0	0	16HSV	12/3/13	Stefanie Berges
0	0	0	0	0	1	1FLAG	7/25/13	Seth Ritter
0	0	0	0	0	2	2FLAG	9/25/13	Stefanie Berges
0	0	0	0	0	4	4FLAG	10/8/13	Stefanie Berges
0	0	0	0	0	8	8FLAG	10/18/13	Stefanie Berges
0	0	0	0	0	16	16FLAG	11/7/13	Stefanie Berges
0	0	0	0	0	32	32FLAG	12/3/13	Stefanie Berges
1	0	4	0	0	0	1HA4HIS	1/16/14	Stefanie Berges
4	0	4	0	0	0	4HA4HIS	1/16/14	Stefanie Berges
16	0	4	0	0	0	16HA4HIS	1/19/14	Stefanie Berges
1	0	0	1	0	0	1HA1AU1	1/13/14	Stefanie Berges
4	0	0	1	0	0	4HA1AU1	1/13/14	Stefanie Berges
16	0	0	1	0	0	16HA1AU1	8/25/14	Stefanie Berges
16	0	0	1	0	0	16HA1AU1	1/13/14	Stefanie Berges
16	0	0	4	0	0	16HA4AU1	5/23/14	Quentin Dubroff
16	0	0	8	0	0	16HA8AU1	9/9/14	Quentin Dubroff
16	0	0	8	0	0	16HA8AU1	4/22/14	Quentin Dubroff
16	0	0	16	0	0	16HA16AU1	11/6/14	Stefanie Berges
16	0	0	16	0	0	16HA16AU1	5/6/14	Quentin Dubroff
0	0	4	1	0	0	4HIS1AU1	1/16/14	Stefanie Berges
0	0	4	4	0	0	4HIS4AU1	2/7/14	Stefanie Berges
0	0	4	8	0	0	4HIS8AU1	5/13/14	Quentin Dubroff
0	0	4	16	0	0	4HIS16AU1	9/9/14	Quentin Dubroff
0	0	4	16	0	0	4HIS16AU1	4/23/14	Quentin Dubroff

1	0	0	4	0	0	1HA4AU1	2/7/14	Stefanie Berges
1	0	0	8	0	0	1HA8AU1	3/6/14	Quentin Dubroff
1	0	0	16	0	0	1HA16AU1	10/14/14	Quentin Dubroff
4	0	0	4	0	0	4HA4AU1	2/11/14	Stefanie Berges
4	0	0	8	0	0	4HA8AU1	2/7/14	Stefanie Berges
4	0	0	16	0	0	4HA16AU1	8/25/14	Stefanie Berges
4	0	0	16	0	0	4HA16AU1	4/18/14	Quentin Dubroff
1	0	4	1	0	0	1HA4HIS1AU1	1/29/14	Quentin Dubroff
4	0	4	1	0	0	4HA4HIS1AU1	1/19/14	Stefanie Berges
16	0	4	1	0	0	16HA4HIS1AU1	7/15/14	Quentin Dubroff
16	0	4	1	0	0	16HA4HIS1AU1	1/29/14	Quentin Dubroff
16	0	4	1	0	0	16HA4HIS1AU1	9/2/14	Stefanie Berges
1	0	4	4	0	0	1HA4HIS4AU1	2/7/14	Stefanie Berges
4	0	4	4	0	0	4HA4HIS4AU1	2/7/14	Stefanie Berges
16	0	4	4	0	0	16HA4HIS4AU1	10/14/14	Quentin Dubroff
16	0	4	4	0	0	16HA4HIS4AU1	2/7/14	Stefanie Berges
1	0	4	8	0	0	1HA4HIS8AU1	3/18/14	Quentin Dubroff
1	0	4	16	0	0	1HA4HIS16AU1	8/25/14	Stefanie Berges
1	0	4	16	0	0	1HA4HIS16AU1	3/18/14	Quentin Dubroff
4	0	4	8	0	0	4HA4HIS8AU1	7/30/14	Quentin Dubroff
4	0	4	16	0	0	4HA4HIS16AU1	10/14/14	Quentin Dubroff
4	0	4	16	0	0	4HA4HIS16AU1	4/23/14	Quentin Dubroff
16	0	4	8	0	0	16HA4HIS8AU1	9/9/14	Quentin Dubroff
16	0	4	16	0	0	16HA4HIS16AU1	10/14/14	Quentin Dubroff
1	0	4	1	1	0	1HA4HIS1AU11GluGlu	6/26/14	Quentin Dubroff
0	0	0	8	1	0	8AU11GluGlu	6/26/14	Quentin Dubroff
0	0	4	0	1	0	4HIS1GluGlu	6/26/14	Quentin Dubroff
1	0	0	8	1	0	1HA8AU11GluGlu	6/26/14	Quentin Dubroff
0	0	4	1	1	0	4HIS1AU11GluGlu	6/26/14	Quentin Dubroff
1	0	4	0	1	0	1HA4HIS1GluGlu	6/26/14	Quentin Dubroff
4	0	4	4	1	0	4HA4HIS4AU11GluGlu	10/10/14	Quentin Dubroff
1	0	4	4	1	0	1HA4HIS4AU11GluGlu	10/10/14	Quentin Dubroff
0	0	0	1	1	0	1AU11GluGlu	10/10/14	Quentin Dubroff
0	0	0	1	16	0	1AU116GluGlu	11/20/14	Quentin Dubroff
0	0	0	4	1	0	4AU11GluGlu	6/26/14	Quentin Dubroff
0	0	0	4	16	0	4AU116GluGlu	11/20/14	Quentin Dubroff
0	0	0	16	1	0	16AU11GluGlu	12/19/14	Stefanie Berges
0	0	0	16	16	0	16AU116GluGlu	11/20/14	Quentin Dubroff
1	0	0	0	1	0	1HA1GluGlu	6/26/14	Quentin Dubroff
1	0	0	0	16	0	1HA16GluGlu	11/20/14	Quentin Dubroff
4	0	0	0	1	0	4HA1GluGlu	10/10/14	Quentin Dubroff
4	0	0	0	16	0	4HA16GluGlu	11/20/14	Quentin Dubroff
16	0	0	0	1	0	16HA1GluGlu	11/20/14	Quentin Dubroff
16	0	0	0	16	0	16HA16GluGlu	12/19/14	Stefanie Berges
1	0	0	1	1	0	1HA1AU11GluGlu	10/10/14	Quentin Dubroff
1	0	0	1	16	0	1HA1AU116GluGlu	11/20/14	Quentin Dubroff
1	0	0	4	1	0	1HA4AU11GluGlu	6/26/14	Quentin Dubroff
1	0	0	4	16	0	1HA4AU116GluGlu	11/20/14	Quentin Dubroff
1	0	0	16	1	0	1HA16AU11GluGlu	11/20/14	Quentin Dubroff
1	0	0	16	16	0	1HA16AU116GluGlu	11/20/14	Quentin Dubroff
4	0	0	1	1	0	4HA1AU11GluGlu	11/20/14	Quentin Dubroff
4	0	0	1	16	0	4HA1AU116GluGlu	11/20/14	Quentin Dubroff
4	0	0	4	1	0	4HA4AU11GluGlu	11/20/14	Quentin Dubroff
4	0	0	4	16	0	4HA4AU116GluGlu	11/20/14	Quentin Dubroff
4	0	0	16	1	0	4HA16AU11GluGlu	11/20/14	Quentin Dubroff
4	0	0	16	16	0	4HA16AU116GluGlu	12/19/14	Stefanie Berges
16	0	0	1	1	0	16HA1AU11GluGlu	11/20/14	Quentin Dubroff
16	0	0	1	16	0	16HA1AU116GluGlu	11/20/14	Quentin Dubroff
16	0	0	4	1	0	16HA4AU11GluGlu	10/10/14	Quentin Dubroff
16	0	0	4	16	0	16HA4AU116GluGlu	12/19/14	Stefanie Berges

Table C.2: Abundance of SMRT reads with nonstandard repeat lengths

Epitope	Count	Percent
AU1	170	0.26
FLAG	13	0.45
GLU	153	0.38
HA	108	0.14
HIS	31	0.11
HSV	29	0.48

Table C.3: Unique barcodes found in SMRT sample

Barcode ID	# CCS Reads
4-0-0-0-14-0	328
1-0-0-2-14-0	319
0-16-0-0-0-0	283
4-0-0-1-14-0	244
1-0-0-4-14-0	193
8-0-8-0-0-0	191
4-0-4-8-0-0	172
16-0-0-0-0-0	166
1-0-0-16-0-0	160
0-0-0-4-14-0	158
1-0-0-1-14-0	141
0-0-0-16-1-0	137
13-0-3-0-1-0	137
0-0-0-16-0-0	133
1-0-0-16-1-0	130
1-0-0-16-14-0	124
16-0-0-1-0-0	116
16-0-0-1-1-0	116
0-0-16-0-0-0	109
4-0-4-16-0-0	104
0-0-0-0-0-16	102
0-0-4-16-0-0	94
4-0-4-4-0-0	87
0-0-0-1-14-0	76
1-0-0-0-12-0	76
16-0-4-0-0-0	72
16-0-4-1-0-0	70
1-0-0-0-14-0	65
0-0-4-8-0-0	64
4-0-0-16-0-0	63
4-0-0-4-14-0	62
16-0-0-4-1-0	60
16-0-0-4-0-0	60
0-0-0-0-14-0	59
1-0-4-14-0-0	59
4-0-4-4-1-0	58
8-0-4-0-0-0	58
16-0-0-8-0-0	57
1-0-0-4-1-0	54
1-0-0-4-0-0	54
0-15-0-0-0-0	51
1-0-0-3-14-0	50
16-0-4-4-0-0	45
1-0-4-8-0-0	44
4-0-0-16-1-0	38
4-0-0-8-0-0	33
16-0-4-5-0-0	32
0-0-0-0-28-0	31
8-0-8-1-0-0	30
12-0-3-0-1-0	28
0-8-0-0-0-0	28
0-2-0-0-0-0	27
1-0-0-2-13-0	27
16-0-0-2-0-0	27
0-0-0-0-0-0	26
16-0-4-8-0-0	26
4-0-0-16-14-0	24
13-0-3-1-1-0	22
0-0-0-0-0-4	21
0-0-4-0-0-0	21
4-0-0-2-14-0	21
0-14-0-0-0-0	21
1-0-4-4-0-0	20
16-0-0-2-1-0	20
0-0-0-0-4-0	20
1-0-4-4-1-0	18
4-0-0-0-13-0	18
16-0-0-9-0-0	18
0-0-8-0-0-0	18
1-0-4-1-0-0	18
1-0-0-15-0-0	18
1-0-0-15-1-0	18
1-0-0-8-0-0	18
15-0-4-1-0-0	17
0-0-4-15-0-0	17
1-0-0-2-12-0	16
0-0-0-15-1-0	16
0-0-4-14-0-0	16
3-0-0-0-0-0	16
4-0-4-1-0-0	16
16-0-0-16-1-0	16
0-1-0-0-0-0	15
0-0-0-0-1-0	15
16-0-4-2-0-0	15
8-0-1-0-0-0	15
0-0-0-4-0-0	14
0-0-0-15-14-0	14
15-0-0-0-0-0	14
0-0-0-15-0-0	13
4-0-0-1-13-0	13
0-0-0-0-0-2	13
1-0-0-1-0-0	13
15-0-0-1-0-0	13
1-0-0-8-1-0	13
16-0-0-4-14-0	12
0-0-0-2-0-0	12
0-11-0-0-0-0	12
16-0-4-9-0-0	12
0-0-4-1-0-0	12
1-0-0-11-0-0	1

Barcode ID	# CCS Reads
16-0-4-16-0-0	11
4-0-0-1-12-0	11
4-0-4-0-0-0	11
16-0-0-5-0-0	11
16-0-0-0-14-0	11
0-0-4-9-0-0	11
0-0-0-0-0-15	11
0-0-0-32-0-0	10
4-0-4-5-1-0	10
1-0-0-0-0-0	10
4-0-0-15-0-0	10
2-0-4-0-0-0	10
0-12-0-0-0-0	10
0-0-1-0-0-0	10
0-4-0-0-0-0	10
4-0-0-0-0-0	10
0-0-0-1-1-0	9
1-0-0-1-9-0	9
4-0-0-0-12-0	9
0-0-0-0-2-0	9
1-0-0-1-1-0	9
16-0-0-3-0-0	9
1-0-0-15-14-0	9
2-0-0-0-0-0	9
15-0-0-8-0-0	9
2-0-8-0-0-0	9
4-0-4-17-0-0	9
16-0-0-5-1-0	9
0-0-0-4-13-0	9
0-0-0-8-1-0	8
2-0-1-0-0-0	8
4-0-4-7-0-0	8
0-0-0-14-0-0	8
0-0-0-0-0-32	8
1-0-4-13-0-0	8
1-0-1-0-0-0	8
0-0-15-0-0-0	8
16-0-0-17-1-0	8
1-0-0-1-13-0	8
1-0-4-0-1-0	8
0-0-4-1-1-0	8
4-0-0-4-0-0	8
0-0-0-14-1-0	8
4-0-0-1-1-0	8
4-0-4-5-0-0	8
9-0-0-3-1-0	8
16-0-0-3-1-0	8
4-0-0-1-0-0	8
4-0-0-0-1-0	7
4-0-0-14-0-0	7
4-0-4-9-0-0	7
8-0-7-0-0-0	7
1-0-4-2-1-0	7
15-0-0-4-0-0	7
0-10-0-0-0-0	7
0-0-0-1-13-0	6
11-0-3-0-1-0	6
0-0-0-4-1-0	6
0-0-2-0-0-0	6
14-0-0-1-1-0	6
1-0-0-14-0-0	6
8-0-8-2-0-0	6
1-0-0-5-0-0	6
4-0-4-15-0-0	6
4-0-0-9-0-0	6
4-0-4-6-0-0	6
0-0-4-4-0-0	6
0-0-0-0-0-14	6
0-0-0-8-0-0	6
7-0-8-0-0-0	6
8-0-0-0-0-0	5
15-0-4-0-0-0	5
0-0-0-10-0-0	5
1-0-0-3-1-0	5
14-0-0-0-0-0	5
0-0-0-0-0-8	5
4-0-0-17-0-0	5
0-0-0-0-7-0	5
1-0-0-6-14-0	5
1-0-0-11-14-0	5
15-0-0-5-0-0	5
0-0-0-1-0-0	5
13-0-3-2-1-0	5
1-0-4-0-0-0	5
1-0-0-14-14-0	5
12-0-3-1-1-0	5
15-0-0-1-1-0	5
16-0-0-10-0-0	5
15-0-4-4-0-0	5
14-0-0-4-0-0	5
16-0-4-3-0-0	5
4-0-0-5-14-0	5
0-0-4-0-1-0	5
15-0-4-2-0-0	5
1-0-0-3-12-0	4
1-0-0-9-14-0	4
1-0-0-3-11-0	4
4-0-0-4-13-0	4
16-0-0-2-14-0	4
0-0-0-0-50-0	4

Barcode ID	# CCS Reads
0-0-0-0-0-1	4
1-0-0-17-1-0	4
0-13-0-0-0-0	4
4-0-0-0-15-0	4
15-0-0-17-1-0	4
0-0-0-6-14-0	4
1-0-0-0-13-0	4
0-0-0-0-26-0	4
14-0-0-1-0-0	4
0-0-0-2-14-0	4
0-0-0-0-21-0	4
15-0-0-4-1-0	4
3-0-2-9-0-0	4
16-0-0-18-1-0	4
16-0-0-6-0-0	4
1-0-0-1-11-0	4
0-0-0-0-27-0	4
1-0-0-14-13-0	4
16-0-0-6-1-0	4
0-0-4-12-0-0	4
15-0-4-8-0-0	4
4-0-0-15-1-0	4
4-0-0-4-1-0	4
1-0-0-4-13-0	4
4-0-0-2-1-0	4
10-0-0-0-0-0	4
0-0-0-5-14-0	4
0-9-0-0-0-0	4
1-0-0-0-11-0	4
2-0-0-4-0-0	4
7-0-4-0-0-0	3
0-0-4-6-0-0	3
14-0-4-0-0-0	3
1-0-0-16-13-0	3
4-0-4-12-0-0	3
1-0-0-3-13-0	3
1-0-0-5-14-0	3
12-0-4-1-0-0	3
1-0-0-2-1-0	3
0-0-0-17-0-0	3
1-0-0-14-1-0	3
16-0-4-11-0-0	3
14-0-4-1-0-0	3
1-0-0-13-0-0	3
15-0-0-2-0-0	3
12-0-4-0-0-0	3
13-0-2-0-1-0	3
1-0-0-17-0-0	3
1-0-0-0-10-0	3
0-0-0-0-22-0	3
8-0-1-1-0-0	3
1-0-0-13-14-0	3
4-0-0-12-1-0	3
0-0-0-3-14-0	3
6-0-8-0-0-0	3
0-0-0-13-1-0	3
15-0-0-5-1-0	3
0-0-0-20-0-0	3
1-0-0-15-13-0	3
15-0-4-3-0-0	3
1-0-0-1-12-0	3
10-0-0-4-0-0	3
12-0-0-1-1-0	3
1-0-0-12-1-0	3
16-0-0-0-1-0	3
13-0-0-1-1-0	3
0-0-14-0-0-0	3
12-0-0-4-1-0	3
2-0-0-0-14-0	3
14-0-0-3-0-0	3
14-0-0-4-1-0	3
0-0-4-13-0-0	3
0-0-4-10-0-0	3
16-0-0-11-0-0	3
16-0-4-6-0-0	2
15-0-0-3-0-0	2
4-0-3-7-0-0	2
1-0-0-4-10-0	2
1-0-0-15-12-0	2
1-0-4-15-0-0	2
4-0-0-7-0-0	2
13-0-0-8-0-0	2
4-0-0-17-1-0	2
1-0-4-9-0-0	2
10-0-3-0-1-0	2
0-0-0-4-11-0	2
0-3-0-0-0-0	2
11-0-4-0-0-0	2
4-0-0-6-1-0	2
0-0-0-0-19-0	2
1-0-0-0-8-0	2
0-0-0-11-0-0	2
4-0-0-15-14-0	2
4-0-4-3-0-0	2
14-0-4-5-0-0	2
1-0-0-2-10-0	2
7-0-7-0-0-0	2
6-0-0-1-14-0	2
0-0-0-0-0-5	2
0-0-0-0-0-30	2

Barcode ID	# CCS Reads
0-0-0-0-0-31	2
4-0-0-6-0-0	2
13-0-0-9-0-0	2
8-0-4-1-0-0	2
12-0-0-0-0-0	2
1-0-0-3-0-0	2
0-0-0-25-0-0	2
1-0-0-7-14-0	2
1-0-0-26-0-0	2
15-0-0-9-0-0	2
0-0-0-8-14-0	2
11-0-3-1-1-0	2
16-0-4-7-0-0	2
13-0-0-4-0-0	2
10-0-4-4-0-0	2
2-0-0-1-0-0	2
15-0-0-6-0-0	2
3-0-3-7-0-0	2
14-0-0-2-0-0	2
0-0-0-13-0-0	2
16-0-4-10-0-0	2
0-0-0-2-10-0	2
11-0-2-0-1-0	2
9-0-0-4-1-0	2
16-0-0-1-14-0	2
15-0-0-6-1-0	2
4-0-0-1-10-0	2
3-0-0-1-12-0	2
0-7-0-0-0-0	2
5-0-0-0-14-0	2
9-0-0-4-0-0	2
4-0-4-14-0-0	2
4-0-0-2-12-0	2
0-0-0-0-20-0	2
8-0-5-0-0-0	2
0-0-0-15-13-0	2
0-0-0-31-0-0	2
12-0-3-2-1-0	2
13-0-0-0-0-0	2
4-0-0-17-14-0	2
7-0-3-0-1-0	2
15-0-0-1-14-0	2
7-0-0-2-0-0	2
3-0-0-1-10-0	2
3-0-0-16-0-0	2
1-0-0-4-15-0	2
15-0-0-2-1-0	2
0-0-0-12-0-0	2
3-0-0-4-14-0	2
8-0-4-2-0-0	2
0-0-0-0-17-0	2
4-0-0-2-0-0	2
1-0-0-4-12-0	2
4-0-0-1-9-0	2
0-0-4-7-0-0	2
1-0-0-8-14-0	2
13-0-4-1-0-0	2
0-0-0-9-14-0	2
3-0-0-4-0-0	2
3-0-0-14-0-0	2
4-0-0-14-1-0	2
8-0-0-16-1-0	2
0-0-0-5-19-0	1
0-0-0-0-16-0	1
16-0-0-14-0-0	1
2-0-0-0-24-0	1
2-0-0-4-28-0	1
4-0-0-1-18-0	1
12-0-2-0-1-0	1
2-0-6-0-0-0	1
13-0-0-16-1-0	1
4-0-0-1-7-0	1
4-0-0-5-12-0	1
6-0-6-1-0-0	1
1-0-0-16-10-0	1
14-0-3-1-1-0	1
1-0-0-11-8-0	1
5-0-4-15-0-0	1
3-0-4-4-1-0	1
8-0-0-1-14-0	1
6-0-0-6-0-0	1
3-0-4-11-0-0	1
2-0-0-3-0-0	1
0-0-0-10-9-0	1
14-0-3-0-0-0	1
8-0-0-3-28-0	1
6-0-0-1-0-0	1
5-0-0-15-1-0	1
0-6-0-0-0-0	1
24-0-0-2-2-0	1
17-0-7-14-1-0	1
1-0-0-17-14-0	1
1-0-0-2-11-0	1
0-0-0-3-11-0	1
1-0-0-2-0-0	1
1-0-4-3-0-0	1
6-0-0-0-14-0	1
4-0-3-8-0-0	1
4-0-0-19-1-0	1
0-0-0-10-4-0	1

Barcode ID	# CCS Reads
4-0-0-0-14-0	328
1-0-0-2-14-0	319
0-16-0-0-0-0	283
4-0-0-1-14-0	244
1-0-0-4-14-0	193
8-0-8-0-0-0	191
4-0-4-8-0-0	172
16-0-0-0-0-0	166
1-0-0-16-0-0	160
0-0-0-4-14-0	158
1-0-0-1-14-0	141
0-0-0-16-1-0	137
13-0-3-0-1-0	137
0-0-0-16-0-0	133
1-0-0-16-1-0	130
1-0-0-16-14-0	124
16-0-0-1-0-0	116
16-0-0-1-1-0	116
0-0-16-0-0-0	109
4-0-4-16-0-0	104
0-0-0-0-0-16	102
0-0-4-16-0-0	94
4-0-4-4-0-0	87
0-0-0-1-14-0	76
1-0-0-0-12-0	76
16-0-4-0-0-0	72
16-0-4-1-0-0	70
1-0-0-0-14-0	65
0-0-4-8-0-0	64
4-0-0-16-0-0	63
4-0-0-4-14-0	62
16-0-0-4-1-0	60
16-0-0-4-0-0	60
0-0-0-0-14-0	59
1-0-4-14-0-0	59
4-0-4-4-1-0	58
8-0-4-0-0-0	58
16-0-0-8-0-0	57
1-0-0-4-1-0	54
1-0-0-4-0-0	54
0-15-0-0-0-0	51
1-0-0-3-14-0	50
16-0-4-4-0-0	45
1-0-4-8-0-0	44
4-0-0-16-1-0	38
4-0-0-8-0-0	33
16-0-4-5-0-0	32
0-0-0-0-28-0	31
8-0-8-1-0-0	30
12-0-3-0-1-0	28
0-8-0-0-0-0	28
0-2-0-0-0-0	27
1-0-0-2-13-0	27
16-0-0-2-0-0	27
0-0-0-0-0-0	26
16-0-4-8-0-0	26
4-0-0-16-14-0	24
13-0-3-1-1-0	22
0-0-0-0-0-4	21
0-0-4-0-0-0	21
4-0-0-2-14-0	21
0-14-0-0-0-0	21
1-0-4-4-0-0	20
16-0-0-2-1-0	20
0-0-0-0-4-0	20
1-0-4-4-1-0	18
4-0-0-0-13-0	18
16-0-0-9-0-0	18
0-0-8-0-0-0	18
1-0-4-1-0-0	18
1-0-0-15-0-0	18
1-0-0-15-1-0	18
1-0-0-8-0-0	18
15-0-4-1-0-0	17
0-0-4-15-0-0	17
1-0-0-2-12-0	16
0-0-0-15-1-0	16
0-0-4-14-0-0	16
3-0-0-0-0-0	16
4-0-4-1-0-0	16
16-0-0-16-1-0	16
0-1-0-0-0-0	15
0-0-0-0-1-0	15
16-0-4-2-0-0	15
8-0-1-0-0-0	15
0-0-0-4-0-0	14
0-0-0-15-14-0	14
15-0-0-0-0-0	14
0-0-0-15-0-0	13
4-0-0-1-13-0	13
0-0-0-0-0-2	13
1-0-0-1-0-0	13
15-0-0-1-0-0	13
1-0-0-8-1-0	13
16-0-0-4-14-0	12
0-0-0-2-0-0	12
0-11-0-0-0-0	12
16-0-4-9-0-0	12
0-0-4-1-0-0	12
1-0-0-11-0-0	1

Barcode ID	# CCS Reads
16-0-4-16-0-0	11
4-0-0-1-12-0	11
4-0-4-0-0-0	11
16-0-0-5-0-0	11
16-0-0-0-14-0	11
0-0-4-9-0-0	11
0-0-0-0-0-15	11
0-0-0-32-0-0	10
4-0-4-5-1-0	10
1-0-0-0-0-0	10
4-0-0-15-0-0	10
2-0-4-0-0-0	10
0-12-0-0-0-0	10
0-0-1-0-0-0	10
0-4-0-0-0-0	10
4-0-0-0-0-0	10
0-0-0-1-1-0	9
1-0-0-1-9-0	9
4-0-0-0-12-0	9
0-0-0-0-2-0	9
1-0-0-1-1-0	9
16-0-0-3-0-0	9
1-0-0-15-14-0	9
2-0-0-0-0-0	9
15-0-0-8-0-0	9
2-0-8-0-0-0	9
4-0-4-17-0-0	9
16-0-0-5-1-0	9
0-0-0-4-13-0	9
0-0-0-8-1-0	8
2-0-1-0-0-0	8
4-0-4-7-0-0	8
0-0-0-14-0-0	8
0-0-0-0-0-32	8
1-0-4-13-0-0	8
1-0-1-0-0-0	8
0-0-15-0-0-0	8
16-0-0-17-1-0	8
1-0-0-1-13-0	8
1-0-4-0-1-0	8
0-0-4-1-1-0	8
4-0-0-4-0-0	8
0-0-0-14-1-0	8
4-0-0-1-1-0	8
4-0-4-5-0-0	8
9-0-0-3-1-0	8
16-0-0-3-1-0	8
4-0-0-1-0-0	8
4-0-0-0-1-0	7
4-0-0-14-0-0	7
4-0-4-9-0-0	7
8-0-7-0-0-0	7
1-0-4-2-1-0	7
15-0-0-4-0-0	7
0-10-0-0-0-0	7
0-0-0-1-13-0	6
11-0-3-0-1-0	6
0-0-0-4-1-0	6
0-0-2-0-0-0	6
14-0-0-1-1-0	6
1-0-0-14-0-0	6
8-0-8-2-0-0	6
1-0-0-5-0-0	6
4-0-4-15-0-0	6
4-0-0-9-0-0	6
4-0-4-6-0-0	6
0-0-4-4-0-0	6
0-0-0-0-0-14	6
0-0-0-8-0-0	6
7-0-8-0-0-0	6
8-0-0-0-0-0	5
15-0-4-0-0-0	5
0-0-0-10-0-0	5
1-0-0-3-1-0	5
14-0-0-0-0-0	5
0-0-0-0-0-8	5
4-0-0-17-0-0	5
0-0-0-0-7-0	5
1-0-0-6-14-0	5
1-0-0-11-14-0	5
15-0-0-5-0-0	5
0-0-0-1-0-0	5
13-0-3-2-1-0	5
1-0-4-0-0-0	5
1-0-0-14-14-0	5
12-0-3-1-1-0	5
15-0-0-1-1-0	5
16-0-0-10-0-0	5
15-0-4-4-0-0	5
14-0-0-4-0-0	5
16-0-4-3-0-0	5
4-0-0-5-14-0	5
0-0-4-0-1-0	5
15-0-4-2-0-0	5
1-0-0-3-12-0	4
1-0-0-9-14-0	4
1-0-0-3-11-0	4
4-0-0-4-13-0	4
16-0-0-2-14-0	4
0-0-0-0-50-0	4

Barcode ID	# CCS Reads
0-0-0-0-0-1	4
1-0-0-17-1-0	4
0-13-0-0-0-0	4
4-0-0-0-15-0	4
15-0-0-17-1-0	4
0-0-0-6-14-0	4
1-0-0-0-13-0	4
0-0-0-0-26-0	4
14-0-0-1-0-0	4
0-0-0-2-14-0	4
0-0-0-0-21-0	4
15-0-0-4-1-0	4
3-0-2-9-0-0	4
16-0-0-18-1-0	4
16-0-0-6-0-0	4
1-0-0-1-11-0	4
0-0-0-0-27-0	4
1-0-0-14-13-0	4
16-0-0-6-1-0	4
0-0-4-12-0-0	4
15-0-4-8-0-0	4
4-0-0-15-1-0	4
4-0-0-4-1-0	4
1-0-0-4-13-0	4
4-0-0-2-1-0	4
10-0-0-0-0-0	4
0-0-0-5-14-0	4
0-9-0-0-0-0	4
1-0-0-0-11-0	4
2-0-0-4-0-0	4
7-0-4-0-0-0	3
0-0-4-6-0-0	3
14-0-4-0-0-0	3
1-0-0-16-13-0	3
4-0-4-12-0-0	3
1-0-0-3-13-0	3
1-0-0-5-14-0	3
12-0-4-1-0-0	3
1-0-0-2-1-0	3
0-0-0-17-0-0	3
1-0-0-14-1-0	3
16-0-4-11-0-0	3
14-0-4-1-0-0	3
1-0-0-13-0-0	3
15-0-0-2-0-0	3
12-0-4-0-0-0	3
13-0-2-0-1-0	3
1-0-0-17-0-0	3
1-0-0-0-10-0	3
0-0-0-0-22-0	3
8-0-1-1-0-0	3
1-0-0-13-14-0	3
4-0-0-12-1-0	3
0-0-0-3-14-0	3
6-0-8-0-0-0	3
0-0-0-13-1-0	3
15-0-0-5-1-0	3
0-0-0-20-0-0	3
1-0-0-15-13-0	3
15-0-4-3-0-0	3
1-0-0-1-12-0	3
10-0-0-4-0-0	3
12-0-0-1-1-0	3
1-0-0-12-1-0	3
16-0-0-0-1-0	3
13-0-0-1-1-0	3
0-0-14-0-0-0	3
12-0-0-4-1-0	3
2-0-0-0-14-0	3
14-0-0-3-0-0	3
14-0-0-4-1-0	3
0-0-4-13-0-0	3
4-0-4-10-0-0	3
16-0-0-11-0-0	3
16-0-4-6-0-0	2
15-0-0-3-0-0	2
4-0-3-7-0-0	2
1-0-0-4-10-0	2
1-0-0-15-12-0	2
1-0-4-15-0-0	2
4-0-0-7-0-0	2
13-0-0-8-0-0	2
4-0-0-17-1-0	2
1-0-4-9-0-0	2
10-0-3-0-1-0	2
0-0-0-4-11-0	2
0-3-0-0-0-0	2
11-0-4-0-0-0	2
4-0-0-6-1-0	2
0-0-0-0-19-0	2
1-0-0-0-8-0	2
0-0-0-11-0-0	2
4-0-0-15-14-0	2
4-0-4-3-0-0	2
14-0-4-5-0-0	2
1-0-0-2-10-0	2
7-0-7-0-0-0	2
6-0-0-1-14-0	2
0-0-0-0-0-5	2
0-0-0-0-0-30	2

Barcode ID	# CCS Reads
0-0-0-0-0-31	2
4-0-0-6-0-0	2
13-0-0-9-0-0	2
8-0-4-1-0-0	2
12-0-0-0-0-0	2
1-0-0-3-0-0	2
0-0-0-25-0-0	2
1-0-0-7-14-0	2
1-0-0-26-0-0	2
15-0-0-9-0-0	2
0-0-0-8-14-0	2
11-0-3-1-1-0	2
16-0-4-7-0-0	2
13-0-0-4-0-0	2
10-0-4-4-0-0	2
2-0-0-1-0-0	2
15-0-0-6-0-0	2
3-0-3-7-0-0	2
14-0-0-2-0-0	2
0-0-0-13-0-0	2
16-0-4-10-0-0	2
0-0-0-2-10-0	2
11-0-2-0-1-0	2
9-0-0-4-1-0	2
16-0-0-1-14-0	2
15-0-0-6-1-0	2
4-0-0-1-10-0	2
3-0-0-1-12-0	2
0-7-0-0-0-0	2
5-0-0-0-14-0	2
9-0-0-4-0-0	2
4-0-4-14-0-0	2
4-0-0-2-12-0	2
0-0-0-0-20-0	2
8-0-5-0-0-0	2
0-0-0-15-13-0	2
0-0-0-31-0-0	2
12-0-3-2-1-0	2
13-0-0-0-0-0	2
4-0-0-17-14-0	2
7-0-3-0-1-0	2
15-0-0-1-14-0	2
7-0-0-2-0-0	2
3-0-0-1-10-0	2
3-0-0-16-0-0	2
1-0-0-4-15-0	2
15-0-0-2-1-0	2
0-0-0-12-0-0	2
3-0-0-4-14-0	2
8-0-4-2-0-0	2
0-0-0-0-17-0	2
4-0-0-2-0-0	2
1-0-0-4-12-0	2
4-0-0-1-9-0	2
0-0-4-7-0-0	2
1-0-0-8-14-0	2
13-0-4-1-0-0	2
0-0-0-9-14-0	2
3-0-0-4-0-0	2
3-0-0-14-0-0	2
4-0-0-14-1-0	2
8-0-0-16-1-0	2
0-0-0-5-19-0	1
0-0-0-0-16-0	1
16-0-0-14-0-0	1
2-0-0-0-24-0	1
2-0-0-4-28-0	1
4-0-0-1-18-0	1
12-0-2-0-1-0	1
2-0-6-0-0-0	1
13-0-0-16-1-0	1
4-0-0-1-7-0	1
4-0-0-5-12-0	1
6-0-6-1-0-0	1
1-0-0-16-10-0	1
14-0-3-1-1-0	1
1-0-0-11-8-0	1
5-0-4-15-0-0	1
3-0-4-4-1-0	1
8-0-0-1-14-0	1
6-0-0-6-0-0	1
3-0-4-11-0-0	1
2-0-0-3-0-0	1
0-0-0-10-9-0	1
14-0-3-0-0-0	1
8-0-0-3-28-0	1
6-0-0-1-0-0	1
5-0-0-15-1-0	1
0-6-0-0-0-0	1
24-0-0-2-2-0	1
17-0-7-14-1-0	1
1-0-0-17-14-0	1
1-0-0-2-11-0	1
0-0-0-3-11-0	1
1-0-0-2-0-0	1
1-0-4-3-0-0	1
6-0-0-0-14-0	1
4-0-3-8-0-0	1
4-0-0-19-1-0	1
0-0-0-10-4-0	1

Appendix D

SINGLE-CELL DETECTION AND QUANTIFICATION OF ENDOGENOUS LOW ABUNDANCE REPEAT PROTEINS IN YEAST USING FLOW CYTOMETRY

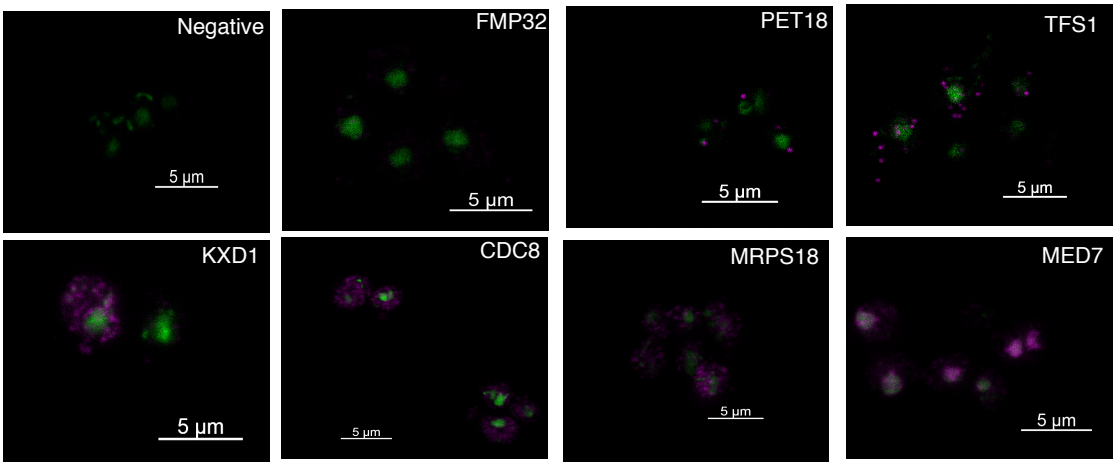


Figure D.1: 16FLAG fusions enable detection of low abundance endogenous proteins by confocal fluorescence microscopy. Fluorescent confocal images of yeast cells taken with a 63x oil lens. Protein expression and localization is shown in pink and the nucleus is shown in green.

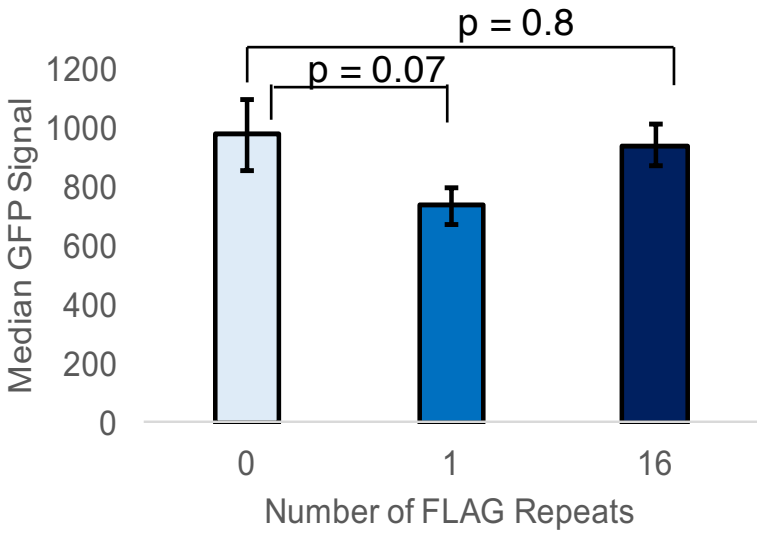


Figure D.2: Expression of GFP with or without FLAG fusion. GFP signal was measured by flow cytometry for surface-displayed alpha-agglutinin GFP fusion proteins also expressing 0, 1, or 16 FLAG repeats. GFP signal did not decrease significantly when FLAG fusions were added, suggesting FLAG fusions do not alter protein expression levels.

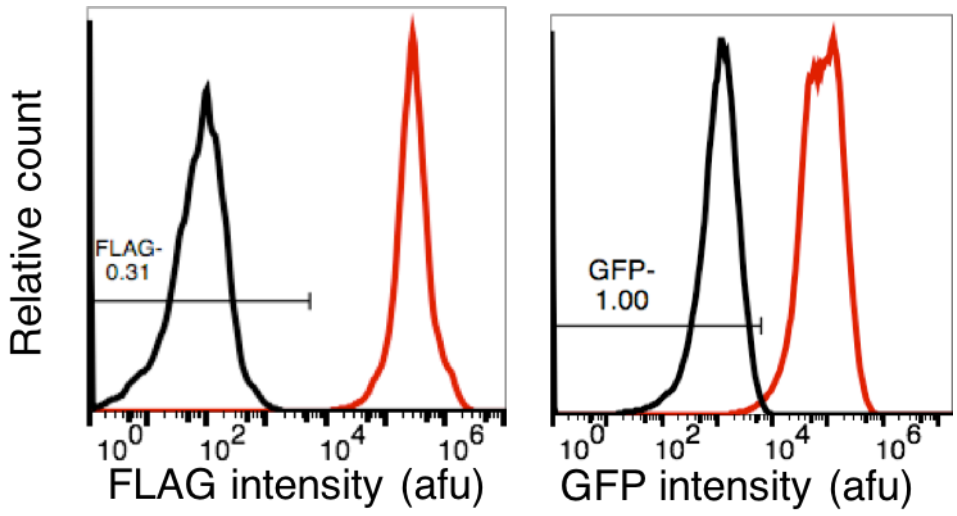


Figure D.3: Detection of a highly expressed endogenous yeast protein by FLAG or GFP fusion. The THD3 protein was detected in yeast using flow cytometry by fusion to either 16FLAG or a GFP, showing a low rate of false negatives and false positives with either detection method.

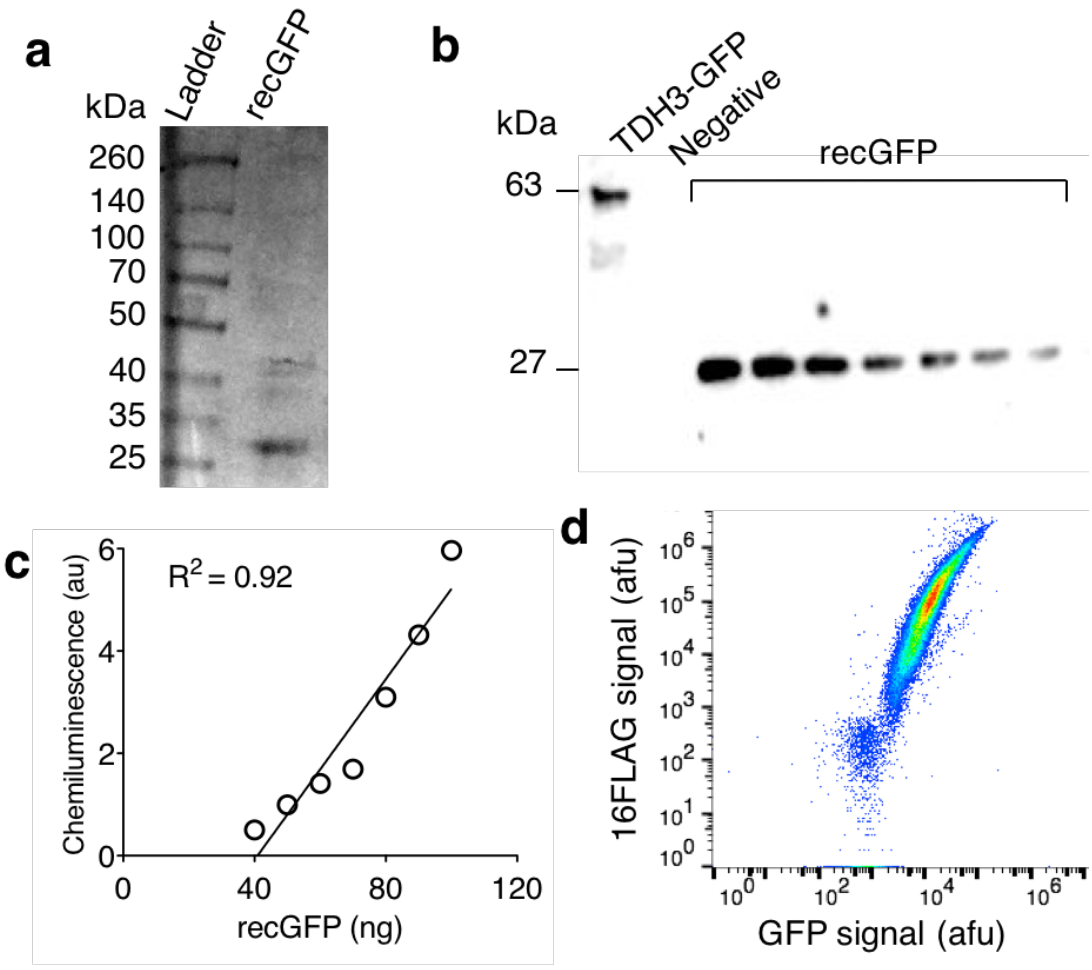


Figure D.4: Conversion of GFP and 16FLAG signals from arbitrary fluorescence units to protein abundance**.** (**a**) recGFP purity was estimated to be 90% by Coomassie stain. (**b**) Western blot of TDH3-GFP whole cell lysate and a standard curve of purified recGFP. (**c**) Quantification of (b) estimates the abundance of TDH3-GFP as 2.7 million molecules per cell. A relationship between median arbitrary GFP fluorescence and GFP protein abundance was determined by linear regression (GFP molecules per cell*10^6 = 8.81*Median GFP fluorescence – 6,741. (**d**) A relationship between molecules per cell and median 16FLAG signal was determined by immunolabeling cells expressing a GFP and 16FLAG AGα1 fusion protein. The relationship between molecules per cell and 16FLAG signal was related by linear regression to be 1.41*16FLAG signal – 199.63.

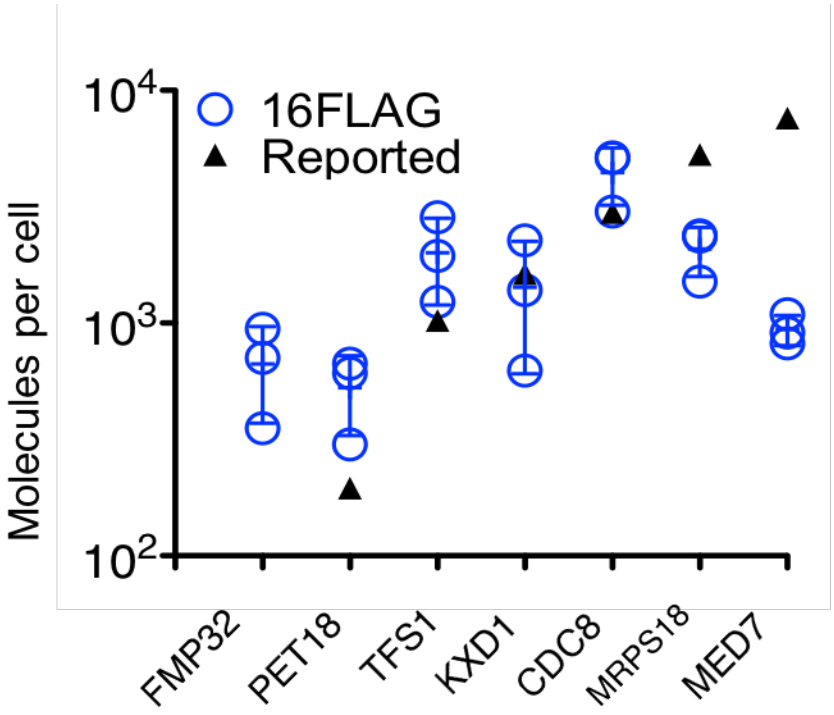


Figure D.5: Quantification of protein abundance. Protein abundance was quantified by relating protein expression detected by flow cytometry and Western blotting. Our results agree somewhat with previous reports ($R^2 = 0.24$).

Appendix E

ASSESSMENT OF RELATIVE ABUNDANCE OF BARCODES IN 7-COLOR LIBRARIES

Table E.1: Relative abundance of barcodes in 7-color library before FACS sorting

HA	HSV	HIS	AU1	GLU	FLAG	% Abundance
4	0	0	0	0	0	6.774
1	0	0	1	1	0	5.478
0	4	0	0	0	0	5.022
4	0	0	1	0	1	4.252
0	0	1	1	0	1	3.868
4	0	1	0	1	0	3.638
0	4	0	1	0	1	3.587
0	4	1	0	1	0	3.548
0	1	0	0	0	0	2.848
1	4	1	0	1	0	2.751
1	4	0	0	0	0	2.718
1	0	1	1	1	0	2.632
0	0	0	1	0	0	2.342
0	0	1	1	1	0	2.191
0	0	1	1	0	0	2.065
1	1	0	1	1	0	2.007
0	0	0	0	0	0	1.761
0	1	1	1	0	0	1.644
0	0	0	1	1	0	1.604
4	1	0	1	0	1	1.456
0	4	0	0	1	1	1.398
0	1	1	1	0	1	1.356
0	0	1	0	1	0	1.353
4	1	1	0	1	0	1.353
0	0	1	0	1	1	1.340
1	0	1	1	0	0	1.319
0	4	0	0	0	1	1.191
0	0	1	4	0	0	1.085
1	4	0	0	1	0	1.055
0	0	1	0	0	0	0.984
0	4	0	1	1	0	0.909
1	0	1	0	1	0	0.884
0	0	0	1	0	1	0.733
4	0	1	1	0	1	0.710
0	1	0	1	0	0	0.646
1	1	0	0	1	1	0.582
4	1	1	1	0	0	0.579
4	1	0	1	0	0	0.566
0	1	0	1	0	1	0.554
1	0	0	0	0	0	0.546
0	1	1	0	0	0	0.540
1	0	0	1	1	1	0.531
0	1	0	1	1	0	0.514
0	0	0	0	0	1	0.512
0	0	1	4	1	0	0.489
0	4	1	0	1	1	0.474
0	0	1	4	0	1	0.457
1	0	0	0	1	0	0.440
1	1	1	1	1	0	0.437
0	1	0	0	1	1	0.436
0	1	1	0	1	1	0.415
1	0	0	1	0	1	0.413
4	0	0	1	0	0	0.392
4	0	1	1	1	0	0.390
1	1	1	0	1	0	0.389

HA	HSV	HIS	AU1	GLU	FLAG	% Abundance
0	4	1	1	0	1	0.092
4	4	1	1	1	0	0.090
4	0	1	4	0	0	0.087
0	0	1	1	1	1	0.087
0	0	0	4	0	1	0.086
0	1	0	4	0	0	0.085
1	4	0	4	0	0	0.083
4	4	0	1	0	1	0.079
0	1	0	4	1	0	0.076
1	1	0	1	0	1	0.075
0	4	1	0	0	0	0.072
0	1	1	0	0	1	0.071
1	4	0	0	0	1	0.070
4	1	1	4	0	0	0.069
0	0	0	0	1	0	0.068
1	0	1	0	1	1	0.067
0	1	0	0	1	0	0.064
4	4	0	0	0	1	0.064
4	4	0	0	1	1	0.061
4	1	1	1	1	0	0.060
4	4	0	1	0	0	0.059
0	4	1	4	1	0	0.059
1	1	1	1	0	1	0.054
4	4	1	4	0	0	0.046
4	4	0	1	1	0	0.043
0	0	1	4	1	1	0.042
4	1	1	0	1	1	0.041
4	1	0	1	1	1	0.037
0	4	0	4	1	1	0.036
0	4	0	1	1	1	0.036
1	4	1	1	0	0	0.035
1	0	1	0	0	1	0.035
0	1	1	4	1	0	0.035
1	4	0	1	0	1	0.033
1	4	1	4	1	0	0.032
1	4	1	1	1	1	0.032
4	1	1	0	0	1	0.030
1	4	1	0	0	1	0.029
0	4	1	4	1	1	0.028
4	0	1	0	1	1	0.026
1	4	1	0	0	0	0.025
4	0	1	1	1	1	0.024
1	4	0	4	1	0	0.024
4	0	0	1	1	1	0.024
1	1	0	0	0	1	0.024
0	0	0	4	1	1	0.023
1	0	1	1	1	1	0.023
1	1	1	1	1	1	0.023
4	1	1	1	1	1	0.022
4	4	0	0	1	0	0.022
1	1	1	0	1	1	0.022
4	0	1	0	0	1	0.021
1	0	1	0	0	0	0.021
4	1	1	0	0	0	0.021
1	4	0	1	1	1	0.020

1	1	0	0	1	0	0.375
0	4	1	1	1	1	0.370
1	1	0	1	0	0	0.355
0	1	0	0	0	1	0.328
4	4	0	0	0	0	0.313
0	4	1	1	1	0	0.309
1	1	1	1	0	0	0.297
1	1	0	4	0	0	0.294
4	4	1	0	1	0	0.285
0	0	0	1	1	1	0.282
0	0	0	4	1	0	0.271
0	1	1	1	1	0	0.270
0	0	0	0	1	1	0.253
4	1	1	1	0	1	0.252
4	1	0	0	1	1	0.237
0	0	0	4	0	0	0.236
1	0	0	1	0	0	0.230
4	0	1	1	0	0	0.229
1	4	1	0	1	1	0.212
1	4	0	1	1	0	0.212
1	4	0	1	0	0	0.207
0	1	1	0	1	0	0.198
0	4	1	0	0	1	0.194
1	4	1	1	1	0	0.187
4	1	0	0	0	0	0.184
4	0	0	4	0	1	0.182
1	0	0	0	0	1	0.169
4	0	0	0	0	1	0.166
0	1	1	1	1	1	0.150
4	4	1	1	0	0	0.144
1	1	0	0	0	0	0.144
0	1	1	4	0	0	0.143
0	0	1	0	0	1	0.143
0	4	0	0	1	0	0.142
0	4	0	1	0	0	0.141
1	0	1	1	0	1	0.139
4	4	0	4	0	1	0.139
1	1	1	0	0	0	0.136
1	1	0	1	1	1	0.135
4	4	1	1	0	1	0.135
4	1	0	4	0	1	0.130
1	0	1	4	0	0	0.129
4	4	1	0	1	1	0.121
4	0	1	4	0	1	0.119
0	4	0	4	1	0	0.118
0	1	0	1	1	1	0.117
1	4	0	0	1	1	0.113
4	0	0	0	1	1	0.112
0	4	1	1	0	0	0.108
4	1	1	4	0	1	0.101
0	4	0	4	0	1	0.096
0	4	1	4	0	0	0.096
4	0	1	0	0	0	0.095

4	0	0	1	1	0	0.020
0	4	0	4	0	0	0.018
4	1	0	1	1	0	0.018
4	1	0	0	0	1	0.017
0	1	1	4	0	1	0.016
1	4	1	4	0	0	0.016
1	1	1	0	0	1	0.015
1	0	0	4	0	1	0.014
4	4	1	0	0	1	0.014
1	0	1	4	0	1	0.013
4	4	1	1	1	1	0.012
4	4	1	0	0	0	0.012
4	0	0	4	0	0	0.011
1	0	0	4	1	0	0.011
4	1	0	4	0	0	0.011
4	4	0	1	1	1	0.011
1	1	0	4	1	0	0.008
1	1	1	4	0	0	0.008
0	1	0	4	1	1	0.008
4	1	1	4	1	0	0.007
0	1	1	4	1	1	0.007
1	4	0	4	0	1	0.007
1	4	1	1	0	1	0.007
1	1	1	4	1	0	0.006
0	4	1	4	0	1	0.006
1	0	1	4	1	0	0.006
0	1	0	4	0	1	0.006
1	1	1	4	0	1	0.004
1	1	0	4	0	1	0.004
4	4	0	4	0	0	0.003
4	1	0	0	1	0	0.003
4	4	1	4	0	1	0.003
4	4	1	4	1	0	0.003
4	4	0	4	1	1	0.003
1	4	0	4	1	1	0.002
4	0	1	4	1	0	0.002
4	1	0	4	1	0	0.002
4	4	0	4	1	0	0.002
4	4	1	4	1	1	0.002
1	4	1	4	1	1	0.002
4	0	0	0	1	0	0.001
1	1	1	4	1	1	0.001
1	0	0	4	1	1	0.001
1	0	1	4	1	1	0.001
1	0	0	4	0	0	0.000
4	0	0	4	1	0	0.000
1	1	0	4	1	1	0.000
4	1	0	4	1	1	0.000
1	4	1	4	0	1	0.000
4	0	1	4	1	1	0.000
4	1	1	4	1	1	0.000
4	0	0	4	1	1	0.000
1	0	0	0	1	1	0.000

Table E.2: Relative abundance of barcodes in sorted 7-color libraries

HA	HSV	HIS	AU1	GLU	FLAG	Unsorted	HA+AU1+GLU-sort	HA+AU1-GLU+sort	HSV+ AND (FLAG OR HIS+) SORT	Binary library	Average 5 libraries
4	0	0	0	0	0	6.774	3.006	4.076	0.730	0.000	3.045
1	0	0	1	1	0	3.638	0.000	0.464	0.000	12.063	2.976
0	4	0	0	0	0	5.478	1.797	1.686	3.974	0.000	2.938
4	0	0	1	0	1	0.546	15.037	0.001	0.378	0.000	2.712
0	0	1	1	0	1	3.868	1.089	0.099	3.485	3.001	2.530
4	0	1	0	1	0	0.710	0.000	13.580	0.269	0.000	2.485
0	4	0	1	0	1	0.531	0.157	0.050	7.798	0.000	2.116
0	4	1	0	1	0	1.085	0.011	2.205	5.717	0.000	2.066
0	1	0	0	0	0	4.252	0.296	0.918	1.301	1.938	1.908
1	4	1	0	1	0	0.194	0.008	9.389	1.170	0.000	1.891
1	4	0	0	0	0	0.884	7.094	0.845	0.947	0.000	1.768
1	0	1	1	1	0	0.108	0.000	0.438	0.007	9.702	1.702
0	0	0	1	0	0	3.548	0.614	0.025	0.365	3.426	1.649
0	0	1	1	1	0	3.587	0.009	1.494	1.680	0.000	1.565
0	0	1	1	0	0	2.848	0.700	0.054	1.597	1.684	1.514
1	1	0	1	1	0	1.398	0.000	0.102	0.002	6.839	1.495
0	0	0	0	0	0	2.342	0.099	0.353	0.148	3.837	1.330
0	1	1	1	0	0	0.582	0.151	0.009	3.549	1.634	1.328
0	0	0	1	1	0	5.022	0.003	0.121	0.149	0.000	1.313
4	1	0	1	0	1	0.252	6.911	0.003	0.000	0.000	1.204
0	4	0	0	1	1	0.440	0.011	0.814	3.820	0.000	1.201
0	1	1	1	0	1	0.375	0.112	0.006	3.473	1.326	1.201
0	0	1	0	1	0	0.909	0.008	5.242	0.415	0.000	1.197
4	1	1	0	1	0	0.198	0.000	6.440	0.328	0.000	1.194
0	0	1	0	1	1	0.282	0.000	0.058	0.232	6.386	1.192
1	0	1	1	0	0	0.437	5.278	0.004	0.273	0.649	1.156
0	4	0	0	0	1	0.294	0.390	0.184	3.898	0.000	1.143
0	0	1	4	0	0	2.191	1.270	0.032	1.296	0.000	1.087
1	4	0	0	1	0	0.083	0.050	6.199	0.101	0.000	1.077
0	0	1	0	0	0	2.751	0.111	0.435	0.865	0.300	1.044
0	4	0	1	1	0	2.632	0.033	0.371	1.274	0.000	1.043
1	0	1	0	1	0	0.184	0.002	4.204	0.004	1.822	1.042
0	0	0	1	0	1	2.007	0.913	0.009	1.452	0.000	1.017
4	0	1	1	0	1	0.271	5.312	0.000	0.197	0.000	0.994
0	1	0	1	0	0	1.356	0.482	0.022	0.536	2.312	0.938
1	1	0	0	1	1	0.117	0.000	3.040	0.377	1.711	0.907
4	1	1	1	0	0	0.136	3.942	0.002	0.854	0.000	0.898
4	1	0	1	0	0	0.144	5.058	0.008	0.103	0.000	0.898
0	1	0	1	0	1	1.055	0.498	0.004	0.000	3.151	0.866
1	0	0	0	0	0	1.604	0.297	1.552	0.137	0.701	0.856
0	1	1	0	0	0	0.579	0.039	0.141	2.390	0.494	0.854
1	0	0	1	1	1	0.141	0.003	0.012	0.002	4.722	0.817
0	1	0	1	1	0	2.718	0.002	0.106	0.447	0.000	0.809
0	0	0	0	0	1	1.644	0.083	0.327	0.568	1.058	0.795
1	0	0	0	1	1	1.353	0.001	0.495	0.167	1.941	0.791
0	0	1	4	1	0	2.065	0.000	0.057	1.002	0.000	0.776
0	4	1	0	1	1	0.113	0.014	0.280	2.710	0.000	0.754
0	0	1	4	0	1	1.340	0.564	0.007	1.151	0.000	0.717
1	0	0	0	1	0	0.253	0.000	3.680	0.000	0.233	0.709
1	1	1	1	1	0	0.121	0.001	0.915	0.658	2.197	0.708
0	1	0	0	1	1	0.566	0.001	0.496	1.885	0.000	0.695
0	1	1	0	1	1	0.237	0.001	0.110	2.439	0.000	0.688
1	0	0	1	0	1	0.236	1.276	0.000	0.191	2.173	0.676

4	0	0	1	0	0	1.191	1.805	0.010	0.156	0.000	0.636
4	0	1	1	1	0	0.554	0.000	2.532	0.177	0.000	0.600
1	1	1	0	1	0	0.087	0.004	2.068	0.296	0.918	0.589
1	1	0	0	1	0	0.071	0.000	2.265	0.001	1.156	0.583
0	4	1	1	1	1	0.087	0.001	0.114	2.061	0.000	0.556
1	1	0	1	0	0	0.143	2.697	0.005	0.083	0.200	0.536
0	1	0	0	0	1	0.540	0.062	0.136	0.000	2.218	0.533
4	4	0	0	0	0	0.489	1.642	0.357	0.306	0.000	0.529
0	4	1	1	1	0	0.309	0.004	0.381	1.550	0.000	0.528
1	1	1	1	0	0	0.085	1.784	0.002	0.725	0.154	0.522
1	1	0	4	0	0	0.230	2.472	0.003	0.198	0.000	0.516
4	4	1	0	1	0	0.061	0.003	2.609	0.253	0.000	0.509
0	0	0	1	1	1	1.456	0.007	0.005	0.494	0.000	0.490
0	0	0	4	1	0	1.761	0.000	0.007	0.122	0.000	0.472
0	1	1	1	1	0	0.457	0.000	0.182	1.310	0.000	0.472
0	0	0	0	1	1	1.353	0.001	0.495	0.167	0.000	0.462
4	1	1	1	0	1	0.021	2.538	0.000	0.093	0.000	0.447
4	1	0	0	1	1	0.068	0.002	2.230	0.228	0.000	0.442
0	0	0	4	0	0	1.319	0.310	0.005	0.184	0.000	0.428
1	0	0	1	0	0	0.436	0.399	0.006	0.052	1.425	0.424
4	0	1	1	0	0	0.118	2.239	0.012	0.083	0.000	0.422
1	4	1	0	1	1	0.022	0.005	1.364	0.742	0.000	0.417
1	4	0	1	1	0	0.733	0.079	0.866	0.309	0.000	0.416
1	4	0	1	0	0	0.415	1.771	0.045	0.043	0.000	0.414
0	1	1	0	1	0	0.389	0.000	1.281	0.384	0.000	0.405
0	4	1	0	0	1	0.016	0.021	0.008	1.544	0.000	0.395
1	4	1	1	1	0	0.130	0.020	1.549	0.412	0.000	0.394
4	1	0	0	0	0	0.355	0.615	0.547	0.306	0.000	0.357
4	0	0	4	0	1	0.033	2.021	0.007	0.017	0.000	0.347
1	0	0	0	0	1	0.297	0.026	0.053	0.153	1.310	0.342
4	0	0	0	0	1	0.646	0.368	0.181	0.354	0.000	0.341
0	1	1	1	1	1	0.169	0.000	0.012	1.150	0.000	0.332
4	4	1	1	0	0	0.007	1.923	0.006	0.035	0.000	0.329
1	1	0	0	0	0	0.512	0.235	0.494	0.215	0.148	0.327
0	1	1	4	0	0	0.166	0.065	0.001	1.085	0.000	0.324
0	0	1	0	0	1	0.392	0.006	0.020	0.229	0.899	0.308
0	4	0	0	1	0	0.370	0.036	0.701	0.315	0.000	0.293
0	4	0	1	0	0	0.984	0.076	0.071	0.084	0.000	0.291
1	0	1	1	0	1	0.229	0.349	0.011	0.175	0.779	0.289
4	4	0	4	0	1	0.007	1.432	0.000	0.198	0.000	0.287
1	1	1	0	0	0	0.142	0.031	0.061	0.893	0.066	0.285
1	1	0	1	1	1	0.054	0.003	0.006	0.269	1.160	0.274
4	4	1	1	0	1	0.003	1.003	0.000	0.424	0.000	0.272
4	1	0	4	0	1	0.021	1.473	0.001	0.000	0.000	0.248
1	0	1	4	0	0	0.092	1.140	0.000	0.062	0.000	0.227
4	4	1	0	1	1	0.011	0.004	0.853	0.319	0.000	0.224
4	0	1	4	0	1	0.064	1.132	0.000	0.050	0.000	0.215
0	4	0	4	1	0	0.474	0.000	0.000	0.370	0.000	0.211
0	1	0	1	1	1	0.144	0.002	0.004	0.669	0.000	0.204
1	4	0	0	1	1	0.022	0.003	1.002	0.114	0.000	0.200
4	0	0	0	1	1	0.313	0.006	0.606	0.077	0.000	0.199
0	4	1	1	0	0	0.096	0.092	0.027	0.611	0.000	0.196
4	1	1	4	0	1	0.090	0.009	0.000	0.659	0.000	0.189
0	4	0	4	0	1	0.032	0.009	0.000	0.714	0.000	0.188
0	4	1	4	0	0	0.095	0.006	0.000	0.647	0.000	0.186
4	0	1	0	0	0	0.328	0.081	0.283	0.175	0.000	0.186

0	4	1	1	0	1	0.023	0.020	0.003	0.667	0.000	0.176
4	4	1	1	1	0	0.037	0.000	0.791	0.129	0.000	0.172
4	0	1	4	0	0	0.060	0.900	0.000	0.034	0.000	0.172
0	0	1	1	1	1	0.413	0.000	0.026	0.254	0.000	0.171
0	0	0	4	0	1	0.270	0.401	0.019	0.126	0.000	0.168
0	1	0	4	0	0	0.390	0.100	0.021	0.202	0.000	0.168
1	4	0	4	0	0	0.075	0.883	0.000	0.013	0.000	0.168
4	4	0	1	0	1	0.012	0.749	0.000	0.152	0.000	0.164
0	1	0	4	1	0	0.514	0.000	0.000	0.132	0.000	0.162
1	1	0	1	0	1	0.064	0.480	0.000	0.000	0.399	0.161
0	4	1	0	0	0	0.143	0.077	0.083	0.357	0.000	0.151
0	1	1	0	0	1	0.032	0.003	0.005	0.146	0.609	0.146
1	4	0	0	0	1	0.024	0.161	0.054	0.419	0.000	0.146
4	1	1	4	0	0	0.020	0.621	0.000	0.151	0.000	0.145
0	0	0	0	1	0	0.212	0.000	0.521	0.002	0.000	0.139
1	0	1	0	1	1	0.079	0.000	0.172	0.107	0.371	0.136
0	1	0	0	1	0	0.212	0.000	0.434	0.031	0.000	0.133
4	4	0	0	0	1	0.024	0.075	0.091	0.388	0.000	0.131
4	4	0	0	1	1	0.014	0.001	0.591	0.105	0.000	0.127
4	1	1	1	1	0	0.076	0.000	0.379	0.166	0.000	0.123
4	4	0	1	0	0	0.129	0.497	0.015	0.022	0.000	0.122
0	4	1	4	1	0	0.043	0.000	0.001	0.439	0.000	0.121
1	1	1	1	0	1	0.036	0.136	0.001	0.267	0.117	0.118
4	4	1	4	0	0	0.059	0.018	0.004	0.394	0.000	0.117
4	4	0	1	1	0	0.207	0.010	0.224	0.103	0.000	0.116
0	0	1	4	1	1	0.285	0.001	0.002	0.166	0.000	0.113
4	1	1	0	1	1	0.013	0.002	0.316	0.222	0.000	0.111
4	1	0	1	1	1	0.069	0.022	0.024	0.319	0.000	0.105
0	4	0	4	1	1	0.029	0.000	0.000	0.384	0.000	0.103
0	4	0	1	1	1	0.072	0.003	0.039	0.313	0.000	0.103
1	4	1	1	0	0	0.012	0.471	0.015	0.069	0.000	0.100
1	0	1	0	0	1	0.046	0.002	0.047	0.044	0.406	0.097
0	1	1	4	1	0	0.139	0.000	0.010	0.240	0.000	0.097
1	4	0	1	0	1	0.020	0.187	0.001	0.215	0.000	0.090
1	4	1	4	1	0	0.042	0.000	0.002	0.304	0.000	0.087
1	4	1	1	1	1	0.007	0.001	0.179	0.221	0.000	0.087
4	1	1	0	0	1	0.023	0.042	0.001	0.289	0.000	0.085
1	4	1	0	0	1	0.003	0.021	0.009	0.312	0.000	0.084
0	4	1	4	1	1	0.008	0.000	0.000	0.325	0.000	0.083
4	0	1	0	1	1	0.070	0.000	0.285	0.065	0.000	0.081
1	4	1	0	0	0	0.036	0.111	0.071	0.163	0.000	0.080
4	0	1	1	1	1	0.101	0.001	0.271	0.037	0.000	0.079
1	4	0	4	1	0	0.187	0.000	0.000	0.121	0.000	0.077
4	0	0	1	1	1	0.150	0.013	0.066	0.106	0.000	0.077
1	1	0	0	0	1	0.135	0.024	0.069	0.000	0.160	0.076
0	0	0	4	1	1	0.182	0.002	0.032	0.086	0.000	0.072
1	0	1	1	1	1	0.135	0.000	0.047	0.107	0.000	0.068
1	1	1	1	1	1	0.006	0.000	0.005	0.032	0.351	0.068
4	1	1	1	1	1	0.024	0.002	0.104	0.172	0.000	0.067
4	4	0	0	1	0	0.022	0.030	0.296	0.018	0.000	0.064
1	1	1	0	1	1	0.006	0.000	0.076	0.056	0.193	0.060
4	0	1	0	0	1	0.086	0.071	0.000	0.103	0.000	0.059
1	0	1	0	0	0	0.067	0.007	0.027	0.029	0.171	0.058
4	1	1	0	0	0	0.035	0.041	0.030	0.149	0.000	0.058
1	4	0	1	1	1	0.015	0.007	0.058	0.167	0.000	0.056
4	0	0	1	1	0	0.000	0.012	0.000	0.207	0.000	0.054

Appendix F

ASSESSMENT OF BARCODES IN 12-COLOR LIBRARIES

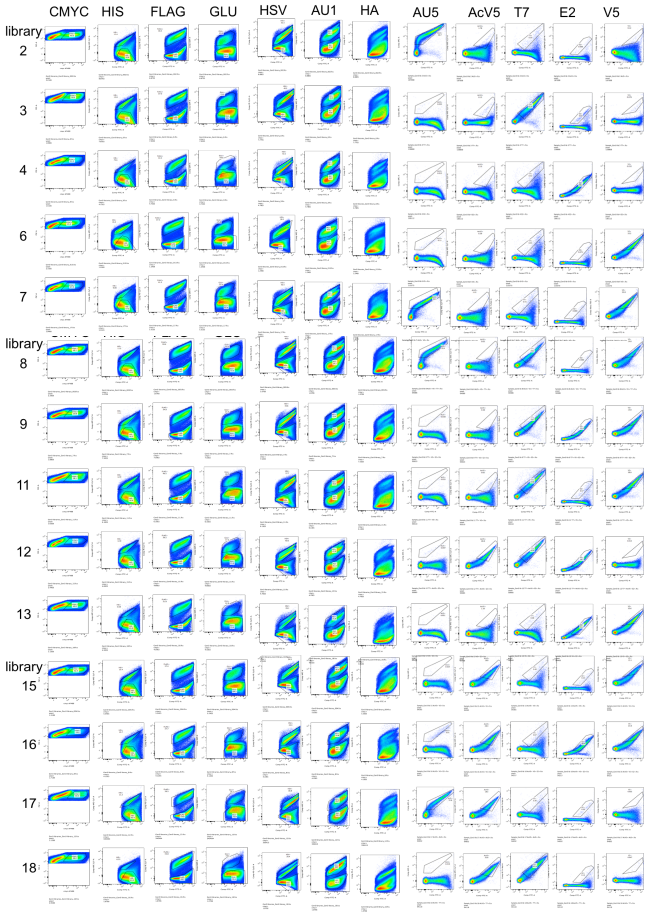


Figure F.1: Flow cytometry analysis of 11-epitope tag barcode libraries. Flow cytometry analysis shows 14 out of 18 libraries had a significant, 25-50%, of cells expressing barcodes, and that all cells expressing barcodes contained the expected combination of T7, V5, AcV5, AU5, and E2 epitope tags. Moreover, 13 out of 14 libraries contained 85-90% new barcode combinations of up to 11 epitope tags.

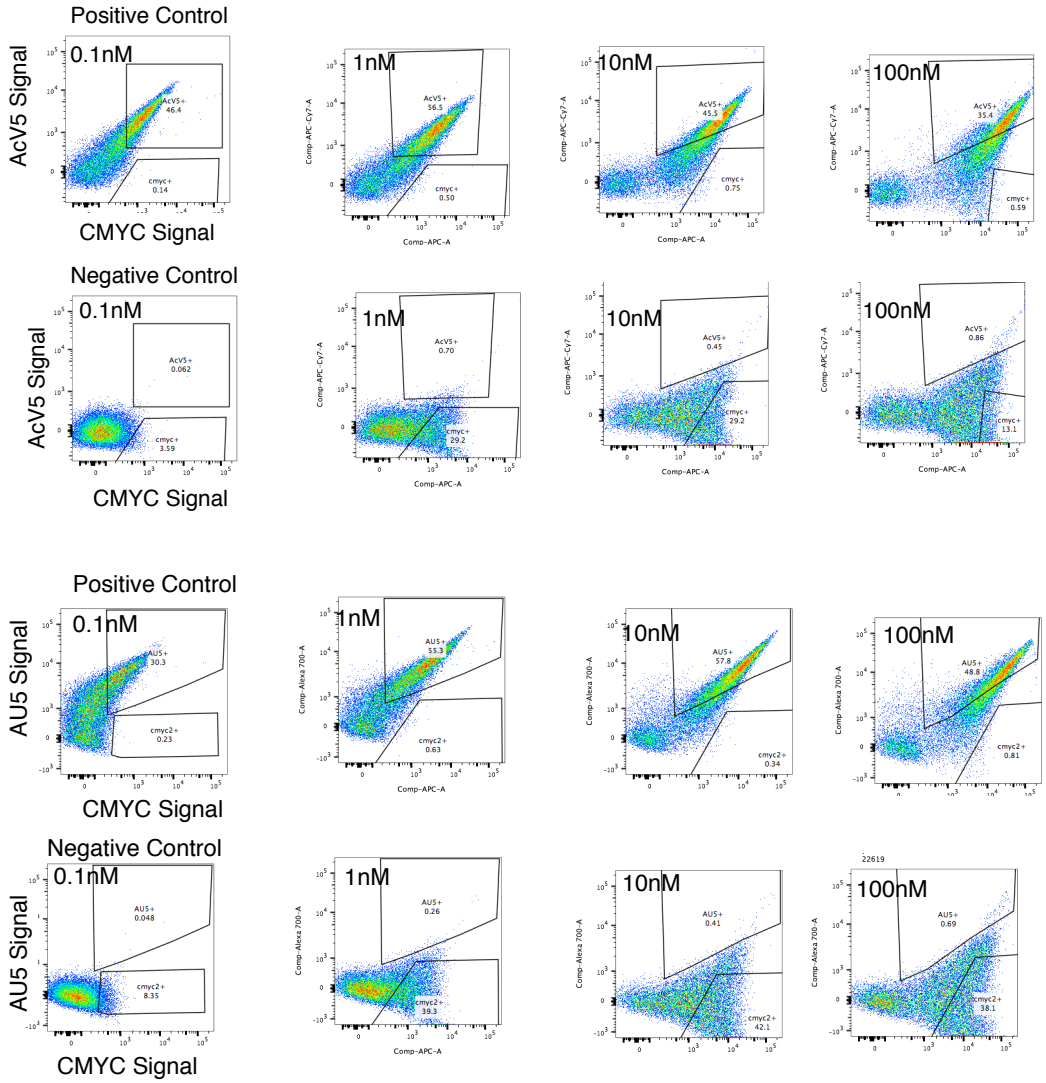


Figure F.2: CMYC AF647 titration. Control barcodes containing AcV5, AU5, and CMYC barcodes were titrated with CMYC antibody and labeled with either 100nM AcV5 APC-CY7 or 100nM AU5 AF700 and 35 nM α-chicken AF647 antibody. It was found that 1-10 nM CMYC antibody was optimal for capturing the highest amount of barcodes with the lowest percentage of false positives.

Table F.1: Antibody concentrations and epitope tag fluorophore pairs used for flow cytometry analysis.

Epitope	Fluorophore	Ab (nM)
AcV5	APC-Cy7	100
AU5	AF700	100
AU1	PE-Cy5	100
E2	PE-Cy7	100
FLAG	PerCP	100
HA	PE	10
HSV	PE-Cy5.5	10
T7	PE-TexasRed	100
V5*	QD25	100
GLU	Marina Blue	100
CMYC**	AF647	3

*35nM anti-human biotin and 10nM streptavidin QD525 were also used

** 35nM anti-chicken AF647 was also used

Appendix G

BARCODE IDENTIFICATION AND QUANTIFICATION SOFTWARE PYTHON SCRIPT

```
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
import time
import csv
import json
from collections import defaultdict
from scipy.stats import gaussian_kde
###############################################################################
#data structure containing rows, points, and label data.
class RPLdata:
    rows = []
    points = []
    labels = []
    rpl = []

    def __init__(self,r=None,p=None,l=None,rpl=None):
        self.rp_by_label_dict = defaultdict(list)

        if rpl == None:
            self.rows = r
            self.points = p
            self.labels = l
            self.rpl = map(lambda x,y,z: [x,y,z], r,p,l)
            self.load_rp_by_label_dict()
        elif rpl != None:
            self.update_rpl(rpl)
            #self.load_rp_by_label_dict()


    #refresh the datastructure with new RPL values
    def update_rpl(self,new_rpl):
        self.rpl = new_rpl
        self.split_rpl()
        self.load_rp_by_label_dict()


    #updates r,p,l  lists from rpl
    def split_rpl(self):
        self.rows = []
        self.points = []
        self.labels = []
        for r,p,l in self.rpl:
```

```
            self.rows.append(r)
            self.points.append(p)
            self.labels.append(l)


    #loads a dict with labels as keys and list of row/point#  pairs as values
    def load_rp_by_label_dict(self):
        for r,p,l in self.rpl:
            self.rp_by_label_dict[l].append([r,p])
       # print "RPL dict has this many keys, should match
#clusters:",len(self.rp_by_label_dict.keys())
        for k,v in self.rp_by_label_dict.items():
        #    print "Keys",k, "Length:", len(v)
            pass

    #take in new list for the given label. Update RPL, but changes order!
    def update_rpl_by_label(self,label,list):
            self.rp_by_label_dict[label] = list
            self.make_rpl_from_label_dict()

    #add a new cluster to rpl, replace old cluster label 1
    #middle cluster label becomes 1 and top cluster becomes 2
    def split_rp_dict_label(self,old_label,rpl):
        print 'old label to be removed is:',old_label
        #print 'B4 pop'
        for k in self.rp_by_label_dict.keys():
            pass
            #print 'key',k
            #print 'value',len(self.rp_by_label_dict[k])
        #print 'After pop'
        self.rp_by_label_dict.pop(old_label)
        for k in self.rp_by_label_dict.keys():
            pass
            #print 'key',k
            #print 'value',len(self.rp_by_label_dict[k])
        max = 0
        max_label = ''
        for r,p,l in rpl:
            if l != -1:
                if p[1] > max:
                    max = p[1]
                    max_label = l
        #print "max_label is:", max_label
        label_list = []
        for r,p,l, in rpl:
            if l != -1:
                label_list.append(l)
        number_unique_labels = len(set(label_list))
        print 'number of unique dbscan labels is:',number_unique_labels

        if number_unique_labels == 1:
            for r,p,l in rpl:
                if l == max_label:
                    self.rp_by_label_dict[1].append([r,p])
        elif number_unique_labels == 2:
            for r,p,l in rpl:
                if l == max_label:
```

```
                    self.rp_by_label_dict[2].append([r,p])
                elif l != -1:
                    self.rp_by_label_dict[1].append([r,p])
        elif number_unique_labels > 2:
            min_length = 1000000
            count_dict = defaultdict(int)
            for r,p,l in rpl:
                if l != -1:
                    count_dict[l]+=1
            for l in count_dict.keys():
                if count_dict[l] < min_length:
                    min_length = count_dict[l]
                    min_count_label = l
            print 'minimum count label is:',min_count_label
            max = 0
            for r,p,l in rpl:
                if l not in [-1,min_count_label]:
                    if p[1] > max:
                        max = p[1]
                        max_label = l
            print 'max label is:',max_label
            for r,p,l in rpl:
                if l == max_label:
                    self.rp_by_label_dict[2].append([r,p])
                elif l not in [min_count_label,-1]:
                    self.rp_by_label_dict[1].append([r,p])

        self.make_rpl_from_label_dict()
        for k,v in self.rp_by_label_dict.items():
            print "Keys",k, "Length:", len(v)


    #ends up sorting by label, but not main goal
    def make_rpl_from_label_dict(self):
        self.rpl = []
        for label,rp_list in self.rp_by_label_dict.items():
            #each dict value is an rp_list pair. Need to make an rpl triple
            new_l = map(lambda x: [x[0],x[1],label] ,rp_list)
            self.rpl += new_l
            #print "Len of rpl is now:", len(self.rpl), len(v)
        self.split_rpl()

    #returns list of points with given label
    def get_rp_for_label(self,label):
        return self.rp_by_label_dict[label]

##############################################################################
##########
#input data from flowjo columns fluorophores rows fluorescence values for
each cell
#output linear, log, and normalized log transformed data with negative values
set to value near zero cluster
def transposedata(filename,tagspresent_dict,tfval_dict,start):

    d = np.genfromtxt(filename,delimiter=',',dtype='float',skip_header=1)

    data = np.zeros_like(d) #store data
```

```
    #transform data if negative to value near zero, also take log10 of
transformed data
    for fluor in tagspresent_dict.keys():
        print 'fluorophore:',fluor
        col = tagspresent_dict[fluor][2] #column number that fluor data is
stored
        print 'column:',col
        eptag = tagspresent_dict[fluor][0] #epitope tag name string
        print 'epitope tag:',eptag
        tagpresent = tagspresent_dict[fluor][1] #is tag present in data set
        print 'tag present?',tagpresent

        for row in range(0,len(d)):
            if eptag in ['FSC','GFP']:
                #set negative values = 1 for GFP or FSC channels
                dpoint = d[row,col]
                if dpoint > 0:
                    data[row,col] = dpoint
                elif dpoint <= 0:
                    data[row,col] = 1
            else:
                tfval = tfval_dict[fluor]
                #print 'transformation value:',tfval
                dpoint = d[row,col]
                if dpoint <= 0: #if negative value, set to value that is in 0
cluster
                    dpoint_tf = tfval
                elif dpoint > 0: #values larger than 0
                    dpoint_tf = dpoint
                data[row,col] = dpoint_tf
    log_data = np.log10(data)
    np.savetxt('log transposed data.csv',log_data,fmt='%f',delimiter=',')
    #np.savetxt('linear transposed data.csv',data,fmt='%f',delimiter=',')


    #find cmyc column
    for fluor in tagspresent_dict.keys():
        eptag = tagspresent_dict[fluor][0]
        col = tagspresent_dict[fluor][2]
        if eptag == 'CMYC':
            cmyccol = col
    print 'cmyc is in column:',cmyccol

    #normalize epitope fluorescence values (except cmyc, gfp, fsc, by cmyc
fluor)
    data_norm = np.zeros_like(d)
    for fluor in tagspresent_dict.keys():
        print 'fluorophore:',fluor
        col = tagspresent_dict[fluor][2] #column number that fluor data is
stored
        print 'column:',col
        eptag = tagspresent_dict[fluor][0] #epitope tag name string
        print 'epitope tag:',eptag
        tagpresent = tagspresent_dict[fluor][1] #is tag present in data set
        print 'tag present?',tagpresent
        for row in range(0,len(data_norm)):
```

```
            datapoint = d
            if eptag in ['CMYC','FSC','GFP']: #do not normalize
                data_norm[row,col] = data[row,col]
            else: #normalize by cmyc fluor
                data_norm[row,col] =
float(data[row,col])/float(data[row,cmyccol])
    log_norm_data = np.log10(data_norm)

    #np.savetxt('normalized transposed
data.csv',data_norm,fmt='%f',delimiter=',')
    np.savetxt('normalized log transposed
data.csv',log_norm_data,fmt='%f',delimiter=',')

    print 'finished normalizing data'
    print 'Run time: %s' % str(time.time()-start)
    print '---'
    return data

##############################################################################
##########################################
#input is transformed fluorescence data, performs DBSCAN on subset of data
for each fluorophore
#calculates the correct cluster 0 or 1 by computing the minimum fluor in each
cluster
#output is dictionary maps epitope to RPL (row, point, label)
def
binary_dbscan(norm_data_fname,tagspresent_dict,dbscanparams_dict,binary_dict,
currfluor,plotclusters,file_path_binary,directory):
    start = time.time()

    data =
np.genfromtxt(norm_data_fname,delimiter=',',dtype='float',skip_header=1)

    #data = data[0:100000,:] #testing using a subset of data
    print 'total number of data points is:', len(data)

    #Run DBSCAN for the fluorophore input called currfluor
    for fluor in tagspresent_dict.keys():
        eptag = tagspresent_dict[fluor][0]
        if eptag == 'CMYC':
            cmyccol = tagspresent_dict[fluor][2]
    print 'cmyc is in column:',cmyccol

    epitope = tagspresent_dict[currfluor][0]
    tagpresent = tagspresent_dict[currfluor][1]
    colnumber = tagspresent_dict[currfluor][2]
    print 'Current fluorophore is:', currfluor
    print 'Epitope is:', epitope
    print 'Is epitope present?', tagpresent
    print 'Data stored in column number:', colnumber

    #use a subset of data bc dbscan is slow
    if len(data) < 100000:
        samplesize = len(data)
    else:
        samplesize = 100000
```

```
    rows_list = range(0,samplesize)

    #Select data subset for DBSCAN 2-D clustering
    db_data = np.zeros((samplesize,2))
    if 'normalized' in norm_data_fname:
        db_data[:,0] = data[0:samplesize,colnumber] #epitope tag, x-axis
        db_data[:,1] = data[0:samplesize,0] #FSC, y-axis
    else:
        db_data[:,0] = data[0:samplesize,cmyccol] #cmyc,x-axis
        db_data[:,1] = data[0:samplesize,colnumber] #epitope tag, y-axis

    if epitope not in ['FSC','GFP','CMYC'] and tagpresent == 'yes':

        #Look up DBSCAN parameters for the current epitope tag
        searchdist = dbscanparams_dict[epitope][0]
        minpoints = dbscanparams_dict[epitope][1]
        print 'DBSCAN max search distance:',searchdist
        print 'DBSCAN min points:',minpoints

        #Run DBSCAN####
        db =
DBSCAN(eps=searchdist,min_samples=minpoints,algorithm="kd_tree").fit(db_data)

        #create list of labels (0, 1, 2, ect.) for data point (row) #if label
== -1 is a noise point
        # number of unique labels == number of clusters
        labels = db.labels_

        # Number of clusters in labels, ignoring noise if present
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        print('Number of DBSCAN clusters: %d' % n_clusters_)

        dbscan_labeled_points = map(lambda x,y:[x,y],labels,db_data)
        #find minimum value in dataset and cooresponding label to find low
cluster
        min = 100000
        min_label = ''
        for l,p in dbscan_labeled_points:
            if l != -1:
                if 'normalized' in norm_data_fname:
                    if p[0] < min:
                        min = p[0]
                        min_label = l
                else:
                    if p[1] < min:
                        min = p[1]
                        min_label = l
        print 'min dbscan cluster id is:',min_label
        if n_clusters_ > 1:
            #find maximum value and corresponding label
            max = 0
            max_label = ''
            for l,p in dbscan_labeled_points:
                if l != -1:
                    if 'normalized' in norm_data_fname:
                        if p[0] > max:
                            max = p[0]
```

```
                        max_label = l
                else:
                    if p[1] > max:
                        max = p[1]
                        max_label = l
            print 'max dbscan cluster id is:',max_label
        #find mid cluster if clusters == 3
        if n_clusters_ == 3:
            unique_labels = set(labels)
            for l in unique_labels:
                if l not in [min_label,max_label,'-1',-1]:
                    mid_label = l
            print 'mid dbscan cluster id is:',mid_label

        #update labels so that low cluster is 0 and high cluster is 1
        updated_labels = []
        for l,p in dbscan_labeled_points:
            #print 'old label',l
            if l != -1:
                if n_clusters_ <= 2:
                    if l == min_label:
                        updated_labels.append(0)
                    else:
                        updated_labels.append(1)
                if n_clusters_ == 3:
                    if l == max_label:
                        updated_labels.append(2)
                    elif l == mid_label:
                        updated_labels.append(1)
                    elif l == min_label:
                        updated_labels.append(0)
            if l == -1:
                updated_labels.append(-1)
                #print 'new label',-1


        updated_dbscan_labeled_points = map(lambda x,y:
[x,y],updated_labels,db_data)

        #make RPL object to store row, point, label for DBSCAN
        rows_list = range(0,samplesize)
        rpl = RPLdata(r=rows_list,p=db_data,l=updated_labels)
        print 'first 10 rows:',rpl.rows[0:10]
        print 'first 10 points:',rpl.points[0:10]
        print 'first 10 labels:',rpl.labels[0:10]

        ###################### Plot results ####################
        if plotclusters == True:
            #create matrix of zeros with same size as labels
            core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
            #set to True if data point belongs to any cluster, set to False
if noise
            core_samples_mask[db.core_sample_indices_] = True

            unique_labels = set(labels)
            colors = ['r','b','g']
            plt.figure()
```

```
            for k, c in zip(unique_labels, colors):
                if k == -1:
                    c = 'k'
                ### creates array of length = number of samples
                ### returns True if data point is in a particular cluster,
else False
                ### the number of class_member_mask arrays created is equal
to the number of clusters found by dbscan +1 (noise)
                class_member_mask = (labels == k)

                ### xy matrix contains the data points within a particular
cluster, as well as their coordinates
                ### # rows = number of samples in cluster, # columns = number
of dimensions
                xy = db_data[class_member_mask & core_samples_mask]

                #plot clustered data
                plt.plot(xy[:,0], xy[:,1], '.', markerfacecolor=c,
                        markeredgecolor='k', markersize=4)

                #plot outliers
                xy = db_data[class_member_mask & ~core_samples_mask]
                plt.plot(xy[:,0], xy[:,1], '.', markerfacecolor=c,
                        markeredgecolor='k', markersize=2)
            if 'normalized' in norm_data_fname:
                plt.ylabel('FSC')
                plt.xlabel(epitope)
                plt.axis([-6,2,3,5])
            else:
                plt.ylabel(epitope)
                plt.xlabel('CMYC')
                plt.axis([3,5.5,0,5.5])
            os.chdir(file_path_binary)
            plt.savefig(epitope +'.png')
            #plt.show()
            os.chdir(directory)

    elif epitope not in ['FSC','GFP','CMYC'] or tagpresent == 'no':
        labels = []
        for i in range(0,samplesize):
            labels.append(0)
        #make RPL object to store row, point, label for DBSCAN
        rows_list = range(0,samplesize)
        rpl = RPLdata(r=rows_list,p=db_data,l=labels)
        print 'first 10 rows:',rpl.rows[0:10]
        print 'first 10 points:',rpl.points[0:10]
        print 'first 10 labels:',rpl.labels[0:10]

    #make rpl.points list instead of numpy array
    xyvals = []
    for x,y in rpl.points:
        xyvals.append([x,y])
    #save off RPL for each binary ID
    binary_dict[epitope] = map(lambda x,y,z:
[x,y,z],rpl.rows,xyvals,rpl.labels)

    with open('binary dbscan clusters data.txt', 'w') as f:
```

```
        f.write(json.dumps(binary_dict,separators=(',',
':'),indent=4,sort_keys=True))

    print 'Run time: %s' % str(time.time()-start)
    print '---'
    print 'finished DBSCAN'
    return binary_dict
################################################################################
########################################
#input:dictionary from dbscan with key epitope, value RPL (row,point,label)
(point is norm epitope fluor,FSC)
#output: dictionary with key epitope to L (label) to value RP
def dbscan_binary_dict_by_label(dbscan_fname):

    DB_bin_dict_by_label = defaultdict(lambda: defaultdict(list))

    with open(dbscan_fname,'r') as f:
        binary_dict = json.load(f)

    for epitope in binary_dict.keys():
        #print 'epitope is:',epitope
        rows = map(lambda x:x[0],binary_dict[epitope])
        points = map(lambda x:x[1],binary_dict[epitope])
        labels = map(lambda x:x[2],binary_dict[epitope])
        RPs = map(lambda x,y:[x,y],rows,points)
        for i in range(0,len(labels)):
            DB_bin_dict_by_label[epitope][labels[i]].append(RPs[i])

    for epitope in DB_bin_dict_by_label.keys():
        #print 'epitope is:',epitope
        for clusterid in DB_bin_dict_by_label[epitope]:
            #print 'cluster id is:',clusterid
            RP_list = DB_bin_dict_by_label[epitope][clusterid]
            #print 'first 3 RPs',RP_list[0:3]

    with open('dbscan binary dict by label.txt','w') as f:
        f.write(json.dumps(DB_bin_dict_by_label,separators=(',',
':'),sort_keys=True))
    print '---'
    print 'finished making binary dict by label'
    return DB_bin_dict_by_label
################################################################################
########################################
#input: dbscan binary dict by label, maps epitope to label (clusterid) to
value RP pairs (point is norm epitope,FSC)
#output: dictionary key epitope to label (clusterid) to value RP pairs (point
is cmyc,epitope)

def
notnorm_binary_dict_by_label(DB_bin_dict_by_label,log_data_fname,tagspresent_
dict):

    data =
np.genfromtxt(log_data_fname,delimiter=',',dtype='float',skip_header=1)

    NN_binary_dict_by_label = defaultdict(lambda:defaultdict(list))
```

```
    #find cmyc column
    for fluor in tagspresent_dict.keys():
        eptag = tagspresent_dict[fluor][0]
        col = tagspresent_dict[fluor][2]
        if eptag == 'CMYC':
            cmyccol = col
    #print 'cmyc is in column:',cmyccol

    #print 'keys',DB_bin_dict_by_label.keys()
    for epitope in DB_bin_dict_by_label:
        #print 'epitope is:',epitope
        for fluor in tagspresent_dict.keys():
            eptag = tagspresent_dict[fluor][0]
            epcol = tagspresent_dict[fluor][2]
            if epitope == eptag:
                currcol = epcol
        #print 'current column:',currcol
        for clusterid in DB_bin_dict_by_label[epitope]:
            #print 'clusterid',clusterid
            RP_list = DB_bin_dict_by_label[epitope][clusterid]
            rows_list = map(lambda x:x[0],RP_list)
            #print 'rows:',rows_list[0:5]
            RP_list_new = []
            for r in rows_list:
                cmycpoint = data[r,cmyccol]
                eppoint = data[r,currcol]
                RP_list_new.append([r,[cmycpoint,eppoint]])
            NN_binary_dict_by_label[epitope][clusterid]=RP_list_new

    for e in NN_binary_dict_by_label.keys():
        print 'epitope is',e
        for cid in NN_binary_dict_by_label[e]:
            print 'cluster id is:',cid
            RPs = NN_binary_dict_by_label[e][cid]
            print 'first 3 RPs is',RPs[0:3]

    with open('not norm binary dict by label.txt','w') as f:
        f.write(json.dumps(NN_binary_dict_by_label,separators=(',',
':'),sort_keys=True))

    print '---'
    print 'finished making binary dict by label'
    return NN_binary_dict_by_label

################################################################################
########################################
#input is dictionary with keys epitope to label (clusterid) to RP pair
(row,point)
#calculates statistics for each epitope, clusterid in binary DBSCAN
#output is stats dict with keys epitope, clusterid,
'mean','min','max','abundance' and corresponding values

def calc_dbscan_stats(dict_by_label,epitope_col,savefile):

    stats_dict = defaultdict(lambda: defaultdict(lambda: defaultdict(list)))

    totalpoints = 0
```

```
    for epitope in dict_by_label.keys():
        for clusterid in dict_by_label[epitope]:
            if clusterid not in [-1,'-1']:
                RPs = dict_by_label[epitope][clusterid]
                points = map(lambda x:x[1],RPs)
                eppoints = map(lambda x:x[epitope_col],points)
                #calc stats
                min_cluster = min(eppoints)
                max_cluster = max(eppoints)
                mean_cluster = np.mean(eppoints)
                sd_cluster = np.std(eppoints)
                #print 'minimum value:',min_cluster
                #print 'maximum value:',max_cluster
                #print 'mean value:',mean_cluster
                #print 'standard deviation:',sd_cluster
                stats_dict[epitope][clusterid]['min'].append(min_cluster)
                stats_dict[epitope][clusterid]['max'].append(max_cluster)
                stats_dict[epitope][clusterid]['mean'].append(mean_cluster)
                stats_dict[epitope][clusterid]['std dev'].append(sd_cluster)
                stats_dict[epitope][clusterid]['number
points'].append(len(points))
                totalpoints+=len(points)

    for epitope in dict_by_label.keys():
        for clusterid in dict_by_label[epitope]:
            RPs = dict_by_label[epitope][clusterid]
            points = map(lambda x:x[1],RPs)
            eppoints = map(lambda x:x[epitope_col],points)
            abundance = float(len(eppoints))/float(totalpoints)*100
            stats_dict[epitope][clusterid]['abundance'].append(abundance)

    with open(savefile, 'w') as f:
        f.write(json.dumps(stats_dict,f,separators=(',',
':'),indent=4,sort_keys=True))

    print 'finished calculating dbscan stats'
    print '---'
    #return stats_dict

##############################################################################
########################################
#inputs are normalized data array, and normalized stats dictionary
#determines cluster id for each data point based on 'min' and 'max' values
calculated from DBSCAN clustering results
#output is dictionary with keys epitope,clusterid, value RP (row, normalized
epitope fluorescence)

def binary_minmax(tagspresent_dict,stats_fname,data_fname):

    binary_minmax_dict = defaultdict(lambda: defaultdict(list))

    with open(stats_fname,'r') as f:
        stats_dict = json.load(f)

    data =
np.genfromtxt(data_fname,delimiter=',',dtype='float',skip_header=1)
```

```
    for epitope in stats_dict.keys():
        #print 'epitope is:',epitope
        for fluor in tagspresent_dict.keys():
            ep = tagspresent_dict[fluor][0]
            col = tagspresent_dict[fluor][2]
            if ep == epitope:
                currcol = col
                #print 'current column is:',currcol
        currdata = data[:,currcol]
        for clusterid in stats_dict[epitope]:
            #print 'cluster id is:',clusterid
            if clusterid not in [-1,'-1']: #do not include the noise points
                minval = stats_dict[epitope][clusterid]['min']
                maxval = stats_dict[epitope][clusterid]['max']
                for r in range(0,len(currdata)):
                    datapoint = currdata[r]
                    if datapoint >= minval and datapoint <= maxval:

binary_minmax_dict[epitope][clusterid].append([r,datapoint])


    #check that the stats are ok
    #for epitope in binary_minmax_dict.keys():
    #    print 'epitope is:',epitope
        for clusterid in binary_minmax_dict[epitope]:
    #        print 'cluster id is:',clusterid
            points = map(lambda
x:x[1],binary_minmax_dict[epitope][clusterid])
    #        print 'mean of cluster',clusterid,'is:',np.mean(points)
    #        print 'abundance of
cluster',clusterid,'is:',float(len(points))/float(len(data))*100

    with open('all binary cluster data.txt', 'w') as f:
        f.write(json.dumps(binary_minmax_dict,f,separators=(',',
':'),sort_keys=True))

    print 'finished clustering binary data'
    print '---'
    return binary_minmax_dict

##############################################################################
##########################################
#input is dictionary with keys epitope, clusterid, value RP pair (rownumber,
normalized epitope fluorescence value)
#output dictionary: maps row number to epitope name to clusterid (dbscan
label)
# deletes any rows that are missing a binary ID

def binaryids_by_rows(minmax_fname):

    with open(minmax_fname,'r') as f:
        binary_minmax_dict = json.load(f)

    binary_dict_by_rows = defaultdict(lambda: defaultdict(str))

    numeps = len(binary_minmax_dict.keys())
    print 'the number of epitope tags is:',numeps
```

```
    row_delete_counter = 0
    for epitope in binary_minmax_dict.keys():
        for clusterid in binary_minmax_dict[epitope]:
            rp = binary_minmax_dict[epitope][clusterid]
            row_list = map(lambda x:x[0],rp)
            points_list = map(lambda x:x[1],rp)
            for r in row_list:
                binary_dict_by_rows[r][epitope] = clusterid
    #delete any rows that don't have all of the epitopes mapped to a cluster
id
    for row in binary_dict_by_rows.keys():
        epitopes = binary_dict_by_rows[row]
        if len(epitopes) != numeps:
            del binary_dict_by_rows[row]
            #print 'deleted row'
            row_delete_counter+=1

    print "This is how many rows we deleted:", row_delete_counter
    with open('binary dict by rows.txt', 'w') as f:
        f.write(json.dumps(binary_dict_by_rows,separators=(',',
':'),indent=4,sort_keys=True))

    return binary_dict_by_rows

################################################################################
##########################################
#input: A dictionary of row numbers to epitope tags to label (dbscan
clusterid)
#output: A dictionary of row numbers to cluster binary id value
(Concatenating the epitope clusterids in order)

def make_row_binaryid_dict(bin_ids_dict,binary_signature_order):
    row_binaryid_dict ={}
    for row in bin_ids_dict:
        signature = ""
        #Use epitopes in signature order input list to order the signature.
        for epitope in binary_signature_order:
            # print "current index:", bin_ids_dict[row][epitope]
            clusterid = bin_ids_dict[row][epitope]
            signature += (str(clusterid))
        row_binaryid_dict[row] = signature
       # print "signature for row",row,"is:", signature

    return row_binaryid_dict
################################################################################
##########################################
#input: A dictionary of rownumbers to a binary cluster id value
#output: A dictionary of binary tag ids to corresponding row numbers

def swap_binaryid_row_dict(input_dict):

    binaryid_row_dict = defaultdict(list)
    #rows are k, clusterids are v
    for k, v in input_dict.items():
        #clusterids now used for key, append the row value to the list
        binaryid_row_dict[v].append(k)
```

```
    #print "Are these 1?:",set(cluster_rows_dict.keys()) ==
set(input_dict.values())
    print "Number of unique binary cluster IDS:",
len(binaryid_row_dict.keys())

    with open("binary_IDS_dict.txt",'w') as f:
        f.write(json.dumps(binaryid_row_dict,f,separators=(',',
':'),indent=4,sort_keys=True))
    return binaryid_row_dict
################################################################################
##########################################
#input binaryid_row_dict with key binaryID map to list of rows, normalized
and non-normalized data
#output: stats dictionary with abundance %, number of points in each binary
id (cluster), and MFIs + SDs for each fluorophore

def
calc_binaryID_stats(tagspresentdict,binID_fname,norm_data_fname,data_fname):

    with open(binID_fname,'r') as f:
        binary_ID_dict = json.load(f)

    norm_data =
np.genfromtxt(norm_data_fname,skip_header=1,delimiter=',',dtype='float')
    data =
np.genfromtxt(data_fname,skip_header=1,delimiter=',',dtype='float')

    binid_stats_dict_normalized = defaultdict(lambda: defaultdict(float))
    binid_stats_dict = defaultdict(lambda: defaultdict(float))

    #find total data that was clustered
    totaldata = len(data)
    total_clustered_data = 0
    for binaryid in binary_ID_dict.keys():
        rowlist = binary_ID_dict[binaryid]
        total_clustered_data+=len(rowlist)

    #calculate statistics for each binaryID
    for binaryid in binary_ID_dict.keys():
        rowlist = binary_ID_dict[binaryid]
        clustered_abundance =
float(len(rowlist))/float(total_clustered_data)*100
        total_abundance = float(len(rowlist))/float(totaldata)*100

        binid_stats_dict[binaryid]['Number points'] = len(rowlist)
        binid_stats_dict[binaryid]['Abundance clustered'] =
clustered_abundance
        binid_stats_dict[binaryid]['Abundance total'] =total_abundance

        binid_stats_dict_normalized[binaryid]['Number points'] = len(rowlist)
        binid_stats_dict_normalized[binaryid]['Abundance clustered'] =
clustered_abundance
        binid_stats_dict_normalized[binaryid]['Abundance total']
=total_abundance

        for fluor in tagspresentdict.keys():
```

```
            epitope = tagspresentdict[fluor][0]
            col = tagspresentdict[fluor][2]
            if epitope not in ['FSC','CMYC','GFP']:
                fluorlist = []
                fluorlistnorm = []
                for r in rowlist:
                    fluorlist.append(data[r,col])
                    fluorlistnorm.append(norm_data[r,col])
                binid_stats_dict[binaryid][epitope+' MFI']=np.mean(fluorlist)
                binid_stats_dict[binaryid][epitope+' SD'] = np.std(fluorlist)
                binid_stats_dict_normalized[binaryid][epitope+'
MFI']=np.mean(fluorlistnorm)
                binid_stats_dict_normalized[binaryid][epitope+' SD'] =
np.std(fluorlistnorm)


    with open("BinaryID stats dict.txt",'w') as f:
        f.write(json.dumps(binid_stats_dict,separators=(',',
':'),indent=4,sort_keys=True))
    with open('BinaryID normalized stats dict.txt','w') as f:
        f.write(json.dumps(binid_stats_dict_normalized,separators=(',',
':'),indent=4,sort_keys=True))


    fields =
['Barcode','T7','V5','AU5','AcV5','E2','HIS','GLU','FLAG','Number
points','Abundance clustered','Abundance total',
             'T7 MFI','T7 SD','V5 MFI','V5 SD','AU5 MFI','AU5 SD','AcV5
MFI','AcV5 SD','E2 MFI','E2 SD','HIS SD'
              ,'GLU MFI','GLU SD','FLAG MFI','FLAG SD']

    barcode_indices = ['T7','V5','AU5','AcV5','E2','HIS','GLU','FLAG']

    for key in binid_stats_dict:
        for i in range(0,len(barcode_indices)):
            binid_stats_dict[key][barcode_indices[i]] = key[i]

    for key in binid_stats_dict_normalized:
        for i in range(0,len(barcode_indices)):
            binid_stats_dict_normalized[key][barcode_indices[i]] = key[i]

    dw = binid_stats_dict
    with open('binary barcode stats.csv','w') as f:
        w = csv.DictWriter(f,fields)
        w.writeheader()
        for k in dw:
            w.writerow({field: dw[k].get(field) or k for field in fields})

    dw = binid_stats_dict_normalized
    with open('binary barcode normalized stats.csv','w') as f:
        w = csv.DictWriter(f,fields)
        w.writeheader()
        for k in dw:
            w.writerow({field: dw[k].get(field) or k for field in fields})

    return binid_stats_dict,binid_stats_dict_normalized
```

```
##############################################################################
##########################################
#input original data file (transposed, not normalized data) and binaryIDs
dict filtered by large clusters only
#output dictionary with keys binaryID to epitope to points (cmyc,epitope
fluor) pairs for all barcode colors
def plot_binary_clusters_data(binaryID_fname,tagspresent_dict,data_fname):

    data =
np.genfromtxt(data_fname,delimiter=',',dtype='float',skip_header=1)

    with open(binaryID_fname,'r') as f:
        binaryID_dict = json.load(f)

    for fluor in tagspresent_dict.keys():
        ep = tagspresent_dict[fluor][0]
        col = tagspresent_dict[fluor][2]
        if ep == 'CMYC':
            cmyccol = col
    #print cmyccol

    plotdict = defaultdict(lambda:defaultdict(list))

    for binID in binaryID_dict.keys():
        #print 'binid id',binID
        rows_list = binaryID_dict[binID]
        for fluor in tagspresent_dict.keys():
            epitope = tagspresent_dict[fluor][0]
            column = tagspresent_dict[fluor][2]
            #print 'epitope is:',epitope
            if epitope not in ['FSC','CMYC','GFP']:
                #print 'banana'
                xypoints = []
                for r in rows_list:
                    #print r
                    cmycpoint = data[r,cmyccol]
                    #print cmycpoint
                    epitopepoint = data[r,column]
                    #print epitopepoint
                    xypoints.append([cmycpoint,epitopepoint])
                #print len(xypoints)
                #print 'apple'
                plotdict[binID][epitope]=xypoints

    with open('plotdict.txt','w') as f:
        f.write(json.dumps(plotdict,separators=(',',
':'),indent=4,sort_keys=True))
    print 'finished finding plotting data'
    return plotdict
##############################################################################
##########################################
def plot_binary_clusters(plotdict,directory,file_path_plots):

    for binID in plotdict.keys():
        fig = plt.figure()
        fig.canvas.set_window_title(binID)
        i = 0
```

```
        j = 0
        for epitope in plotdict[binID]:
            ax = plt.subplot2grid((3,4),(i,j))
            x = map(lambda x:x[0],plotdict[binID][epitope])
            y = map(lambda x:x[1],plotdict[binID][epitope])
            ax.plot(x,y,'.',markerfacecolor = 'c',markersize=2)
            plt.axis([3,5,0,5])
            plt.xlabel('CMYC')
            plt.xticks(fontsize=10)
            plt.yticks(fontsize=10)
            plt.ylabel(epitope)
            plt.tight_layout()
            #print i,j
            if j < 3:
                j+=1
            else:
                j = 0
                i+=1
        #plt.show()
        os.chdir(file_path_plots)
        plt.savefig(binID+' barcode plot.png')
        os.chdir(directory)
    print '---'
    print 'finished making binary plots'

################################################################################
##########################################
#input files containing dictionary of binaryid mapped to rows list,
dictionary of binaryid mapped to stats
#output dictionary of binaryid map to rows list after filtering criteria have
been applied to remove false positive barcodes
def
filter_binary_clusters(binarydict_fname,binary_stats_fname,expected_value):

    filtered_binary_clusters = defaultdict(list)
    filtered_binary_stats = defaultdict(lambda:defaultdict(float))

    with open(binarydict_fname,'r') as f:
        binaryid_dict = json.load(f)

    with open(binary_stats_fname,'r') as f:
        binaryid_stats_dict = json.load(f)

    for binaryid in binaryid_dict.keys():
        print 'binaryid:',binaryid
        cluster_abundance = binaryid_stats_dict[binaryid]['Abundance total']
        num_points = binaryid_stats_dict[binaryid]['Number points']
        #remove low abundance (<0.01% of sample) barcodes
        if cluster_abundance > expected_value and num_points >=100:
            filtered_binary_clusters[binaryid] = binaryid_dict[binaryid]
            filtered_binary_stats[binaryid] = binaryid_stats_dict[binaryid]

    print 'the number of clusters before
filtering:',len(binaryid_dict.keys())
    print 'the number of clusters after
filtering:',len(filtered_binary_clusters.keys())
```

```
    with open('filtered binary barcode dict.txt','w') as f:
        f.write(json.dumps(filtered_binary_clusters,f,separators=(',',
':'),sort_keys=True,indent=4))

    with open('filtered binary barcode stats.txt','w') as f:
        f.write(json.dumps(filtered_binary_stats,f,separators=(',',
':'),sort_keys=True,indent=4))


    fields =
['Barcode','T7','V5','AU5','AcV5','E2','HIS','GLU','FLAG','Number
points','Abundance clustered','Abundance total',
             'T7 MFI','T7 SD','V5 MFI','V5 SD','AU5 MFI','AU5 SD','AcV5
MFI','AcV5 SD','E2 MFI','E2 SD','HIS SD'
              ,'GLU MFI','GLU SD','FLAG MFI','FLAG SD']

    barcode_indices = ['T7','V5','AU5','AcV5','E2','HIS','GLU','FLAG']

    for key in filtered_binary_stats:
        for i in range(0,len(barcode_indices)):
            filtered_binary_stats[key][barcode_indices[i]] = key[i]

    dw = filtered_binary_stats
    with open('filtered binary barcode stats.csv','w') as f:
        w = csv.DictWriter(f,fields)
        w.writeheader()
        for k in dw:
            w.writerow({field: dw[k].get(field) or k for field in fields})

    return filtered_binary_clusters,filtered_binary_stats
##############################################################################
###############################################
#input: dictionary with keys binary id string and value list of row numbers
in that binary cluster (8 tags)
#output: dictionary maps binary id to hsv clusterid to RP list (row,point)
#output2: dictionary with key binaryid map to hsv clusterid to 'HA' or 'AU1'
to RPL list (row, point,label)

def
multicluster(dbscanparams_dict,tagspresent_dict,final_multi_dict,binaryid_fna
me,data_fname,epitope_to_cluster,plotclusters,directory,subdirectory):
    #plotclusters = False ###!!!DO NOT PLOT

    #load original data matrix
    data =
np.genfromtxt(data_fname,delimiter=',',dtype='float',skip_header=1)

    #load dictionary containing key binaryID map to value rows
    with open(binaryid_fname) as f:
        binaryid_row_dict = json.load(f)

    for fluor in tagspresent_dict.keys():
        ep = tagspresent_dict[fluor][0]
        col = tagspresent_dict[fluor][2]
        if ep == 'CMYC':
            cmyccol = col
        if ep == epitope_to_cluster:
```

```
            multiepcol = col
    print 'CMYC data is stored in column:',cmyccol
    print epitope_to_cluster + 'data is stored in column:', multiepcol

    #DBSCAN params
    if epitope_to_cluster == 'HSV':
        #store data after clustering
        multi_dict = defaultdict(lambda: defaultdict(list))
    elif epitope_to_cluster in ['AU1','HA']:
        with open(directory+'/HSV clusters.txt','r') as f:
            multi_dict = json.load(f)

    if epitope_to_cluster == 'HSV':
        for binary_ID in binaryid_row_dict.keys():
            multiepdata = []
            cmycdata = []
            print 'binary cluster is:',binary_ID
            rows_list = binaryid_row_dict[binary_ID]
            #get data from rows in each binary id, for HSV and cmyc fluors
            for r in rows_list:
                multiepdata.append(data[r,multiepcol])
                cmycdata.append(data[r,cmyccol])
            print 'the total number of cells in this cluster
is:',len(multiepdata)

            db_data = np.zeros((len(multiepdata),2))
            for r in range(0,len(multiepdata)):
                db_data[r,0] = cmycdata[r]
                db_data[r,1] = multiepdata[r]

            #override default parameter if data set is small
            num_points = len(db_data)
            print 'number of points is:',num_points
            if num_points < 3000:
                searchdist = 0.2
                corepoint = 30
            elif num_points < 10000:
                searchdist = 0.1
                corepoint = 80
            elif num_points < 15000:
                searchdist = 0.1
                corepoint = 100

            else:
                searchdist = dbscanparams_dict['HSV_bin'][0]
                corepoint = dbscanparams_dict['HSV_bin'][1]
            print 'DBSCAN search distance is:',searchdist
            print 'DBSCAN core point is:',corepoint

            print 'starting dbscan'
            db =
DBSCAN(eps=searchdist,min_samples=corepoint,algorithm="kd_tree").fit(db_data)
            dbscan_labels = db.labels_

            #print "Number of points:",len(db_data)
            #print "Number of rows:", len(rows_list)
            #print "Len labels:", len(dbscan_labels)
```

```
            #print "db first 10:", db_data[0:10]
            #print "rows first 10:", rows_list[0:10]
            #print "labels first 10:",dbscan_labels[0:10]
            dbscan_labeled_points = map(lambda x,y:
[x,y],dbscan_labels,db_data)

            # Number of clusters in labels, ignoring noise if present
            n_clusters_ = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels
else 0)
            print('Number of DBSCAN clusters: %d' % n_clusters_)

            #find minimum value in dataset and cooresponding label to find
low cluster
            min = 100000
            min_label = ''
            for l,p in dbscan_labeled_points:
                if l != -1:
                    if p[1] < min:
                        min = p[1]
                        min_label = l
            print 'min dbscan cluster id is:',min_label

            if n_clusters_ > 1:
                 #find maximum value and corresponding label
                max = 0
                max_label = ''
                for l,p in dbscan_labeled_points:
                    if l != -1:
                        if p[1] > max:
                            max = p[1]
                            max_label = l
                print 'max dbscan cluster id is:',max_label
            #find mid cluster if clusters == 3
            if n_clusters_ == 3:
                unique_labels = set(dbscan_labels)
                for l in unique_labels:
                    if l not in [min_label,max_label,'-1',-1]:
                        mid_label = l
                print 'mid dbscan cluster id is:',mid_label

            #update labels so that clusters are numbered in order of fluor
intensity instead of randomly
            updated_labels = []
            for l,p in dbscan_labeled_points:
                #print 'old label',l
                if l != -1:
                    if n_clusters_ <= 2:
                        if l == min_label:
                            updated_labels.append(0)
                        else:
                            updated_labels.append(1)
                    if n_clusters_ == 3:
                        if l == max_label:
                            updated_labels.append(2)
                        elif l == mid_label:
                            updated_labels.append(1)
                        elif l == min_label:
```

```
                            updated_labels.append(0)
                    if l == -1:
                        updated_labels.append(-1)

            updated_dbscan_labeled_points = map(lambda
x,y:[x,y],db_data,updated_labels)

            rpl = RPLdata(r=rows_list,p=db_data,l=updated_labels)
            print "RPL:", len(rpl.labels),len(rpl.points),len(rpl.rows)

            #Get counts in DBSCAN clusters by label
            counts = defaultdict(int)
            for label in updated_labels:
                counts[label] +=1
            print "Counts histo:", counts

            ##################### Plot results ####################
            if plotclusters == True:
                plt.figure()
                core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
                core_samples_mask[db.core_sample_indices_] = True
                unique_labels = set(dbscan_labels)
                colors = ['r','b','g','m']
                for k, c in zip(unique_labels, colors):
                    if k == -1:
                        c = 'k'
                    class_member_mask = (db.labels_ == k)
                    xy = db_data[class_member_mask & core_samples_mask]
                    #plot clustered data
                    plt.plot(xy[:,0], xy[:,1], '.',
markerfacecolor=c,markeredgecolor='k', markersize=6)
                    #plot outliers
                    xy = db_data[class_member_mask & ~core_samples_mask]
                    plt.plot(xy[:,0], xy[:,1], '.',
markerfacecolor=c,markeredgecolor='k', markersize=2)
                    plt.ylabel(epitope_to_cluster)
                    plt.xlabel('CMYC')
                    plt.title(binary_ID)
                    plt.axis([3,5.5,0,5],fontsize=8)
                #plt.show()
                os.chdir(subdirectory)
                plt.savefig(binary_ID+" "+epitope_to_cluster+'
DBSCAN_r1.png')
                os.chdir(directory)

            #Filtering/Fixing the bad top cluster.
            if n_clusters_ == 2:
                print "Two CLUSTERS, STARTING FILTER"
                rpl =
filter_noise(rpl,binary_ID,epitope_to_cluster,hsvcluster='',label=1,directory
=directory,subdirectory=subdirectory)
            elif n_clusters_ == 1:
                print "ONE cluster, STARTING FILTER"
                rpl =
filter_noise(rpl,binary_ID,epitope_to_cluster,hsvcluster='',label=0,directory
=directory,subdirectory=subdirectory)
            else:
```

```
                print "More than TWO CLUSTERS, NO FILTER"

            #make rpl.points list instead of numpy array
            xyvals = []
            for x,y in rpl.points:
                xyvals.append([x,y])
            #save off RP for each binary ID by label
            for i in range(0,len(rpl.labels)):
                label = rpl.labels[i]
                row = rows_list[i]
                point = xyvals[i]
                multi_dict[binary_ID][label].append([row,point])

        with open(directory+'/HSV clusters.txt', 'w') as f:
            f.write(json.dumps(multi_dict,f,separators=(',',
':'),sort_keys=True,indent=4))

    elif epitope_to_cluster in ['HA','AU1']:
        for binary_ID in multi_dict.keys():
            for hsvcluster in multi_dict[binary_ID]:
                if hsvcluster not in ['-1',-1]:
                    multiepdata = []
                    cmycdata = []
                    print 'binary cluster is:',binary_ID
                    print 'hsv cluster is:',hsvcluster
                    rp_list = multi_dict[binary_ID][hsvcluster]
                    rows_list = map(lambda x:x[0],rp_list)
                    for r in rows_list:
                        multiepdata.append(data[r,multiepcol])
                        cmycdata.append(data[r,cmyccol])
                    print 'the total number of cells in this cluster
is:',len(multiepdata)
                    db_data = np.zeros((len(multiepdata),2))
                    for r in range(0,len(multiepdata)):
                        db_data[r,0] = cmycdata[r]
                        db_data[r,1] = multiepdata[r]

                    #override default dbscan core point parameter if data set
is small
                    num_points = len(rows_list)
                    print 'len rows list',num_points
                    if epitope_to_cluster in ['HA','AU1']:
                        if num_points < 2000:
                            corepoint = 30
                            searchdist = 0.25
                        elif num_points < 3000:
                            corepoint = 30
                            searchdist = 0.2
                        elif num_points < 5000:
                            corepoint = 40
                            searchdist = 0.15
                        elif num_points < 10000:
                            corepoint = 60
                            searchdist = 0.15
                        elif num_points < 20000:
                            corepoint = 80
                            searchdist = 0.1
```

```
                    elif num_points < 40000:
                        corepoint = 100
                        searchdist = 0.1
                    else:
                        corepoint = 100
                        searchdist = 0.1
                if epitope_to_cluster == 'AU1':
                    if binary_ID == '00000000' and num_points > 20000:
                        searchdist = 0.15
                        corepoint = 100


                #elif epitope_to_cluster == 'HA' and num_points > 100:
                #    searchdist = dbscanparams_dict['HA_bin'][0]
                #    corepoint = dbscanparams_dict['HA_bin'][1]
                #elif epitope_to_cluster == 'AU1' and num_points > 100:
                #    searchdist = dbscanparams_dict['AU1_bin'][0]
                #    corepoint = dbscanparams_dict['AU1_bin'][1]
                print 'dbscan corepoint:',corepoint
                print 'dbscan search distance',searchdist

                print 'starting dbscan'
                db =
DBSCAN(eps=searchdist,min_samples=corepoint,algorithm="kd_tree").fit(db_data)

                #create list of labels (0, 1, 2, ect.) for data point
(row)m if label == -1 is a noise point
                # number of unique labels == number of clusters
                dbscan_labels = db.labels_

                #print "Number of points:",len(db_data)
                #print "Number of rows:", len(rows_list)
                #print "Len labels:", len(dbscan_labels)
                #print "db first 10:", db_data[0:10]
                #print "rows first 10:", rows_list[0:10]
                #print "labels first 10:",dbscan_labels[0:10]
                dbscan_labeled_points = map(lambda x,y:
[x,y],dbscan_labels,db_data)

                # Number of clusters in labels, ignoring noise if present
                n_clusters_ = len(set(dbscan_labels)) - (1 if -1 in
dbscan_labels else 0)
                print('Number of DBSCAN clusters: %d' % n_clusters_)

                #find minimum value in dataset and cooresponding label to
find low cluster
                min = 100000
                min_label = ''
                for l,p in dbscan_labeled_points:
                    if l != -1:
                        if p[1] < min:
                            min = p[1]
                            min_label = l
                print 'min dbscan cluster id is:',min_label

                if n_clusters_ > 1:
                     #find maximum value and corresponding label
```

```
                        max = 0
                        max_label = ''
                        for l,p in dbscan_labeled_points:
                            if l != -1:
                                if p[1] > max:
                                    max = p[1]
                                    max_label = l
                        print 'max dbscan cluster id is:',max_label
                    #find mid cluster if clusters == 3
                    if n_clusters_ == 3:
                        unique_labels = set(dbscan_labels)
                        for l in unique_labels:

                            if l not in [min_label,max_label,'-1',-1]:
                                mid_label = l
                        print 'mid dbscan cluster id is:',mid_label

                    #update labels so that clusters are numbered in order of
fluor intensity instead of randomly
                    updated_labels = []
                    for l,p in dbscan_labeled_points:
                        #print 'old label',l
                        if l != -1:
                            if n_clusters_ <= 2:
                                if l == min_label:
                                    updated_labels.append(0)
                                else:
                                    updated_labels.append(1)
                            if n_clusters_ == 3:
                                if l == max_label:
                                    updated_labels.append(2)
                                elif l == mid_label:
                                    updated_labels.append(1)
                                elif l == min_label:
                                    updated_labels.append(0)
                        if l == -1:
                            updated_labels.append(-1)

                    updated_dbscan_labeled_points = map(lambda
x,y:[x,y],db_data,updated_labels)

                    rpl = RPLdata(r=rows_list,p=db_data,l=updated_labels)
                    print "RPL:",
len(rpl.labels),len(rpl.points),len(rpl.rows)

                    #Get counts in DBSCAN clusters by label
                    counts = defaultdict(int)
                    for label in updated_labels:
                        counts[label] +=1
                    print "Counts histo:", counts

                    ################### Plot results ###################
                    if plotclusters == True:
                        plt.figure()
                        core_samples_mask = np.zeros_like(db.labels_,
dtype=bool)
                        core_samples_mask[db.core_sample_indices_] = True
```

```
                        unique_labels = set(dbscan_labels)
                        colors = ['r','b','g','m']
                        for k, c in zip(unique_labels, colors):
                            if k == -1:
                                c = 'k'
                            class_member_mask = (db.labels_ == k)
                            xy = db_data[class_member_mask &
core_samples_mask]
                            #plot clustered data
                            plt.plot(xy[:,0], xy[:,1], '.',
markerfacecolor=c,markeredgecolor='k', markersize=6)
                            #plot outliers
                            xy = db_data[class_member_mask &
~core_samples_mask]
                            plt.plot(xy[:,0], xy[:,1], '.',
markerfacecolor=c,markeredgecolor='k', markersize=2)
                            plt.ylabel(epitope_to_cluster)
                            plt.xlabel('CMYC')
                            plt.title(binary_ID)
                            plt.axis([3,5.5,0,5],fontsize=8)
                        #plt.show()
                        os.chdir(subdirectory)
                        plt.savefig(binary_ID+" "+epitope_to_cluster+'
'+hsvcluster+' DBSCAN_r1.png')
                        os.chdir(directory)

                    #Filtering/Fixing the bad top cluster.
                    if n_clusters_ == 2:
                        print "TWO CLUSTERS, STARTING FILTER"
                        rpl =
filter_noise(rpl,binary_ID,epitope_to_cluster,hsvcluster,label=1,directory=di
rectory,subdirectory=subdirectory)
                    elif n_clusters_ == 1:
                        print 'one cluster starting filter'
                        rpl =
filter_noise(rpl,binary_ID,epitope_to_cluster,hsvcluster,label=0,directory=di
rectory,subdirectory=subdirectory)
                    else:
                        print "NOT 1 or 2 CLUSTERS, NO FILTER"

                    xyvals = []
                    for x,y in rpl.points:
                        xyvals.append([x,y])
                    #save off RPL for each binary ID
                    rpl_list = []
                    for i in range(0,len(rpl.rows)):

rpl_list.append([rpl.rows[i],xyvals[i],rpl.labels[i]])

final_multi_dict[binary_ID][hsvcluster][epitope_to_cluster] = rpl_list

        with open(directory+'/final barcode dict '+epitope_to_cluster+'.txt',
'w') as f:
            f.write(json.dumps(final_multi_dict,f,separators=(',', ':')))
#,sort_keys=True,indent=4))

    return multi_dict, final_multi_dict
```

```
##############################################################################
##########################################
#takes in the an RPLdata by hsvcluster and label, removes noise using
gaussian_kde, returns data.
def
filter_noise(rpl,binary_ID,epitope_to_cluster,hsvcluster,label,directory,subd
irectory):

    #gets a list of (row,point) for the given label
    data = rpl.get_rp_for_label(label)
    #if label == '1stpass':
    #    data = []
    #    for i in range(0,len(rpl.rows)):
    #        data.append([rpl.rows[i],rpl.points[i]])
    print "Number of points before filtering:", len(data)
    num_points = len(data)
    #plot data before filtering w kde fit
    x = map(lambda x: x[1][0],data)
    y = map(lambda x: x[1][1],data)
    stack = np.vstack([x,y])
    kde = gaussian_kde(stack)(stack)
    fig1, ax= plt.subplots(2)
    cax = ax[0].scatter(x,y, c=kde, s=10, edgecolor='')
    fig1.colorbar(cax)

    #show filtered
    #pair up the list of rp points with the kde values
    rpl_data_with_kde = map(lambda x,y: [x,y],data,kde)
    #sort list from lowest to highest kde value (lower = noisier/less dense)
    sorted_kde = sorted(kde)
    #cutoff_thresh = 0.25 if (num_points < 50000) else 0.2
    #print "KDE min/max:", min(kde),max(kde)
    #cutoff_value = (max(kde)-min(kde))*cutoff_thresh
    max_kde = max(kde)
    print 'int max_kde:',np.round(max_kde)
    if epitope_to_cluster == 'HSV':
        if label == 0:
            cutoff_frac = 0.2
            cutoff_value = float(max_kde)*cutoff_frac
        if label == 1:
            if np.round(max_kde) <= 2:
                cutoff_frac = 0.3
            elif np.round(max_kde) <= 3 and num_points < 10000:
                cutoff_frac = 0.3
            elif np.round(max_kde) <= 3 and num_points > 10000:
                cutoff_frac = 0.3
            elif num_points < 6000 and np.round(max_kde) <= 3:
                cutoff_frac = 0.2
            #elif np.round(max_kde) <= 4:
            #    cutoff_value = float(max_kde)*0.4
            else:
                cutoff_frac = 0.1
            if num_points > 10000 and binary_ID == '00000000':
                cutoff_frac = 0.5
            cutoff_value = float(max_kde)*cutoff_frac
    if epitope_to_cluster == 'HA':
        print 'label',label
```

```
        if label == 1:
            binidchar = list(binary_ID)
            if binidchar[6] == '1' and num_points > 2000:
                cutoff_frac = 0.6
            elif np.round(max_kde) <= 1:
                cutoff_frac = 0.3
            elif np.round(max_kde) <= 2:
                cutoff_frac = 0.4
            elif np.round(max_kde) <= 4:
                cutoff_frac = 0.4
            else:
                cutoff_frac = 0.2
            cutoff_value = float(max_kde)*cutoff_frac
        elif label == 0:
            if num_points > 5000:
                cutoff_frac = 0.4
                cutoff_value = float(max_kde)*cutoff_frac
            if np.round(max_kde) <= 1:
                cutoff_frac = 0.2
                cutoff_value = float(max_kde)*cutoff_frac
            if binary_ID == '00000110':
                cutoff_frac = 0.5
                cutoff_value = float(max_kde)*cutoff_frac
            else:
                cutoff_frac = 0.2
                cutoff_value = float(max_kde)*cutoff_frac
        if binary_ID == '00000000' and label == 1:
            cutoff_frac = 0.7
            cutoff_value = float(max_kde)*cutoff_frac
        if binary_ID == '00000001' and label == 1:
            cutoff_frac = 0.6
            cutoff_value = float(max_kde)*cutoff_frac
    if epitope_to_cluster == 'AU1':
        if label == 1:
            if num_points > 10000:
                cutoff_frac = 0.3
                cutoff_value = float(max_kde)*cutoff_frac
            elif np.round(max_kde) > 4:
                cutoff_frac = 0.3
            else:
                cutoff_frac = 0.15
            if binary_ID == '00000000' and num_points > 5000:
                cutoff_frac = 0.5
            cutoff_value = float(max_kde)*cutoff_frac
        if label == 0:
            cutoff_frac = 0.15
            cutoff_value = float(max_kde)*cutoff_frac

    print 'max_kde is:',max_kde
    print 'cutoff_fraction is',cutoff_frac

    filtered_data_with_kde = filter(lambda x: x[1] > cutoff_value,
rpl_data_with_kde) #throw away noisy points (low kde)
    filtered_rpl_data = map(lambda x: x[0], filtered_data_with_kde) #grab
only rpl, no kde
    #print "filtered data first 10:", filtered_rpl_data[0:10]
    filtered_data = map(lambda x: x[1], filtered_rpl_data) #grab only
```

```
coordinates, no rpl no kde
    #print "Lengths:", "Pre filter:",len(rpl_data_with_kde),"After
filter:",len(filtered_data_with_kde), "Diff:", len(filtered_data_with_kde)-
len(rpl_data_with_kde)
    #print 'filtered data fluor values only:',filtered_data[0:10]
    old_rpl_len = len(rpl.rpl)
    #print "Old RPL length:", len(rpl.rpl),
    rpl.update_rpl_by_label(label,filtered_rpl_data)
    #print "New RPL length:", len(rpl.rpl)
    #print "change in rpl length:", len(rpl.rpl)- old_rpl_len
    print 'number of deleted points is:',len(data)-len(filtered_rpl_data)
    #filtered_kde = map(lambda x: x[1], filtered_data_with_kde) #grab only
kde values

    #plot after filtering
    x = map(lambda x: x[0],filtered_data) #grab x-coordinates
    y = map(lambda x: x[1],filtered_data) #grab y-coordinates
    stack = np.vstack([x,y])
    #fig2, ax[1] = plt.subplots(212)
    cax2 = ax[1].scatter(x,y, s=10, edgecolor='')
    #plt.show()
    os.chdir(subdirectory)
    plt.savefig(binary_ID+' '+epitope_to_cluster+' '+hsvcluster+' KDE.png')
    os.chdir(directory)

    #Need to run DBSCAN on filtered_data, load back into RPL
    #hsv_searchdist = dbscanparams_dict['HSV_multi'][0]
    #hsv_corepoint = dbscanparams_dict['HSV_multi'][1]

    #choose parameters based on number of points in top cluster
    if epitope_to_cluster == 'HSV':
        if label == 0:
            hsv_corepoint = 50
            hsv_searchdist = 0.1
        elif label == 1:
            if num_points < 2000:
                hsv_corepoint = 30
                hsv_searchdist = 0.15
            elif num_points < 3000:
                hsv_corepoint = 40
                hsv_searchdist = 0.1
            elif num_points < 5000:
                hsv_corepoint = 60
                hsv_searchdist = 0.1
            elif num_points > 9000:
                hsv_searchdist = 0.07
                hsv_corepoint = 100
            else:
                hsv_corepoint=80
                hsv_searchdist=0.08
    elif epitope_to_cluster == 'HA' or epitope_to_cluster == 'AU1':
        if num_points < 300:
            hsv_corepoint = 30
            hsv_searchdist = 0.2
        elif num_points < 1000:
            hsv_corepoint = 40
            hsv_searchdist = 0.5
```

```
        elif num_points < 2000:
            hsv_corepoint = 40
            hsv_searchdist = 0.1
        elif num_points < 3000:
            hsv_corepoint = 60
            hsv_searchdist = 0.1
        elif num_points < 6000:
            hsv_corepoint = 80
            hsv_searchdist = 0.08
        else:
            hsv_corepoint = 100
            hsv_searchdist = 0.08
        if label == 0:
            if num_points > 10000:
                hsv_searchdist = 0.1
                hsv_corepoint = 100
            elif num_points > 5000:
                hsv_searchdist = 0.1
                hsv_corepoint = 80
            elif num_points < 1000:
                hsv_corepoint = 20
                hsv_searchdist = 0.2
            else:
                hsv_corepoint = 60
                hsv_searchdist = 0.15
    #elif epitope_to_cluster == 'AU1':

    print 'DBSCAN search dist is:',hsv_searchdist
    print 'DBSCAN core points is:',hsv_corepoint

    db =
DBSCAN(eps=hsv_searchdist,min_samples=hsv_corepoint,algorithm="kd_tree").fit(
filtered_data)
    labels = db.labels_
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    print 'Number of dbscan clusters is:', n_clusters_

    # while n_clusters_ > 2:
    #     print 'N clusters greater than 2. starting iterative dbscan'
    #     hsv_searchdist = hsv_searchdist+0.01
    #     db =
DBSCAN(eps=hsv_searchdist,min_samples=hsv_corepoint,algorithm="kd_tree").fit(
filtered_data)
    #     labels = db.labels_
    #     n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    #     print 'Number of dbscan clusters is:', n_clusters_

    counts = defaultdict(int)
    for L in labels:
        counts[L] +=1
    print "Counts histo:", counts


    ##################### Plot results ####################
    #get x and y coordinates
    x = map(lambda x: x[0],filtered_data) #grab x-coordinates
    y = map(lambda x: x[1],filtered_data) #grab y-coordinates
```

```
    plt.figure()
    unique_labels = set(labels)

    #cluster 0 = red, cluster 1 = blue, cluster 2 = yellow
    colors_dict = {0:'r',1:'b',2:'y',-1:'k',3:'m',4:'g',5:'c'}
    #bin data by label for plotting

    dict_by_label = defaultdict(list)
    for index in range(0,len(labels)):
        l = labels[index]
        dict_by_label[l].append([x[index],y[index]])

    for l in unique_labels:
        xl = map(lambda x: x[0],dict_by_label[l])
        yl = map(lambda x: x[1],dict_by_label[l])
        if l == -1:
            plt.plot(xl,yl,'.',markerfacecolor=colors_dict[l],markeredgecolor
= 'k',markersize=2)
        else:
            plt.plot(xl,yl,'.',markerfacecolor=colors_dict[l],markeredgecolor
= 'k',markersize=6)
    plt.ylabel(epitope_to_cluster)
    plt.xlabel('CMYC')
    plt.axis([3,5.5,0,5],fontsize=8)
    #plt.show()
    os.chdir(subdirectory)
    plt.savefig(binary_ID+" "+epitope_to_cluster+' '+hsvcluster+'
DBSCAN_r2.png')
    os.chdir(directory)

    if n_clusters_ > 1 and n_clusters_ < 4:
        #new_db_labels = labels
        #makes list of form [[r1,p1,l1],[r2,p2,l2]]
        new_filtered_rpl_data = map(lambda x,y:
[x[0],x[1],y],filtered_rpl_data,labels)
        #print new_filtered_rpl_data
        #given the old label, we take a new rpl list and make two new labels.
        #give it a list of the form [[r1,p1,l1],[r2,p2,l2]]
        print 'label to remove data from is:',label
        print len(new_filtered_rpl_data)
        rpl.split_rp_dict_label(label,new_filtered_rpl_data)
    else:
        pass
    return rpl
################################################################################
################################
#input binaryid map to hsv label map to 'HA' map to RPL
#input binaryid map to hsv label map to 'AU1' map to RPL
#output 1: dictionary with epitopeID (all 11) to rows list
#output 2: 'barcode abundance' dictionary with epitope ID to abundance,
number of cells, MFI, SD for each fluor
#also makes csv file with epitopeID, abundance normalized by clustered and by
total data
def make_barcode_table(fname,data_fname,tagspresent_dict):

    print "Making barcode table."
```

```
    #load dictionaries containing binaryid map to hsv label map to 'HA' or
'AU1' map to RPL list
    for i in range(0,len(fname)):
        with open(fname[i]) as f:
            if 'HA' in fname[i]:
                HA_dict = json.load(f)
            elif 'AU1' in fname[i]:
                AU1_dict = json.load(f)

    original_data =
np.genfromtxt(data_fname,delimiter=',',dtype='float',skip_header=1)
    total_data = len(original_data)
    total_clustered = 0

    #makes barcode_to_row dict epitope ID (11 tags) map to row list
    #makes row_barcode_dict maps row number to epitope ID
    barcode_abundance_dict = defaultdict(lambda: defaultdict(float))
    print "Banana."
   # print "Ha_dict keys:", HA_dict.keys()
    #count the full (11 color) barcode appearances
    barcode_count_dict = defaultdict(int)
    barcode_to_row_dict =defaultdict(list)
    #dictionary of rows to full (11 color) binary ids
    row_barcode_dict = {}
    print "Len of keys:",len(HA_dict.keys())
    for binary_id in HA_dict.keys():
        print "Bin_id", binary_id
        #print "id:",binary_id
        print "Len of this hsvlabel:",len(HA_dict[binary_id].keys())
        for hsvlabel in HA_dict[binary_id].keys():
            #print "hsvlabel:",hsvlabel
            #print "HSV keys:",HA_dict[binary_id][hsvlabel].keys()
            rpl_list_HA = HA_dict[binary_id][hsvlabel]['HA']
            rpl_list_AU = AU1_dict[binary_id][hsvlabel]['AU1']
            print "length of rpl lists:",len(rpl_list_HA),len(rpl_list_AU)
            rpl_HA = RPLdata(rpl=rpl_list_HA)
            rpl_AU = RPLdata(rpl=rpl_list_AU)
            rpl_HA_row_dict = dict(zip(rpl_HA.rows,rpl_HA.labels))
            rpl_AU_row_dict = dict(zip(rpl_AU.rows,rpl_AU.labels))
            present_rows = set(rpl_HA_row_dict.keys() +
rpl_AU_row_dict.keys())
            #print len(present_rows)
            for row in present_rows:
                if row in rpl_HA_row_dict and row in rpl_AU_row_dict:
                    if rpl_HA_row_dict[row] != -1 and rpl_AU_row_dict[row] !=
-1:
                        #print "making full barcode with:",
binary_id[0:5],rpl_HA_row_dict[row],
hsvlabel,binary_id[5],rpl_AU_row_dict[row],binary_id[6:]
                        full_barcode = str(binary_id[0:5]) +
str(rpl_HA_row_dict[row]) + str(hsvlabel) + str(binary_id[5]) +
str(rpl_AU_row_dict[row]) + str(binary_id[6:])
                        #print "Full barcode is:",full_barcode
                        row_barcode_dict[row] = full_barcode
                        barcode_to_row_dict[full_barcode].append(row)
                        barcode_count_dict[full_barcode] +=1
                        total_clustered +=1
```

```
    #For each barcode id, calculates cluster abundance, number points, mean,
sd fluorescence for each epitope
    for full_barcode in barcode_to_row_dict.keys():
        #print 'full barcode',full_barcode
        num_points = len(barcode_to_row_dict[full_barcode])
        abundance_clustered =
float(barcode_count_dict[full_barcode])/float(total_clustered)*100
        abundance_total =
float(barcode_count_dict[full_barcode])/float(total_data)*100
        barcode_abundance_dict[full_barcode]['Abundance clustered'] =
abundance_clustered
        barcode_abundance_dict[full_barcode]['Abundance total'] =
abundance_total
        barcode_abundance_dict[full_barcode]['Number points'] = num_points
        for fluor in tagspresent_dict.keys():
            #print fluor
            currep = tagspresent_dict[fluor][0]
            currcol = tagspresent_dict[fluor][2]
            if currep not in ['FSC','CMYC','GFP']:
                fluorlist = []
                for row in barcode_to_row_dict[full_barcode]:
                    fluorlist.append(original_data[row,currcol])
                mean_fluor = np.mean(fluorlist)
                sd_fluor = np.std(fluorlist)
                barcode_abundance_dict[full_barcode][currep+' MFI'] =
mean_fluor
                barcode_abundance_dict[full_barcode][currep+' SD'] = sd_fluor
    #print 'keys',barcode_abundance_dict[full_barcode].keys()

    header =
['Barcode','T7','V5','AU5','AcV5','E2','HA','HSV','HIS','AU1','GLU','FLAG','N
umber points','Abundance clustered','Abundance total',
             'T7 MFI','T7 SD','V5 MFI','V5 SD','AU5 MFI','AU5 SD','AcV5
MFI','AcV5 SD','E2 MFI','E2 SD','HA MFI','HA SD','HSV MFI','HSV SD',
             'HIS MFI','HIS SD','AU1 MFI','AU1 SD','GLU MFI','GLU SD','FLAG
MFI','FLAG SD']
    barcode_indices =
['T7','V5','AU5','AcV5','E2','HA','HSV','HIS','AU1','GLU','FLAG']

    for key in barcode_abundance_dict:
        for i in range(0,len(barcode_indices)):
            barcode_abundance_dict[key][barcode_indices[i]] = key[i]
    #print barcode_count_dict
    #print barcode_to_row_dict

    with open('Barcode abundance dict.txt','w') as f:
        f.write(json.dumps(barcode_abundance_dict,separators=(',',
':'),indent=4,sort_keys=True))

    with open ('barcodeID to rows dict.txt','w') as f:
        f.write(json.dumps(barcode_to_row_dict,separators=(',',
':'),indent=4,sort_keys=True))

    dw = barcode_abundance_dict
    with open('Barcode abundances.csv','w') as f:
        w = csv.DictWriter(f,header)
```

```
        w.writeheader()
        for k in dw:
            w.writerow({field: dw[k].get(field) or k for field in header})

    return barcode_to_row_dict,barcode_abundance_dict

################################################################################
##########################################
def filter_clusters_multi(rowsdictfilename,statsdictfilename,expected_value):
#input: dictionary of barcode ids to rows, dictionary of barcode ids to
statistics
#output: dictionary of barcode ids to rows, filtered by expected value
criteria

    filtered_barcodes_dict = defaultdict(list)
    filtered_barcodes_stats = defaultdict(lambda: defaultdict(float))

    with open(rowsdictfilename,'r') as f:
        barcode_dict_rows = json.load(f)

    with open(statsdictfilename,'r') as f:
        stats_dict = json.load(f)

    #append barcode and rows list to new dict if meets abundance cutoff
criteria
    for barcodeid in stats_dict.keys():
        bar_abundance = stats_dict[barcodeid]['Abundance clustered']
        num_points = stats_dict[barcodeid]['Number points']
        rows_list = barcode_dict_rows[barcodeid]
        if bar_abundance > expected_value and num_points >= 100:
            filtered_barcodes_dict[barcodeid] = rows_list
            filtered_barcodes_stats[barcodeid] = stats_dict[barcodeid]

    with open('filtered barcodeID to rows dict.txt','w') as f:
        f.write(json.dumps(filtered_barcodes_dict,separators=(',',
':'),indent=4,sort_keys=True))

    with open('filtered barcodeID stats dict.txt','w') as f:
        f.write(json.dumps(filtered_barcodes_stats,separators=(',',
':'),indent=4,sort_keys=True))

    barcode_indices =
['T7','V5','AU5','AcV5','E2','HA','HSV','HIS','AU1','GLU','FLAG']

    for key in filtered_barcodes_stats:
        for i in range(0,len(barcode_indices)):
            filtered_barcodes_stats[key][barcode_indices[i]] = key[i]

    header =
['Barcode','T7','V5','AU5','AcV5','E2','HA','HSV','HIS','AU1','GLU','FLAG','N
umber points','Abundance clustered','Abundance total',
             'T7 MFI','T7 SD','V5 MFI','V5 SD','AU5 MFI','AU5 SD','AcV5
MFI','AcV5 SD','E2 MFI','E2 SD','HA MFI','HA SD','HSV MFI','HSV SD',
             'HIS MFI','HIS SD','AU1 MFI','AU1 SD','GLU MFI','GLU SD','FLAG
MFI','FLAG SD']

    dw = filtered_barcodes_stats
```

```
    with open('Filtered barcode abundances.csv','w') as f:
        w = csv.DictWriter(f,header)
        w.writeheader()
        for k in dw:
            w.writerow({field: dw[k].get(field) or k for field in header})

    print 'number of barcodes before
filtering:',len(barcode_dict_rows.keys())
    print 'number of barcodes after
filtering:',len(filtered_barcodes_dict.keys())

    return filtered_barcodes_dict,filtered_barcodes_stats
################################################################################
##########################################
#input: A dictionary of binary tag ids to corresponding row numbers list
#output: A dictionary with binary tag ids to corresponding list of GFP fluor
values

def GFP_hist(binID_fname,data_fname,BY_fname,GFP_col,directory):

    data =
np.genfromtxt(data_fname,delimiter=',',dtype='float',skip_header=1)
    bydata =
np.genfromtxt(BY_fname,delimiter=',',dtype='float',skip_header=1)

    with open(binID_fname,'r') as f:
        binary_ID_dict = json.load(f)

    GFP_dict = defaultdict(list)

    BY_list = []
    for entry in bydata[:,1]:
        if entry > 0:
            BY_list.append(np.log10(entry))

    i = 0
    j = 0
    #large_clusters = 0
    number_clusters = len(binary_ID_dict.keys())
    print 'the number of clusters for plotting is:',number_clusters
    for binaryid in binary_ID_dict.keys():
        #print binaryid
        for row in binary_ID_dict[binaryid]:
            GFPval = data[row,GFP_col]
            if GFPval > 0:
                GFP_dict[binaryid].append(GFPval)
        length = len(GFP_dict[binaryid])
        #print 'npoints in cluster is:',length
        #only want to plot large clusters
        #if length > 100:
        #    large_clusters += 1
    #print 'the number of large clusters is',large_clusters
    plotgridsize = (np.int(np.ceil(float(number_clusters)/float(4))),4)

    for binaryid in GFP_dict.keys():
        BYplot = BY_list[0:len(GFP_dict[binaryid])]
        #print "i,j is: ", i,j
```

```
        #fig = plt.figure()
        #plt.subplots(figsize=(20,10))
        ax = plt.subplot2grid(plotgridsize,(i,j))
        ax.hist(GFP_dict[binaryid],bins=100,range=[0,5],fc =
(0,1,0,0.5),histtype='stepfilled')
        ax.hist(BYplot,bins=100,range=[0,5],fc =
(0.5,0.5,0.5,0.3),histtype='stepfilled')
        if i != plotgridsize[0]-1:
            ax.xaxis.set_visible(False)
        plt.title(binaryid,fontsize=12)
        plt.xticks(fontsize=10)
        plt.yticks(fontsize=8)
        plt.tight_layout()

        if j < plotgridsize[1]-1:
            j+=1
        elif j == plotgridsize[1]-1 and i < plotgridsize[0]-1:
            i+=1
            j=0
    plt.savefig(directory+'/GFP histograms.png')
    #plt.show()
    print 'finished plotting GFP histograms'
    return GFP_dict

################################################################################
##########################################
def plot_results(filename):
    with open(filename,'r') as f:
         hsv_dict = json.load(f)

    #HSV dict has key binary id value row, point xy pair, label
    print hsv_dict.keys()
    for binaryID in hsv_dict:
        print 'Binary ID:',binaryID
        xy = map(lambda x: x[1],hsv_dict[binaryID])
        print len(xy)
        labels = map(lambda labels: labels[2],hsv_dict[binaryID])
        unique_labels = set(labels)
        #cluster 0 = red, cluster 1 = blue, cluster 2 = yellow
        colors_dict = {0:'r',1:'b',2:'y',-1:'k'}

        #bin data by labels
        dict_by_label = defaultdict(list)
        for index in range(0,len(labels)):
            l = labels[index]
            xyval = xy[index]
            dict_by_label[l].append([xyval])

        fig,ax = plt.subplots()
        for l in unique_labels:
            x = map(lambda x: x[0][0],dict_by_label[l])
            y = map(lambda x: x[0][1],dict_by_label[l])
            #print l
            #print len(x)
            #plot clusters only
            if l != -1:
```

```
plt.plot(x,y,'.',markerfacecolor=colors_dict[l],markeredgecolor =
'k',markersize=6)
            #plot outliers and clusters
            #if l == -1:
            #
plt.plot(x,y,'.',markerfacecolor=colors_dict[l],markeredgecolor =
'k',markersize=2)
            #else:
            #
plt.plot(x,y,'.',markerfacecolor=colors_dict[l],markeredgecolor =
'k',markersize=6)
        plt.title(binaryID)
        plt.xlabel('CMYC')
        plt.ylabel('HSV')
        plt.axis([2.5,5.5,0,5])
        fig.savefig(binaryID+'.png')
        #plt.show()
##########################################
def main():

    file_path =
"/Users/Stefanie/PycharmProjects/untitled/Gen3_AcV5_T7_library"
    file_path_binary = file_path + '/binary DBSCAN'
    file_path_hsv = file_path+"/HSV"
    file_path_ha = file_path+"/HA"
    file_path_au = file_path+"/AU1"
    file_path_epitopes = file_path+'/eptiope plots'
    for i in
[file_path,file_path_epitopes,file_path_au,file_path_ha,file_path_hsv,file_pa
th_binary]:
        if not os.path.exists(i):
            os.makedirs(i)
    os.chdir(file_path)

    start = time.time()
    filename = '/Users/Stefanie/Desktop/Gen3 libraries clusters
data/Gen3_library_18.csv'
    #BY_fname = '/Users/Stefanie/Desktop/GFP+Bar mixes 4-2-17/Autofluor
GFP.csv'

    #used this for Gen3 libraries BY
    #key is fluorophore, value[0] is epitope, value[1] is present or not?,
value[2] is column data is stored in
    tagspresent_dict = {'FSC':['FSC','yes',0],'AF647':['AU1','yes',1],'APC-
Cy7':['HIS','yes',2],'AF700':['V5','no',3],'Marina Blue':['GLU','yes',4],

'AF488':['CMYC','yes',5],'QDot525':['AU5','no',12],'PE':['HA','yes',6],'PE-
Cy5':['NA','no',7],
                        'PE-Cy5.5':['HSV','yes',8],'PE-
Cy7':['FLAG','yes',9],'PE-TexasRed':['T7','no',10],
                        'PerCP':['E2','no',11],'QDot705':['AcV5','no',13]}

    #use this for BY Gen3 control sample
    #tagspresent_dict = {'FSC':['FSC','yes',0],'APC':['AU1','yes',1],'APC-
Cy7':['AcV5','yes',2],'AF700':['AU5','yes',3],
    #                    'Marina
Blue':['GLU','yes',4],'AF488':['CMYC','yes',5],'QDot525':['V5','no',12],'PE':
```

```
['HA','yes',6],
    #                   'PE-Cy5':['NA','no',7],'PE-
Cy5.5':['HSV','yes',8],'PE-Cy7':['E2','yes',9],'PE-TexasRed':['T7','yes',10],
    #                   'PerCP':['FLAG','yes',11],'QDot705':['HIS','no',13]}

    #dictionary containing value to set negative data points to in linear
space
    tfval_dict = {'AF647':100,'APC-Cy7':100,'AF700':100,'Marina
Blue':200,'QDot525':100,'PE':100,'PE-Cy5':10,'PE-Cy5.5':100,
                  'PE-Cy7':10,'PE-
TexasRed':100,'PerCP':100,'QDot705':10,'AF488':1}

    #name of file containing normalized log transformed data (you want to do
binary dbscan on this)
    norm_data_fname = file_path+'/normalized log transposed data.csv'
    #name of file containing log transformed data
    log_data_fname = file_path+'/log transposed data.csv'
    #name of file with saved binary dbscan dictionary map epitope name to RPL
list
    dbscan_fname = file_path+'/binary dbscan clusters data.txt'
    #name of file with binary dbscan dictionary map epitope name to label to
RP list
    dbscan_by_label_fname = file_path+'/dbscan binary dict by label.txt'
    #name of file with binary dbscan dictionary map epitope name to label to
RP list, where P is (cmyc,epitope fluor)
    notnorm_by_label_fname = file_path+'/not norm binary dict by label.txt'
    #name of dict with normalized binary dbscan stats, map epitope name to
label to stats
    stats_fname = file_path+'/dbscan norm stats.txt'
    #name of dict with non-normalized binary dbscan stats, map epitope name
to label to stats
    stats_notnorm_fname = file_path+'/dbscan not norm stats.txt'
    #name of dict with all binary clusters, maps epitope name to label to RP
list, where P is epitope fluor
    minmax_fname = file_path+'/all binary cluster data.txt'
    #name of dict with binaryIDS mapped to rows in data matrix
    binaryID_fname = file_path+'/binary_IDS_dict.txt'
    #name of file containing dict of binaryID statistics (abundance, number
points, MFIs, SDs)
    binary_stats_fname = file_path+ '/BinaryID stats dict.txt'
    #name of file containing dict of binaryID normalized statistics
(abundance, number points, MFIs, SDs)
    binary_norm_stats_fname = file_path+ '/BinaryID normalized stats
dict.txt'
    #name of file containing dict with filtered binaryIDs map to rows list
    filtered_binary_dict_fname = file_path+'/filtered binary barcode
dict.txt'
    #name of file containing dict with filtered binaryIDs map to stats
    filtered_binary_dict_stats_fname = file_path+'/filtered binary barcode
stats.txt'
    #name of dictionary with binaryid map to hsv label to 'HA' map to list of
RPL pairs
    final_barcode_dict_fname_ha = file_path+'/final barcode dict HA.txt'
    #name of dictionary with binaryid map to hsv label to 'HA' map to list of
RPL pairs
    final_barcode_dict_fname_au1 = file_path+'/final barcode dict AU1.txt'
    #name of dictionary with barcode ID mapped abundance and MFI, SD for
```

```
fluororphores
    final_barcode_abundance_fname = file_path+'/Barcode abundance dict.txt'
    #name of dictionary with barcodeid mapped to rows list
    final_barcode_dict_fname = file_path+'/barcodeID to rows dict.txt'


    #use for BY control sample
    #dbscanparams_dict =
{'HIS':[0.08,50],'GLU':[0.08,50],'FLAG':[0.08,50],'HSV':[0.1,50]
    #
,'HA':[0.1,40],'AU1':[0.1,80],'T7':[0.08,50],'E2':[0.08,50],'V5':[0.08,50]
    #                    ,'AcV5':[0.08,50],'AU5':[0.08,50]}

    #use for Gen3 BY libraries
    dbscanparams_dict =
{'HIS':[0.1,100],'GLU':[0.1,100],'FLAG':[0.1,100],'HSV_bin':[0.15,100]

,'HA_bin':[0.1,40],'AU1_bin':[0.1,80],'T7':[0.08,50],'E2':[0.08,50],'V5':[0.0
8,50]
                        ,'AcV5':[0.08,50],'AU5':[0.08,50]}

    #use for Gen3 control sample
    #binary_signature_order =
['T7','V5','AU5','AcV5','E2','HA','HSV','HIS','AU1','GLU','FLAG']

    #use for Gen3 BY Libs
    binary_signature_order = ['T7','V5','AU5','AcV5','E2','HIS','GLU','FLAG']


    #############RUN FUNCTIONS###################
    #transpose data, normalizes data
    data = transposedata(filename,tagspresent_dict,tfval_dict,start)

    #performs DBSCAN to cluster data subset, maps epitope name to RPL
    binary_dict = defaultdict(list)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                                ,binary_dict=binary_dict,currfluor = 'Marina
Blue',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                                ,binary_dict=binary_dict,currfluor =
'QDot525',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                                ,binary_dict=binary_dict,currfluor =
'QDot705',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                                ,binary_dict=binary_dict,currfluor =
```

```
'PerCP',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    #binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
    #                           ,binary_dict=binary_dict,currfluor =
'PE',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                               ,binary_dict=binary_dict,currfluor = 'PE-
Cy5',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                               ,binary_dict=binary_dict,currfluor = 'PE-
TexasRed',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    #binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
    #                           ,binary_dict=binary_dict,currfluor = 'PE-
Cy5.5',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                               ,binary_dict=binary_dict,currfluor = 'PE-
Cy7',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                               ,binary_dict=binary_dict,currfluor =
'AF700',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
                               ,binary_dict=binary_dict,currfluor = 'APC-
Cy7',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)
    #binary_dict =
binary_dbscan(norm_data_fname=log_data_fname,tagspresent_dict=tagspresent_dic
t,dbscanparams_dict=dbscanparams_dict
    #                           ,binary_dict=binary_dict,currfluor =
'AF647',plotclusters =
True,file_path_binary=file_path_binary,directory=file_path)

    #creates dictionary binning epitope name and label with RP list where
points are (normalized epitope fluor, FSC)
    DB_bin_dict_by_label = dbscan_binary_dict_by_label(dbscan_fname)

    #creates dictionary binning epitope name and label with RP list where
points are (cmyc fluor, epitope fluor)
```

```
    NN_binary_dict_by_label =
notnorm_binary_dict_by_label(DB_bin_dict_by_label,log_data_fname,tagspresent_
dict)

    #calcuates normalized statistics for each binary cluster

calc_dbscan_stats(dict_by_label=DB_bin_dict_by_label,epitope_col=1,savefile='
dbscan norm stats.txt')

    #calcuates non-normalized statistics for each binary cluster

calc_dbscan_stats(dict_by_label=NN_binary_dict_by_label,epitope_col=1,savefil
e='dbscan not norm stats.txt')

    #uses statistics computed from DBSCAN to figure out labels for all
data,maps epitope to label to RP list, where P is epitope fluor
    binary_minmax_dict =
binary_minmax(tagspresent_dict,stats_fname=stats_notnorm_fname,data_fname=log
_data_fname)

    #makes dictionary with row (corresponding to original data matrix) map to
epitope name map to label
    binary_dict_by_rows = binaryids_by_rows(minmax_fname=minmax_fname)

    #makes dictionary with row map to binaryID (concatinated labels for each
epitope)
    row_binaryid_dict =
make_row_binaryid_dict(binary_dict_by_rows,binary_signature_order)

    #makes dictionary with binaryID map to list of rows
    binaryid_row_dict = swap_binaryid_row_dict(row_binaryid_dict)

    #calculates statistics for each binary ID (abundance, number points,
MFIs)
    [binid_stats_dict,binid_stats_dict_norm] =
calc_binaryID_stats(tagspresent_dict,binID_fname=binaryID_fname,norm_data_fna
me=norm_data_fname,data_fname=log_data_fname)

    #creates dictionary with binaryid map to 'epitope' map to list of
points(cmyc,fluor)
    plotdict =
plot_binary_clusters_data(binaryID_fname=binaryID_fname,tagspresent_dict=tags
present_dict,data_fname=log_data_fname)

    #plots data from plotdict epitope vs cmyc for all binaryids

#plot_binary_clusters(plotdict,directory=file_path,file_path_plots=file_path_
epitopes)

    #filters out FP barcodes based on expected value criteria
    [filtered_binary_clusters,filtered_binary_stats] =
filter_binary_clusters(binarydict_fname =
binaryID_fname,binary_stats_fname=binary_stats_fname,expected_value=0.1)

    #plots GFP histogram for each barcode

#GFP_hist(binID_fname=binary_ID_large_fname,data_fname=log_data_fname,BY_fnam
```

```
e=BY_fname,GFP_col=5,directory=file_path)

    #######USE THESE ONLY IF YOU NEED TO USE KDE FILTERING FOR MULTIPLE
INTENSITY EPITOPES##########
    #clusters multi-intensity barcodes using DBSCAN and KDE filtering
    final_multi_dict = defaultdict(lambda: defaultdict(lambda:
defaultdict(list)))
    #[multi_dict,final_multi_dict] =
multicluster(dbscanparams_dict,tagspresent_dict,final_multi_dict,binaryid_fna
me=filtered_binary_dict_fname,
    #
data_fname=log_data_fname,epitope_to_cluster='HSV',plotclusters=True,director
y=file_path,subdirectory=file_path_hsv)

    #[multi_dict,final_multi_dict] =
multicluster(dbscanparams_dict,tagspresent_dict,final_multi_dict,binaryid_fna
me=filtered_binary_dict_fname,
    #
data_fname=log_data_fname,epitope_to_cluster='HA',plotclusters=True,directory
=file_path,subdirectory=file_path_ha)

    [multi_dict,final_multi_dict] =
multicluster(dbscanparams_dict,tagspresent_dict,final_multi_dict,binaryid_fna
me=filtered_binary_dict_fname,

data_fname=log_data_fname,epitope_to_cluster='AU1',plotclusters=True,director
y=file_path,subdirectory=file_path_au)

    #creates dictionary of barcodeid (11 number string) to row list
    [barcode_to_row_dict,barcode_abundance_dict] =
make_barcode_table(fname=[final_barcode_dict_fname_ha,final_barcode_dict_fnam
e_au1],data_fname=log_data_fname,tagspresent_dict=tagspresent_dict)

    #filters barcodes based on expected value criteria and returns new
dictionary, csv file
    [filtered_barcodes_dict,filtered_barcodes_stats] =
filter_clusters_multi(rowsdictfilename=final_barcode_dict_fname,statsdictfile
name=final_barcode_abundance_fname,expected_value=0.01)


    end = time.time()
    print '--- %s seconds ---' % str(end - start)


#call main
if __name__ == "__main__":
    main()
```

Appendix H

SOFTWARE ESTIMATION OF BARCODE IDENTITIES AND ABUNDANCES IN 11-EPITOPE TAG LIBRARIES

Table H.1: Software estimation of barcode identities and abundances in 11-color library

	T7 Library												
Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	36444	13.38
00000000010	0	0	0	0	0	0	0	0	0	1	0	29822	10.95
00000000110	0	0	0	0	0	0	0	0	1	1	0	23510	8.63
00000010101	0	0	0	0	0	0	1	0	1	0	1	16813	6.17
00000000101	0	0	0	0	0	0	0	0	1	0	1	15542	5.71
00000001010	0	0	0	0	0	0	0	1	0	1	0	12717	4.67
00000010001	0	0	0	0	0	0	1	0	0	0	1	12548	4.61
00000000001	0	0	0	0	0	0	0	0	0	0	1	11548	4.24
00000000100	0	0	0	0	0	0	0	0	1	0	0	10946	4.02
00000010000	0	0	0	0	0	0	1	0	0	0	0	10143	3.72
00000000011	0	0	0	0	0	0	0	0	0	1	1	8153	2.99
00000020000	0	0	0	0	0	0	2	0	0	0	0	7679	2.82
00000010011	0	0	0	0	0	0	1	0	0	1	1	5549	2.04
00000200000	0	0	0	0	0	2	0	0	0	0	0	4363	1.60
00000100010	0	0	0	0	0	1	0	0	0	1	0	4081	1.50
00000101010	0	0	0	0	0	1	0	1	0	1	0	3156	1.16
00000010100	0	0	0	0	0	0	1	0	1	0	0	3155	1.16
00000100000	0	0	0	0	0	1	0	0	0	0	0	3103	1.14
00000001101	0	0	0	0	0	0	0	1	1	0	1	2875	1.06
00000020200	0	0	0	0	0	0	2	0	2	0	0	2780	1.02
00000020101	0	0	0	0	0	0	2	0	1	0	1	2649	0.97
00000010010	0	0	0	0	0	0	1	0	0	1	0	2235	0.82
00000001000	0	0	0	0	0	0	0	1	0	0	0	2137	0.78
00000011010	0	0	0	0	0	0	1	1	0	1	0	1981	0.73
00000020001	0	0	0	0	0	0	2	0	0	0	1	1906	0.70
00000010110	0	0	0	0	0	0	1	0	1	1	0	1887	0.69
00000001110	0	0	0	0	0	0	0	1	1	1	0	1616	0.59
00000001100	0	0	0	0	0	0	0	1	1	0	0	1599	0.59
00000110001	0	0	0	0	0	1	1	0	0	0	1	1586	0.58
00000210001	0	0	0	0	0	2	1	0	0	0	1	1426	0.52
00000100001	0	0	0	0	0	1	0	0	0	0	1	1414	0.52
00000200001	0	0	0	0	0	2	0	0	0	0	1	1322	0.49
00000210000	0	0	0	0	0	2	1	0	0	0	0	1278	0.47
00000021010	0	0	0	0	0	0	2	1	0	1	0	1109	0.41
00000020010	0	0	0	0	0	0	2	0	0	1	0	1085	0.40
00000000111	0	0	0	0	0	0	0	0	1	1	1	1084	0.40
00000111011	0	0	0	0	0	1	1	1	0	1	1	1082	0.40
00000020011	0	0	0	0	0	0	2	0	0	1	1	1032	0.38
00000220000	0	0	0	0	0	2	2	0	0	0	0	1013	0.37
00000101011	0	0	0	0	0	1	0	1	0	1	1	1001	0.37

00000011000	0	0	0	0	0	0	1	1	0	0	0	1000	0.37
00000001200	0	0	0	0	0	0	0	1	2	0	0	955	0.35
00000110000	0	0	0	0	0	1	1	0	0	0	0	953	0.35
00000101100	0	0	0	0	0	1	0	1	1	0	0	902	0.33
00000020110	0	0	0	0	0	0	2	0	1	1	0	880	0.32
00000010111	0	0	0	0	0	0	1	0	1	1	1	801	0.29
00000011100	0	0	0	0	0	0	1	1	1	0	0	799	0.29
00000011101	0	0	0	0	0	0	1	1	1	0	1	700	0.26
00000120000	0	0	0	0	0	1	2	0	0	0	0	663	0.24
00000100011	0	0	0	0	0	1	0	0	0	1	1	642	0.24
00000210101	0	0	0	0	0	2	1	0	1	0	1	637	0.23
00000200101	0	0	0	0	0	2	0	0	1	0	1	506	0.19
00000001001	0	0	0	0	0	0	0	1	0	0	1	468	0.17
00000011200	0	0	0	0	0	0	1	1	2	0	0	463	0.17
00000111100	0	0	0	0	0	1	1	1	1	0	0	463	0.17
00000110101	0	0	0	0	0	1	1	0	1	0	1	435	0.16
00000110011	0	0	0	0	0	1	1	0	0	1	1	347	0.13
00000200100	0	0	0	0	0	2	0	0	1	0	0	334	0.12
00000011110	0	0	0	0	0	0	1	1	1	1	0	327	0.12
00000100101	0	0	0	0	0	1	0	0	1	0	1	323	0.12
00000220001	0	0	0	0	0	2	2	0	0	0	1	293	0.11
00000110010	0	0	0	0	0	1	1	0	0	1	0	259	0.10
00000121010	0	0	0	0	0	1	2	1	0	1	0	241	0.09
00000211011	0	0	0	0	0	2	1	1	0	1	1	241	0.09
00000100110	0	0	0	0	0	1	0	0	1	1	0	236	0.09
00000101000	0	0	0	0	0	1	0	1	0	0	0	234	0.09
00000120001	0	0	0	0	0	1	2	0	0	0	1	233	0.09
00000201011	0	0	0	0	0	2	0	1	0	1	1	212	0.08
00000000210	0	0	0	0	0	0	0	0	0	2	1	0	199	0.07
00000101101	0	0	0	0	0	1	0	1	1	0	1	176	0.06
00000220101	0	0	0	0	0	2	2	0	1	0	1	153	0.06
00000020100	0	0	0	0	0	0	2	0	1	0	0	147	0.05
00000021011	0	0	0	0	0	0	2	1	0	1	1	142	0.05
00000111000	0	0	0	0	0	1	1	1	0	0	0	129	0.05
00000020111	0	0	0	0	0	0	2	0	1	1	1	123	0.05
00000011001	0	0	0	0	0	0	1	1	0	0	1	119	0.04
00000210100	0	0	0	0	0	2	1	0	1	0	0	117	0.04
00000111111	0	0	0	0	0	1	1	1	1	1	1	113	0.04
00000101111	0	0	0	0	0	1	0	1	1	1	1	112	0.04

V5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	24567	14.89
00000000010	0	0	0	0	0	0	0	0	0	1	0	22146	13.42
00000000110	0	0	0	0	0	0	0	0	1	1	0	16149	9.78
00000101010	0	0	0	0	0	1	0	1	0	1	0	7723	4.68
00000010001	0	0	0	0	0	0	1	0	0	0	1	7481	4.53
00000010000	0	0	0	0	0	0	1	0	0	0	0	7263	4.40
00000000100	0	0	0	0	0	0	0	0	1	0	0	6587	3.99
00000010101	0	0	0	0	0	0	1	0	1	0	1	6563	3.98
00000010011	0	0	0	0	0	0	1	0	0	1	1	6146	3.72
00000000001	0	0	0	0	0	0	0	0	0	0	1	4204	2.55
00000201010	0	0	0	0	0	2	0	1	0	1	0	4191	2.54
00000200000	0	0	0	0	0	2	0	0	0	0	0	3955	2.40
00000000011	0	0	0	0	0	0	0	0	0	1	1	3745	2.27
00000000101	0	0	0	0	0	0	0	0	1	0	1	3555	2.15
00000100010	0	0	0	0	0	1	0	0	0	1	0	2381	1.44
00000001000	0	0	0	0	0	0	0	1	0	0	0	2304	1.40
00000010010	0	0	0	0	0	0	1	0	0	1	0	2063	1.25
00000010100	0	0	0	0	0	0	1	0	1	0	0	1888	1.14
00000101110	0	0	0	0	0	1	0	1	1	1	0	1836	1.11
00000001100	0	0	0	0	0	0	0	1	1	0	0	1715	1.04
00000011011	0	0	0	0	0	0	1	1	0	1	1	1588	0.96
00000001101	0	0	0	0	0	0	0	1	1	0	1	1555	0.94
00000010110	0	0	0	0	0	0	1	0	1	1	0	1531	0.93
00000100000	0	0	0	0	0	1	0	0	0	0	0	1419	0.86
00000110001	0	0	0	0	0	1	1	0	0	0	1	1344	0.81
00000011000	0	0	0	0	0	0	1	1	0	0	0	1307	0.79
00000210000	0	0	0	0	0	2	1	0	0	0	0	1203	0.73
00000010111	0	0	0	0	0	0	1	0	1	1	1	1113	0.67
00000021010	0	0	0	0	0	0	2	1	0	1	0	1047	0.63
00000011100	0	0	0	0	0	0	1	1	1	0	0	1019	0.62
00000020001	0	0	0	0	0	0	2	0	0	0	1	950	0.58
00000111010	0	0	0	0	0	1	1	1	0	1	0	947	0.57
00000011010	0	0	0	0	0	0	1	1	0	1	0	918	0.56
00000020101	0	0	0	0	0	0	2	0	1	0	1	862	0.52
00000100001	0	0	0	0	0	1	0	0	0	0	1	808	0.49
00000001011	0	0	0	0	0	0	0	1	0	1	1	736	0.45
00000210101	0	0	0	0	0	2	1	0	1	0	1	719	0.44
00000000111	0	0	0	0	0	0	0	0	1	1	1	616	0.37
00000110011	0	0	0	0	0	1	1	0	0	1	1	584	0.35
00000121010	0	0	0	0	0	1	2	1	0	1	0	565	0.34

00000100110	0	0	0	0	0	1	0	0	1	1	0	522	0.32
00000011201	0	0	0	0	0	0	1	1	2	0	1	500	0.30
00000110101	0	0	0	0	0	1	1	0	1	0	1	446	0.27
00000100011	0	0	0	0	0	1	0	0	0	1	1	413	0.25
00000210001	0	0	0	0	0	2	1	0	0	0	1	411	0.25
00000200101	0	0	0	0	0	2	0	0	1	0	1	393	0.24
00000110000	0	0	0	0	0	1	1	0	0	0	0	381	0.23
00000020010	0	0	0	0	0	0	2	0	0	1	0	366	0.22
00000011110	0	0	0	0	0	0	1	1	1	1	0	349	0.21
00000200100	0	0	0	0	0	2	0	0	1	0	0	339	0.21
00000020000	0	0	0	0	0	0	2	0	0	0	0	335	0.20
00000100101	0	0	0	0	0	1	0	0	1	0	1	276	0.17
00000011111	0	0	0	0	0	0	1	1	1	1	1	258	0.16
00000001001	0	0	0	0	0	0	0	1	0	0	1	253	0.15
00000200001	0	0	0	0	0	2	0	0	0	0	1	219	0.13
00000020210	0	0	0	0	0	0	2	0	2	1	0	216	0.13
00000110010	0	0	0	0	0	1	1	0	0	1	0	184	0.11
00000100100	0	0	0	0	0	1	0	0	1	0	0	152	0.09
00000111011	0	0	0	0	0	1	1	1	0	1	1	146	0.09
00000120001	0	0	0	0	0	1	2	0	0	0	1	140	0.08
00000001111	0	0	0	0	0	0	0	1	1	1	1	118	0.07
00000020100	0	0	0	0	0	0	2	0	1	0	0	109	0.07
00000021110	0	0	0	0	0	0	2	1	1	1	0	104	0.06
00000201110	0	0	0	0	0	2	0	1	1	1	0	100	0.06

T7/V5/AU5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundanc
00000000000	0	0	0	0	0	0	0	0	0	0	0	10395	20.15
00000010001	0	0	0	0	0	0	1	0	0	0	1	3771	7.31
00000010000	0	0	0	0	0	0	1	0	0	0	0	3714	7.20
00000000100	0	0	0	0	0	0	0	0	1	0	0	3221	6.24
00000010101	0	0	0	0	0	0	1	0	1	0	1	2740	5.31
00000000001	0	0	0	0	0	0	0	0	0	0	1	2677	5.19
00000000011	0	0	0	0	0	0	0	0	0	1	1	2541	4.93
00000101010	0	0	0	0	0	1	0	1	0	1	0	2204	4.27
00000000101	0	0	0	0	0	0	0	0	1	0	1	1908	3.70
00000010011	0	0	0	0	0	0	1	0	0	1	1	1885	3.65
00000200000	0	0	0	0	0	2	0	0	0	0	0	1645	3.19
00000201010	0	0	0	0	0	2	0	1	0	1	0	1452	2.81
00000020000	0	0	0	0	0	0	2	0	0	0	0	1419	2.75
00000100000	0	0	0	0	0	1	0	0	0	0	0	1279	2.48
00000010100	0	0	0	0	0	0	1	0	1	0	0	1197	2.32
00000001101	0	0	0	0	0	0	0	1	1	0	1	1016	1.97
00000210000	0	0	0	0	0	2	1	0	0	0	0	603	1.17
00000110001	0	0	0	0	0	1	1	0	0	0	1	565	1.10
00000001000	0	0	0	0	0	0	0	1	0	0	0	514	1.00
00000020100	0	0	0	0	0	0	2	0	1	0	0	459	0.89
00000000111	0	0	0	0	0	0	0	0	1	1	1	448	0.87
00000110000	0	0	0	0	0	1	1	0	0	0	0	416	0.81
00000001100	0	0	0	0	0	0	0	1	1	0	0	405	0.79
00000010111	0	0	0	0	0	0	1	0	1	1	1	382	0.74
00000011000	0	0	0	0	0	0	1	1	0	0	0	381	0.74
00000101110	0	0	0	0	0	1	0	1	1	1	0	381	0.74
00000100001	0	0	0	0	0	1	0	0	0	0	1	351	0.68
00000011011	0	0	0	0	0	0	1	1	0	1	1	337	0.65
00000211010	0	0	0	0	0	2	1	1	0	1	0	291	0.56
00000011100	0	0	0	0	0	0	1	1	1	0	0	272	0.53
00000100011	0	0	0	0	0	1	0	0	0	1	1	268	0.52
00000101100	0	0	0	0	0	1	0	1	1	0	0	240	0.47
00000220000	0	0	0	0	0	2	2	0	0	0	0	204	0.40
00000001011	0	0	0	0	0	0	0	1	0	1	1	183	0.35
00000110011	0	0	0	0	0	1	1	0	0	1	1	165	0.32
00000011001	0	0	0	0	0	0	1	1	0	0	1	162	0.31
00000111100	0	0	0	0	0	1	1	1	1	0	0	151	0.29
00000021010	0	0	0	0	0	0	2	1	0	1	0	145	0.28
00000120000	0	0	0	0	0	1	2	0	0	0	0	139	0.27
00000110101	0	0	0	0	0	1	1	0	1	0	1	136	0.26
00000001001	0	0	0	0	0	0	0	1	0	0	1	135	0.26

T7/V5/E2 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	12343	15.25
00000000010	0	0	0	0	0	0	0	0	0	1	0	8290	10.24
00000000110	0	0	0	0	0	0	0	0	1	1	0	5852	7.23
00000010000	0	0	0	0	0	0	1	0	0	0	0	5106	6.31
00000010001	0	0	0	0	0	0	1	0	0	0	1	4639	5.73
00000020000	0	0	0	0	0	0	2	0	0	0	0	4174	5.16
00000000001	0	0	0	0	0	0	0	0	0	0	1	4142	5.12
00000000100	0	0	0	0	0	0	0	0	1	0	0	4084	5.05
00000010101	0	0	0	0	0	0	1	0	1	0	1	3682	4.55
00000000101	0	0	0	0	0	0	0	0	1	0	1	3111	3.84
00000001010	0	0	0	0	0	0	0	1	0	1	0	2170	2.68
00000200000	0	0	0	0	0	2	0	0	0	0	0	1857	2.29
00000010100	0	0	0	0	0	0	1	0	1	0	0	1717	2.12
00000020100	0	0	0	0	0	0	2	0	1	0	0	1551	1.92
00000000011	0	0	0	0	0	0	0	0	0	1	1	1409	1.74
00000010010	0	0	0	0	0	0	1	0	0	1	0	1162	1.44
00000100000	0	0	0	0	0	1	0	0	0	0	0	968	1.20
00000010011	0	0	0	0	0	0	1	0	0	1	1	856	1.06
00000210000	0	0	0	0	0	2	1	0	0	0	0	801	0.99
00000100001	0	0	0	0	0	1	0	0	0	0	1	799	0.99
00000010110	0	0	0	0	0	0	1	0	1	1	0	793	0.98
00000001000	0	0	0	0	0	0	0	1	0	0	0	764	0.94
00000001100	0	0	0	0	0	0	0	1	1	0	0	753	0.93
00000110001	0	0	0	0	0	1	1	0	0	0	1	750	0.93
00000220000	0	0	0	0	0	2	2	0	0	0	0	677	0.84
00000100101	0	0	0	0	0	1	0	0	1	0	1	570	0.70
00000100010	0	0	0	0	0	1	0	0	0	1	0	520	0.64
00000200100	0	0	0	0	0	2	0	0	1	0	0	434	0.54
00000120000	0	0	0	0	0	1	2	0	0	0	0	430	0.53
00000101010	0	0	0	0	0	1	0	1	0	1	0	390	0.48
00000210101	0	0	0	0	0	2	1	0	1	0	1	359	0.44
00000011100	0	0	0	0	0	0	1	1	1	0	0	358	0.44
00000110000	0	0	0	0	0	1	1	0	0	0	0	354	0.44
00000001101	0	0	0	0	0	0	0	1	1	0	1	348	0.43
00000020001	0	0	0	0	0	0	2	0	0	0	1	345	0.43
00000011000	0	0	0	0	0	0	1	1	0	0	0	306	0.38
00000110010	0	0	0	0	0	1	1	0	0	1	0	301	0.37
00000110101	0	0	0	0	0	1	1	0	1	0	1	294	0.36
00000001110	0	0	0	0	0	0	0	1	1	1	0	284	0.35
00000000111	0	0	0	0	0	0	0	0	1	1	1	276	0.34

00000020101	0	0	0	0	0	0	2	0	1	0	1	262	0.32
00000210001	0	0	0	0	0	2	1	0	0	0	1	185	0.23
00000100100	0	0	0	0	0	1	0	0	1	0	0	185	0.23
00000210100	0	0	0	0	0	2	1	0	1	0	0	177	0.22
00000100011	0	0	0	0	0	1	0	0	0	1	1	165	0.20
00000100110	0	0	0	0	0	1	0	0	1	1	0	151	0.19
00000011010	0	0	0	0	0	0	1	1	0	1	0	150	0.19
00000101100	0	0	0	0	0	1	0	1	1	0	0	148	0.18
00000220100	0	0	0	0	0	2	2	0	1	0	0	135	0.17
00000010111	0	0	0	0	0	0	1	0	1	1	1	133	0.16
00000001001	0	0	0	0	0	0	0	1	0	0	1	125	0.15
00000110110	0	0	0	0	0	1	1	0	1	1	0	111	0.14
00000200010	0	0	0	0	0	2	0	0	0	1	0	104	0.13

E2 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000110	0	0	0	0	0	0	0	0	1	1	0	16049	11.98
00000000000	0	0	0	0	0	0	0	0	0	0	0	15474	11.55
00000000010	0	0	0	0	0	0	0	0	0	1	0	14484	10.81
00000101010	0	0	0	0	0	1	0	1	0	1	0	7616	5.69
00000000200	0	0	0	0	0	0	0	0	2	0	0	6458	4.82
00000010101	0	0	0	0	0	0	1	0	1	0	1	5335	3.98
00000000101	0	0	0	0	0	0	0	0	1	0	1	5298	3.96
00000201010	0	0	0	0	0	2	0	1	0	1	0	4824	3.60
00000000011	0	0	0	0	0	0	0	0	0	1	1	4180	3.12
00000200000	0	0	0	0	0	2	0	0	0	0	0	4049	3.02
00000010001	0	0	0	0	0	0	1	0	0	0	1	4042	3.02
00000000001	0	0	0	0	0	0	0	0	0	0	1	4037	3.01
00000010011	0	0	0	0	0	0	1	0	0	1	1	3704	2.77
00000010000	0	0	0	0	0	0	1	0	0	0	0	3591	2.68
00000100010	0	0	0	0	0	1	0	0	0	1	0	2032	1.52
00000100000	0	0	0	0	0	1	0	0	0	0	0	2003	1.50
00000010110	0	0	0	0	0	0	1	0	1	1	0	1999	1.49
00000001000	0	0	0	0	0	0	0	1	0	0	0	1743	1.30
00000020000	0	0	0	0	0	0	2	0	0	0	0	1742	1.30
00000001101	0	0	0	0	0	0	0	1	1	0	1	1691	1.26
00000020101	0	0	0	0	0	0	2	0	1	0	1	1536	1.15
00000010100	0	0	0	0	0	0	1	0	1	0	0	1501	1.12
00000010010	0	0	0	0	0	0	1	0	0	1	0	1430	1.07
00000021010	0	0	0	0	0	0	2	1	0	1	0	1222	0.91
00000011000	0	0	0	0	0	0	1	1	0	0	0	1221	0.91
00000020001	0	0	0	0	0	0	2	0	0	0	1	1194	0.89
00000000111	0	0	0	0	0	0	0	0	1	1	1	1046	0.78
00000210000	0	0	0	0	0	2	1	0	0	0	0	1035	0.77
00000001100	0	0	0	0	0	0	0	1	1	0	0	997	0.74
00000010111	0	0	0	0	0	0	1	0	1	1	1	965	0.72
00000111010	0	0	0	0	0	1	1	1	0	1	0	835	0.62
00000011100	0	0	0	0	0	0	1	1	1	0	0	691	0.52
00000100011	0	0	0	0	0	1	0	0	0	1	1	681	0.51
00000020100	0	0	0	0	0	0	2	0	1	0	0	647	0.48
00000110000	0	0	0	0	0	1	1	0	0	0	0	572	0.43
00000110011	0	0	0	0	0	1	1	0	0	1	1	563	0.42
00000211010	0	0	0	0	0	2	1	1	0	1	0	543	0.41
00000220000	0	0	0	0	0	2	2	0	0	0	0	467	0.35
00000101110	0	0	0	0	0	1	0	1	1	1	0	450	0.34
00000120010	0	0	0	0	0	1	2	0	0	1	0	423	0.32

00000111011	0	0	0	0	0	1	1	1	0	1	1	411	0.31
00000101011	0	0	0	0	0	1	0	1	0	1	1	395	0.29
00000020010	0	0	0	0	0	0	2	0	0	1	0	378	0.28
00000121010	0	0	0	0	0	1	2	1	0	1	0	334	0.25
00000020110	0	0	0	0	0	0	2	0	1	1	0	308	0.23
00000100110	0	0	0	0	0	1	0	0	1	1	0	294	0.22
00000120001	0	0	0	0	0	1	2	0	0	0	1	279	0.21
00000101100	0	0	0	0	0	1	0	1	1	0	0	227	0.17
00000120000	0	0	0	0	0	1	2	0	0	0	0	222	0.17
00000001001	0	0	0	0	0	0	0	1	0	0	1	221	0.16
00000120101	0	0	0	0	0	1	2	0	1	0	1	210	0.16
00000201011	0	0	0	0	0	2	0	1	0	1	1	170	0.13
00000000100	0	0	0	0	0	0	0	0	1	0	0	157	0.12
00000111100	0	0	0	0	0	1	1	1	1	0	0	150	0.11
00000200200	0	0	0	0	0	2	0	0	2	0	0	145	0.11
00000021011	0	0	0	0	0	0	2	1	0	1	1	140	0.10
00000110111	0	0	0	0	0	1	1	0	1	1	1	135	0.10
00000211011	0	0	0	0	0	2	1	1	0	1	1	134	0.10
00000111111	0	0	0	0	0	1	1	1	1	1	1	131	0.10
00000100111	0	0	0	0	0	1	0	0	1	1	1	121	0.09
00000101111	0	0	0	0	0	1	0	1	1	1	1	116	0.09
00000101000	0	0	0	0	0	1	0	1	0	0	0	109	0.08
00000211110	0	0	0	0	0	2	1	1	1	1	0	100	0.07

AU5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	28687	13.71
00000000010	0	0	0	0	0	0	0	0	0	1	0	21597	10.32
00000000110	0	0	0	0	0	0	0	0	1	1	0	16781	8.02
00000010001	0	0	0	0	0	0	1	0	0	0	1	11045	5.28
00000010000	0	0	0	0	0	0	1	0	0	0	0	9533	4.56
00000010011	0	0	0	0	0	0	1	0	0	1	1	8464	4.05
00000000200	0	0	0	0	0	0	0	0	2	0	0	7909	3.78
00000000001	0	0	0	0	0	0	0	0	0	0	1	7880	3.77
00000010101	0	0	0	0	0	0	1	0	1	0	1	7800	3.73
00000101010	0	0	0	0	0	1	0	1	0	1	0	7375	3.53
00000000011	0	0	0	0	0	0	0	0	0	1	1	7011	3.35
00000000101	0	0	0	0	0	0	0	0	1	0	1	5397	2.58
00000200000	0	0	0	0	0	2	0	0	0	0	0	5011	2.40
00000201010	0	0	0	0	0	2	0	1	0	1	0	4552	2.18
00000100010	0	0	0	0	0	1	0	0	0	1	0	4543	2.17
00000001101	0	0	0	0	0	0	0	1	1	0	1	2834	1.35
00000010010	0	0	0	0	0	0	1	0	0	1	0	2802	1.34
00000010100	0	0	0	0	0	0	1	0	1	0	0	2618	1.25
00000020000	0	0	0	0	0	0	2	0	0	0	0	2439	1.17
00000020001	0	0	0	0	0	0	2	0	0	0	1	2285	1.09
00000001100	0	0	0	0	0	0	0	1	1	0	0	2216	1.06
00000010110	0	0	0	0	0	0	1	0	1	1	0	2162	1.03
00000001000	0	0	0	0	0	0	0	1	0	0	0	1985	0.95
00000100000	0	0	0	0	0	1	0	0	0	0	0	1901	0.91
00000011010	0	0	0	0	0	0	1	1	0	1	0	1827	0.87
00000010111	0	0	0	0	0	0	1	0	1	1	1	1728	0.83
00000210000	0	0	0	0	0	2	1	0	0	0	0	1697	0.81
00000011100	0	0	0	0	0	0	1	1	1	0	0	1590	0.76
00000110001	0	0	0	0	0	1	1	0	0	0	1	1491	0.71
00000020101	0	0	0	0	0	0	2	0	1	0	1	1485	0.71
00000011011	0	0	0	0	0	0	1	1	0	1	1	1405	0.67
00000011000	0	0	0	0	0	0	1	1	0	0	0	1379	0.66
00000000111	0	0	0	0	0	0	0	0	1	1	1	1376	0.66
00000100110	0	0	0	0	0	1	0	0	1	1	0	1254	0.60
00000101110	0	0	0	0	0	1	0	1	1	1	0	1174	0.56
00000111010	0	0	0	0	0	1	1	1	0	1	0	1166	0.56
00000100001	0	0	0	0	0	1	0	0	0	0	1	1014	0.48
00000110011	0	0	0	0	0	1	1	0	0	1	1	873	0.42
00000121010	0	0	0	0	0	1	2	1	0	1	0	732	0.35
00000100011	0	0	0	0	0	1	0	0	0	1	1	726	0.35

00000101011	0	0	0	0	0	1	0	1	0	1	1	630	0.30
00000110000	0	0	0	0	0	1	1	0	0	0	0	611	0.29
00000110010	0	0	0	0	0	1	1	0	0	1	0	610	0.29
00000210101	0	0	0	0	0	2	1	0	1	0	1	604	0.29
00000020011	0	0	0	0	0	0	2	0	0	1	1	603	0.29
00000001001	0	0	0	0	0	0	0	1	0	0	1	600	0.29
00000021010	0	0	0	0	0	0	2	1	0	1	0	596	0.28
00000020010	0	0	0	0	0	0	2	0	0	1	0	583	0.28
00000020100	0	0	0	0	0	0	2	0	1	0	0	573	0.27
00000020110	0	0	0	0	0	0	2	0	1	1	0	474	0.23
00000101100	0	0	0	0	0	1	0	1	1	0	0	473	0.23
00000220000	0	0	0	0	0	2	2	0	0	0	0	459	0.22
00000110101	0	0	0	0	0	1	1	0	1	0	1	438	0.21
00000011201	0	0	0	0	0	0	1	1	2	0	1	419	0.20
00000210001	0	0	0	0	0	2	1	0	0	0	1	411	0.20
00000011111	0	0	0	0	0	0	1	1	1	1	1	408	0.20
00000111100	0	0	0	0	0	1	1	1	1	0	0	403	0.19
00000120001	0	0	0	0	0	1	2	0	0	0	1	395	0.19
00000200101	0	0	0	0	0	2	0	0	1	0	1	388	0.19
00000011110	0	0	0	0	0	0	1	1	1	1	0	341	0.16
00000100101	0	0	0	0	0	1	0	0	1	0	1	303	0.14
00000200001	0	0	0	0	0	2	0	0	0	0	1	274	0.13
00000101111	0	0	0	0	0	1	0	1	1	1	1	258	0.12
00000200200	0	0	0	0	0	2	0	0	2	0	0	242	0.12
00000100200	0	0	0	0	0	1	0	0	2	0	0	220	0.11
00000120101	0	0	0	0	0	1	2	0	1	0	1	206	0.10
00000120000	0	0	0	0	0	1	2	0	0	0	0	192	0.09
00000201110	0	0	0	0	0	2	0	1	1	1	0	161	0.08
00000110110	0	0	0	0	0	1	1	0	1	1	0	143	0.07
00000001201	0	0	0	0	0	0	0	1	2	0	1	140	0.07
00000021110	0	0	0	0	0	0	2	1	1	1	0	137	0.07
00000120010	0	0	0	0	0	1	2	0	0	1	0	126	0.06
00000020111	0	0	0	0	0	0	2	0	1	1	1	117	0.06
00000210100	0	0	0	0	0	2	1	0	1	0	0	100	0.05

V5/AU5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000010	0	0	0	0	0	0	0	0	0	1	0	23611	15.03
00000000000	0	0	0	0	0	0	0	0	0	0	0	18132	11.54
00000101010	0	0	0	0	0	1	0	1	0	1	0	8496	5.41
00000000110	0	0	0	0	0	0	0	0	1	1	0	8435	5.37
00000010000	0	0	0	0	0	0	1	0	0	0	0	7912	5.04
00000100010	0	0	0	0	0	1	0	0	0	1	0	5709	3.63
00000000100	0	0	0	0	0	0	0	0	1	0	0	5644	3.59
00000000001	0	0	0	0	0	0	0	0	0	0	1	5401	3.44
00000201010	0	0	0	0	0	2	0	1	0	1	0	4982	3.17
00000000011	0	0	0	0	0	0	0	0	0	1	1	4941	3.14
00000010011	0	0	0	0	0	0	1	0	0	1	1	4572	2.91
00000000101	0	0	0	0	0	0	0	0	1	0	1	4410	2.81
00000010101	0	0	0	0	0	0	1	0	1	0	1	4202	2.67
00000010010	0	0	0	0	0	0	1	0	0	1	0	4190	2.67
00000200000	0	0	0	0	0	2	0	0	0	0	0	4172	2.66
00000010001	0	0	0	0	0	0	1	0	0	0	1	4066	2.59
00000010100	0	0	0	0	0	0	1	0	1	0	0	2494	1.59
00000010110	0	0	0	0	0	0	1	0	1	1	0	2153	1.37
00000001101	0	0	0	0	0	0	0	1	1	0	1	2057	1.31
00000210000	0	0	0	0	0	2	1	0	0	0	0	1933	1.23
00000020000	0	0	0	0	0	0	2	0	0	0	0	1821	1.16
00000001000	0	0	0	0	0	0	0	1	0	0	0	1778	1.13
00000100000	0	0	0	0	0	1	0	0	0	0	0	1567	1.00
00000111010	0	0	0	0	0	1	1	1	0	1	0	1227	0.78
00000000111	0	0	0	0	0	0	0	0	1	1	1	1161	0.74
00000110011	0	0	0	0	0	1	1	0	0	1	1	1084	0.69
00000001100	0	0	0	0	0	0	0	1	1	0	0	1061	0.68
00000011000	0	0	0	0	0	0	1	1	0	0	0	1038	0.66
00000100110	0	0	0	0	0	1	0	0	1	1	0	1026	0.65
00000110010	0	0	0	0	0	1	1	0	0	1	0	1011	0.64
00000111011	0	0	0	0	0	1	1	1	0	1	1	874	0.56
00000010111	0	0	0	0	0	0	1	0	1	1	1	826	0.53
00000021010	0	0	0	0	0	0	2	1	0	1	0	817	0.52
00000020010	0	0	0	0	0	0	2	0	0	1	0	794	0.51
00000101011	0	0	0	0	0	1	0	1	0	1	1	777	0.49
00000000210	0	0	0	0	0	0	0	0	2	1	0	761	0.48
00000110000	0	0	0	0	0	1	1	0	0	0	0	755	0.48
00000101110	0	0	0	0	0	1	0	1	1	1	0	719	0.46
00000100011	0	0	0	0	0	1	0	0	0	1	1	710	0.45
00000020001	0	0	0	0	0	0	2	0	0	0	1	636	0.40

00000011100	0	0	0	0	0	0	1	1	1	0	0	617	0.39
00000020100	0	0	0	0	0	0	2	0	1	0	0	608	0.39
00000020101	0	0	0	0	0	0	2	0	1	0	1	562	0.36
00000020011	0	0	0	0	0	0	2	0	0	1	1	559	0.36
00000220000	0	0	0	0	0	2	2	0	0	0	0	472	0.30
00000211010	0	0	0	0	0	2	1	1	0	1	0	470	0.30
00000200101	0	0	0	0	0	2	0	0	1	0	1	441	0.28
00000100001	0	0	0	0	0	1	0	0	0	0	1	439	0.28
00000200100	0	0	0	0	0	2	0	0	1	0	0	418	0.27
00000111110	0	0	0	0	0	1	1	1	1	1	0	393	0.25
00000121010	0	0	0	0	0	1	2	1	0	1	0	356	0.23
00000001001	0	0	0	0	0	0	0	1	0	0	1	332	0.21
00000100111	0	0	0	0	0	1	0	0	1	1	1	302	0.19
00000100101	0	0	0	0	0	1	0	0	1	0	1	229	0.15
00000120000	0	0	0	0	0	1	2	0	0	0	0	226	0.14
00000200001	0	0	0	0	0	2	0	0	0	0	1	215	0.14
00000110110	0	0	0	0	0	1	1	0	1	1	0	206	0.13
00000020210	0	0	0	0	0	0	2	0	2	1	0	203	0.13
00000120010	0	0	0	0	0	1	2	0	0	1	0	188	0.12
00000210100	0	0	0	0	0	2	1	0	1	0	0	186	0.12
00000101100	0	0	0	0	0	1	0	1	1	0	0	179	0.11
00000201110	0	0	0	0	0	2	0	1	1	1	0	178	0.11
00000011001	0	0	0	0	0	0	1	1	0	0	1	143	0.09
00000120101	0	0	0	0	0	1	2	0	1	0	1	124	0.08

T7/V5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	16719	13.77
00000000010	0	0	0	0	0	0	0	0	0	1	0	15570	12.82
00000000110	0	0	0	0	0	0	0	0	1	1	0	10495	8.64
00000000200	0	0	0	0	0	0	0	0	2	0	0	6562	5.40
00000000001	0	0	0	0	0	0	0	0	0	0	1	5921	4.88
00000010001	0	0	0	0	0	0	1	0	0	0	1	5716	4.71
00000000101	0	0	0	0	0	0	0	0	1	0	1	5409	4.46
00000010101	0	0	0	0	0	0	1	0	1	0	1	5383	4.43
00000200000	0	0	0	0	0	2	0	0	0	0	0	4319	3.56
00000001010	0	0	0	0	0	0	0	1	0	1	0	4240	3.49
00000001101	0	0	0	0	0	0	0	1	1	0	1	3196	2.63
00000010000	0	0	0	0	0	0	1	0	0	0	0	2804	2.31
00000000011	0	0	0	0	0	0	0	0	0	1	1	2689	2.21
00000010011	0	0	0	0	0	0	1	0	0	1	1	2584	2.13
00000001000	0	0	0	0	0	0	0	1	0	0	0	2199	1.81
00000010010	0	0	0	0	0	0	1	0	0	1	0	2057	1.69
00000001100	0	0	0	0	0	0	0	1	1	0	0	1910	1.57
00000100010	0	0	0	0	0	1	0	0	0	1	0	1644	1.35
00000020000	0	0	0	0	0	0	2	0	0	0	0	1356	1.12
00000001110	0	0	0	0	0	0	0	1	1	1	0	1348	1.11
00000011000	0	0	0	0	0	0	1	1	0	0	0	1310	1.08
00000010100	0	0	0	0	0	0	1	0	1	0	0	1143	0.94
00000011100	0	0	0	0	0	0	1	1	1	0	0	1111	0.92
00000010210	0	0	0	0	0	0	1	0	2	1	0	1022	0.84
00000011011	0	0	0	0	0	0	1	1	0	1	1	820	0.68
00000011101	0	0	0	0	0	0	1	1	1	0	1	801	0.66
00000210000	0	0	0	0	0	2	1	0	0	0	0	725	0.60
00000011010	0	0	0	0	0	0	1	1	0	1	0	713	0.59
00000000111	0	0	0	0	0	0	0	0	1	1	1	664	0.55
00000020001	0	0	0	0	0	0	2	0	0	0	1	655	0.54
00000100110	0	0	0	0	0	1	0	0	1	1	0	649	0.53
00000101010	0	0	0	0	0	1	0	1	0	1	0	634	0.52
00000010111	0	0	0	0	0	0	1	0	1	1	1	618	0.51
00000020100	0	0	0	0	0	0	2	0	1	0	0	597	0.49
00000100000	0	0	0	0	0	1	0	0	0	0	0	573	0.47
00000020101	0	0	0	0	0	0	2	0	1	0	1	572	0.47
00000120010	0	0	0	0	0	1	2	0	0	1	0	567	0.47
00000110001	0	0	0	0	0	1	1	0	0	0	1	512	0.42
00000020110	0	0	0	0	0	0	2	0	1	1	0	496	0.41
00000001001	0	0	0	0	0	0	0	1	0	0	1	412	0.34

00000100001	0	0	0	0	0	1	0	0	0	0	1	411	0.34
00000120000	0	0	0	0	0	1	2	0	0	0	0	366	0.30
00000021010	0	0	0	0	0	0	2	1	0	1	0	288	0.24
00000001011	0	0	0	0	0	0	0	1	0	1	1	286	0.24
00000001201	0	0	0	0	0	0	0	1	2	0	1	249	0.21
00000111010	0	0	0	0	0	1	1	1	0	1	0	247	0.20
00000011110	0	0	0	0	0	0	1	1	1	1	0	242	0.20
00000110101	0	0	0	0	0	1	1	0	1	0	1	238	0.20
00000100101	0	0	0	0	0	1	0	0	1	0	1	199	0.16
00000210101	0	0	0	0	0	2	1	0	1	0	1	184	0.15
00000011111	0	0	0	0	0	0	1	1	1	1	1	164	0.14
00000200200	0	0	0	0	0	2	0	0	2	0	0	157	0.13
00000011001	0	0	0	0	0	0	1	1	0	0	1	114	0.09
00000121010	0	0	0	0	0	1	2	1	0	1	0	110	0.09
00000021110	0	0	0	0	0	0	2	1	1	1	0	100	0.08

T7/AcV5/E2 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	16719	13.77
00000000010	0	0	0	0	0	0	0	0	0	1	0	15570	12.82
00000000110	0	0	0	0	0	0	0	0	1	1	0	10495	8.64
00000000200	0	0	0	0	0	0	0	0	2	0	0	6562	5.40
00000000001	0	0	0	0	0	0	0	0	0	0	1	5921	4.88
00000010001	0	0	0	0	0	0	1	0	0	0	1	5716	4.71
00000000101	0	0	0	0	0	0	0	0	1	0	1	5409	4.46
00000010101	0	0	0	0	0	0	1	0	1	0	1	5383	4.43
00000200000	0	0	0	0	0	2	0	0	0	0	0	4319	3.56
00000001010	0	0	0	0	0	0	0	1	0	1	0	4240	3.49
00000001101	0	0	0	0	0	0	0	1	1	0	1	3196	2.63
00000010000	0	0	0	0	0	0	1	0	0	0	0	2804	2.31
00000000011	0	0	0	0	0	0	0	0	0	1	1	2689	2.21
00000010011	0	0	0	0	0	0	1	0	0	1	1	2584	2.13
00000001000	0	0	0	0	0	0	0	1	0	0	0	2199	1.81
00000010010	0	0	0	0	0	0	1	0	0	1	0	2057	1.69
00000001100	0	0	0	0	0	0	0	1	1	0	0	1910	1.57
00000100010	0	0	0	0	0	1	0	0	0	1	0	1644	1.35
00000020000	0	0	0	0	0	0	2	0	0	0	0	1356	1.12
00000001110	0	0	0	0	0	0	0	1	1	1	0	1348	1.11
00000011000	0	0	0	0	0	0	1	1	0	0	0	1310	1.08
00000010100	0	0	0	0	0	0	1	0	1	0	0	1143	0.94
00000011100	0	0	0	0	0	0	1	1	1	0	0	1111	0.92
00000010210	0	0	0	0	0	0	1	0	2	1	0	1022	0.84
00000011011	0	0	0	0	0	0	1	1	0	1	1	820	0.68
00000011101	0	0	0	0	0	0	1	1	1	0	1	801	0.66
00000210000	0	0	0	0	0	2	1	0	0	0	0	725	0.60
00000011010	0	0	0	0	0	0	1	1	0	1	0	713	0.59
00000000111	0	0	0	0	0	0	0	0	1	1	1	664	0.55
00000020001	0	0	0	0	0	0	2	0	0	0	1	655	0.54
00000100110	0	0	0	0	0	1	0	0	1	1	0	649	0.53
00000101010	0	0	0	0	0	1	0	1	0	1	0	634	0.52
00000010111	0	0	0	0	0	0	1	0	1	1	1	618	0.51
00000020100	0	0	0	0	0	0	2	0	1	0	0	597	0.49
00000100000	0	0	0	0	0	1	0	0	0	0	0	573	0.47
00000020101	0	0	0	0	0	0	2	0	1	0	1	572	0.47
00000120010	0	0	0	0	0	1	2	0	0	1	0	567	0.47
00000110001	0	0	0	0	0	1	1	0	0	0	1	512	0.42
00000020110	0	0	0	0	0	0	2	0	1	1	0	496	0.41
00000001001	0	0	0	0	0	0	0	1	0	0	1	412	0.34

00000100001	0	0	0	0	0	1	0	0	0	0	1	411	0.34
00000120000	0	0	0	0	0	1	2	0	0	0	0	366	0.30
00000021010	0	0	0	0	0	0	2	1	0	1	0	288	0.24
00000001011	0	0	0	0	0	0	0	1	0	1	1	286	0.24
00000001201	0	0	0	0	0	0	0	1	2	0	1	249	0.21
00000111010	0	0	0	0	0	1	1	1	0	1	0	247	0.20
00000011110	0	0	0	0	0	0	1	1	1	1	0	242	0.20
00000110101	0	0	0	0	0	1	1	0	1	0	1	238	0.20
00000100101	0	0	0	0	0	1	0	0	1	0	1	199	0.16
00000210101	0	0	0	0	0	2	1	0	1	0	1	184	0.15
00000011111	0	0	0	0	0	0	1	1	1	1	1	164	0.14
00000200200	0	0	0	0	0	2	0	0	2	0	0	157	0.13
00000011001	0	0	0	0	0	0	1	1	0	0	1	114	0.09
00000121010	0	0	0	0	0	1	2	1	0	1	0	110	0.09
00000021110	0	0	0	0	0	0	2	1	1	1	0	100	0.08

V5/E2 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000010	0	0	0	0	0	0	0	0	0	1	0	24022	12.09
00000000110	0	0	0	0	0	0	0	0	1	1	0	22931	11.54
00000000000	0	0	0	0	0	0	0	0	0	0	0	20983	10.56
00000001010	0	0	0	0	0	0	0	1	0	1	0	16621	8.36
00000000011	0	0	0	0	0	0	0	0	0	1	1	15137	7.62
00000010101	0	0	0	0	0	0	1	0	1	0	1	7402	3.72
00000010001	0	0	0	0	0	0	1	0	0	0	1	7032	3.54
00000000100	0	0	0	0	0	0	0	0	1	0	0	6675	3.36
00000010011	0	0	0	0	0	0	1	0	0	1	1	6381	3.21
00000000101	0	0	0	0	0	0	0	0	1	0	1	5834	2.94
00000000001	0	0	0	0	0	0	0	0	0	0	1	4573	2.30
00000200000	0	0	0	0	0	2	0	0	0	0	0	4172	2.10
00000000111	0	0	0	0	0	0	0	0	1	1	1	3465	1.74
00000010110	0	0	0	0	0	0	1	0	1	1	0	3221	1.62
00000001011	0	0	0	0	0	0	0	1	0	1	1	3211	1.62
00000020000	0	0	0	0	0	0	2	0	0	0	0	3060	1.54
00000010010	0	0	0	0	0	0	1	0	0	1	0	2917	1.47
00000001100	0	0	0	0	0	0	0	1	1	0	0	2653	1.34
00000101010	0	0	0	0	0	1	0	1	0	1	0	2490	1.25
00000011010	0	0	0	0	0	0	1	1	0	1	0	2123	1.07
00000001110	0	0	0	0	0	0	0	1	1	1	0	1963	0.99
00000100010	0	0	0	0	0	1	0	0	0	1	0	1914	0.96
00000001000	0	0	0	0	0	0	0	1	0	0	0	1878	0.95
00000010000	0	0	0	0	0	0	1	0	0	0	0	1832	0.92
00000111011	0	0	0	0	0	1	1	1	0	1	1	1706	0.86
00000001101	0	0	0	0	0	0	0	1	1	0	1	1645	0.83
00000021010	0	0	0	0	0	0	2	1	0	1	0	1509	0.76
00000010111	0	0	0	0	0	0	1	0	1	1	1	1443	0.73
00000100011	0	0	0	0	0	1	0	0	0	1	1	1247	0.63
00000020100	0	0	0	0	0	0	2	0	1	0	0	1177	0.59
00000110011	0	0	0	0	0	1	1	0	0	1	1	1122	0.56
00000011100	0	0	0	0	0	0	1	1	1	0	0	814	0.41
00000010100	0	0	0	0	0	0	1	0	1	0	0	775	0.39
00000210101	0	0	0	0	0	2	1	0	1	0	1	749	0.38
00000020101	0	0	0	0	0	0	2	0	1	0	1	713	0.36
00000020110	0	0	0	0	0	0	2	0	1	1	0	682	0.34
00000011000	0	0	0	0	0	0	1	1	0	0	0	621	0.31
00000020010	0	0	0	0	0	0	2	0	0	1	0	613	0.31
00000111010	0	0	0	0	0	1	1	1	0	1	0	613	0.31
00000100110	0	0	0	0	0	1	0	0	1	1	0	606	0.30

00000220000	0	0	0	0	0	2	2	0	0	0	0	575	0.29
00000001111	0	0	0	0	0	0	0	1	1	1	1	566	0.28
00000020001	0	0	0	0	0	0	2	0	0	0	1	547	0.28
00000101100	0	0	0	0	0	1	0	1	1	0	0	531	0.27
00000100000	0	0	0	0	0	1	0	0	0	0	0	511	0.26
00000011101	0	0	0	0	0	0	1	1	1	0	1	507	0.26
00000110001	0	0	0	0	0	1	1	0	0	0	1	471	0.24
00000110010	0	0	0	0	0	1	1	0	0	1	0	436	0.22
00000210001	0	0	0	0	0	2	1	0	0	0	1	395	0.20
00000210000	0	0	0	0	0	2	1	0	0	0	0	376	0.19
00000001001	0	0	0	0	0	0	0	1	0	0	1	372	0.19
00000200100	0	0	0	0	0	2	0	0	1	0	0	346	0.17
00000121010	0	0	0	0	0	1	2	1	0	1	0	344	0.17
00000200101	0	0	0	0	0	2	0	0	1	0	1	341	0.17
00000101011	0	0	0	0	0	1	0	1	0	1	1	337	0.17
00000111111	0	0	0	0	0	1	1	1	1	1	1	245	0.12
00000001200	0	0	0	0	0	0	0	1	2	0	0	233	0.12
00000021110	0	0	0	0	0	0	2	1	1	1	0	204	0.10
00000110110	0	0	0	0	0	1	1	0	1	1	0	190	0.10
00000110101	0	0	0	0	0	1	1	0	1	0	1	185	0.09
00000200001	0	0	0	0	0	2	0	0	0	0	1	184	0.09
00000211011	0	0	0	0	0	2	1	1	0	1	1	179	0.09
00000111100	0	0	0	0	0	1	1	1	1	0	0	163	0.08
00000011110	0	0	0	0	0	0	1	1	1	1	0	143	0.07
00000101110	0	0	0	0	0	1	0	1	1	1	0	139	0.07
00000100001	0	0	0	0	0	1	0	0	0	0	1	137	0.07
00000011001	0	0	0	0	0	0	1	1	0	0	1	130	0.07
00000101000	0	0	0	0	0	1	0	1	0	0	0	127	0.06
00000120000	0	0	0	0	0	1	2	0	0	0	0	123	0.06
00000110111	0	0	0	0	0	1	1	0	1	1	1	119	0.06

V5/AcV5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	12556	13.26
00000101010	0	0	0	0	0	1	0	1	0	1	0	6633	7.01
00000000010	0	0	0	0	0	0	0	0	0	1	0	6585	6.95
00000000110	0	0	0	0	0	0	0	0	1	1	0	6575	6.94
00000000100	0	0	0	0	0	0	0	0	1	0	0	6510	6.88
00000010101	0	0	0	0	0	0	1	0	1	0	1	4244	4.48
00000010001	0	0	0	0	0	0	1	0	0	0	1	3801	4.01
00000000101	0	0	0	0	0	0	0	0	1	0	1	3543	3.74
00000000001	0	0	0	0	0	0	0	0	0	0	1	3195	3.37
00000020000	0	0	0	0	0	0	2	0	0	0	0	3122	3.30
00000010000	0	0	0	0	0	0	1	0	0	0	0	3014	3.18
00000010011	0	0	0	0	0	0	1	0	0	1	1	2387	2.52
00000000011	0	0	0	0	0	0	0	0	0	1	1	1936	2.04
00000100010	0	0	0	0	0	1	0	0	0	1	0	1778	1.88
00000001101	0	0	0	0	0	0	0	1	1	0	1	1710	1.81
00000010100	0	0	0	0	0	0	1	0	1	0	0	1701	1.80
00000020100	0	0	0	0	0	0	2	0	1	0	0	1700	1.80
00000010110	0	0	0	0	0	0	1	0	1	1	0	1620	1.71
00000200000	0	0	0	0	0	2	0	0	0	0	0	1578	1.67
00000001100	0	0	0	0	0	0	0	1	1	0	0	1547	1.63
00000101110	0	0	0	0	0	1	0	1	1	1	0	1522	1.61
00000201010	0	0	0	0	0	2	0	1	0	1	0	1422	1.50
00000010010	0	0	0	0	0	0	1	0	0	1	0	1288	1.36
00000100000	0	0	0	0	0	1	0	0	0	0	0	1043	1.10
00000001011	0	0	0	0	0	0	0	1	0	1	1	989	1.04
00000011100	0	0	0	0	0	0	1	1	1	0	0	762	0.80
00000001000	0	0	0	0	0	0	0	1	0	0	0	739	0.78
00000010111	0	0	0	0	0	0	1	0	1	1	1	667	0.70
00000110011	0	0	0	0	0	1	1	0	0	1	1	582	0.61
00000110101	0	0	0	0	0	1	1	0	1	0	1	570	0.60
00000011010	0	0	0	0	0	0	1	1	0	1	0	569	0.60
00000000111	0	0	0	0	0	0	0	0	1	1	1	561	0.59
00000011011	0	0	0	0	0	0	1	1	0	1	1	525	0.55
00000120010	0	0	0	0	0	1	2	0	0	1	0	513	0.54
00000100011	0	0	0	0	0	1	0	0	0	1	1	503	0.53
00000100101	0	0	0	0	0	1	0	0	1	0	1	475	0.50
00000220000	0	0	0	0	0	2	2	0	0	0	0	408	0.43
00000021010	0	0	0	0	0	0	2	1	0	1	0	402	0.42
00000020101	0	0	0	0	0	0	2	0	1	0	1	387	0.41
00000020010	0	0	0	0	0	0	2	0	0	1	0	383	0.40

00000210000	0	0	0	0	0	2	1	0	0	0	0	380	0.40
00000011000	0	0	0	0	0	0	1	1	0	0	0	358	0.38
00000020110	0	0	0	0	0	0	2	0	1	1	0	321	0.34
00000110000	0	0	0	0	0	1	1	0	0	0	0	320	0.34
00000020001	0	0	0	0	0	0	2	0	0	0	1	301	0.32
00000120000	0	0	0	0	0	1	2	0	0	0	0	297	0.31
00000011101	0	0	0	0	0	0	1	1	1	0	1	295	0.31
00000001001	0	0	0	0	0	0	0	1	0	0	1	291	0.31
00000100001	0	0	0	0	0	1	0	0	0	0	1	253	0.27
00000110001	0	0	0	0	0	1	1	0	0	0	1	237	0.25
00000101100	0	0	0	0	0	1	0	1	1	0	0	189	0.20
00000200100	0	0	0	0	0	2	0	0	1	0	0	138	0.15
00000011110	0	0	0	0	0	0	1	1	1	1	0	134	0.14
00000111010	0	0	0	0	0	1	1	1	0	1	0	105	0.11
00000111100	0	0	0	0	0	1	1	1	1	0	0	103	0.11
00000021110	0	0	0	0	0	0	2	1	1	1	0	101	0.11

V5/AcV5/E2 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000110	0	0	0	0	0	0	0	0	1	1	0	19351	13.10
00000010001	0	0	0	0	0	0	1	0	0	0	1	19328	13.09
00000000000	0	0	0	0	0	0	0	0	0	0	0	18618	12.61
00000000100	0	0	0	0	0	0	0	0	1	0	0	14504	9.82
00000100100	0	0	0	0	0	1	0	0	1	0	0	12660	8.57
00000000001	0	0	0	0	0	0	0	0	0	0	1	12574	8.51
00000100000	0	0	0	0	0	1	0	0	0	0	0	9969	6.75
00000010101	0	0	0	0	0	0	1	0	1	0	1	9295	6.29
00000101010	0	0	0	0	0	1	0	1	0	1	0	7229	4.89
00000000101	0	0	0	0	0	0	0	0	1	0	1	7101	4.81
00000020000	0	0	0	0	0	0	2	0	0	0	0	1618	1.10
00000020100	0	0	0	0	0	0	2	0	1	0	0	1546	1.05
00000201010	0	0	0	0	0	2	0	1	0	1	0	1357	0.92
00000000010	0	0	0	0	0	0	0	0	0	1	0	1321	0.89
00000120100	0	0	0	0	0	1	2	0	1	0	0	1277	0.86
00000010000	0	0	0	0	0	0	1	0	0	0	0	1015	0.69
00000001001	0	0	0	0	0	0	0	1	0	0	1	997	0.68
00000001000	0	0	0	0	0	0	0	1	0	0	0	993	0.67
00000010100	0	0	0	0	0	0	1	0	1	0	0	917	0.62
00000120000	0	0	0	0	0	1	2	0	0	0	0	865	0.59
00000110100	0	0	0	0	0	1	1	0	1	0	0	808	0.55
00000110011	0	0	0	0	0	1	1	0	0	1	1	805	0.55
00000001101	0	0	0	0	0	0	0	1	1	0	1	801	0.54
00000110000	0	0	0	0	0	1	1	0	0	0	0	590	0.40
00000010111	0	0	0	0	0	0	1	0	1	1	1	455	0.31
00000200011	0	0	0	0	0	2	0	0	0	1	1	320	0.22
00000110101	0	0	0	0	0	1	1	0	1	0	1	302	0.20
00000110001	0	0	0	0	0	1	1	0	0	0	1	249	0.17
00000001201	0	0	0	0	0	0	0	1	2	0	1	232	0.16
00000010011	0	0	0	0	0	0	1	0	0	1	1	208	0.14
00000001011	0	0	0	0	0	0	0	1	0	1	1	136	0.09
00000001111	0	0	0	0	0	0	0	1	1	1	1	106	0.07

AU5/AcV5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000010	0	0	0	0	0	0	0	0	0	1	0	16666	16.87
00000000000	0	0	0	0	0	0	0	0	0	0	0	13654	13.82
00000000110	0	0	0	0	0	0	0	0	1	1	0	11661	11.80
00000010001	0	0	0	0	0	0	1	0	0	0	1	6993	7.08
00000000001	0	0	0	0	0	0	0	0	0	0	1	5538	5.60
00000010101	0	0	0	0	0	0	1	0	1	0	1	5091	5.15
00000000100	0	0	0	0	0	0	0	0	1	0	0	4905	4.96
00000000101	0	0	0	0	0	0	0	0	1	0	1	3296	3.34
00000010011	0	0	0	0	0	0	1	0	0	1	1	3293	3.33
00000010000	0	0	0	0	0	0	1	0	0	0	0	2771	2.80
00000101010	0	0	0	0	0	1	0	1	0	1	0	2595	2.63
00000000011	0	0	0	0	0	0	0	0	0	1	1	2377	2.41
00000020000	0	0	0	0	0	0	2	0	0	0	0	1535	1.55
00000201010	0	0	0	0	0	2	0	1	0	1	0	1352	1.37
00000020011	0	0	0	0	0	0	2	0	0	1	1	1202	1.22
00000010100	0	0	0	0	0	0	1	0	1	0	0	1026	1.04
00000011111	0	0	0	0	0	0	1	1	1	1	1	1007	1.02
00000100000	0	0	0	0	0	1	0	0	0	0	0	988	1.00
00000001100	0	0	0	0	0	0	0	1	1	0	0	963	0.97
00000010010	0	0	0	0	0	0	1	0	0	1	0	942	0.95
00000200000	0	0	0	0	0	2	0	0	0	0	0	934	0.95
00000001101	0	0	0	0	0	0	0	1	1	0	1	866	0.88
00000010110	0	0	0	0	0	0	1	0	1	1	0	695	0.70
00000110101	0	0	0	0	0	1	1	0	1	0	1	694	0.70
00000021110	0	0	0	0	0	0	2	1	1	1	0	680	0.69
00000211010	0	0	0	0	0	2	1	1	0	1	0	647	0.65
00000020100	0	0	0	0	0	0	2	0	1	0	0	553	0.56
00000110001	0	0	0	0	0	1	1	0	0	0	1	523	0.53
00000011100	0	0	0	0	0	0	1	1	1	0	0	488	0.49
00000110000	0	0	0	0	0	1	1	0	0	0	0	418	0.42
00000100101	0	0	0	0	0	1	0	0	1	0	1	408	0.41
00000100001	0	0	0	0	0	1	0	0	0	0	1	400	0.40
00000001011	0	0	0	0	0	0	0	1	0	1	1	338	0.34
00000101110	0	0	0	0	0	1	0	1	1	1	0	337	0.34
00000020001	0	0	0	0	0	0	2	0	0	0	1	313	0.32
00000020101	0	0	0	0	0	0	2	0	1	0	1	271	0.27
00000001201	0	0	0	0	0	0	0	1	2	0	1	256	0.26
00000021210	0	0	0	0	0	0	2	1	2	1	0	255	0.26
00000121110	0	0	0	0	0	1	2	1	1	1	0	163	0.16
00000011101	0	0	0	0	0	0	1	1	1	0	1	150	0.15
00000021200	0	0	0	0	0	0	2	1	2	0	0	139	0.14
00000021100	0	0	0	0	0	0	2	1	1	0	0	131	0.13
00000021011	0	0	0	0	0	0	2	1	0	1	1	111	0.11
00000220000	0	0	0	0	0	2	2	0	0	0	0	107	0.11

T7/AcV5 Library

Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Number points	Abundance
00000000000	0	0	0	0	0	0	0	0	0	0	0	56769	51.02
00000000010	0	0	0	0	0	0	0	0	0	1	0	8352	7.51
00000000011	0	0	0	0	0	0	0	0	0	1	1	7183	6.46
00000000100	0	0	0	0	0	0	0	0	1	0	0	4397	3.95
00000000101	0	0	0	0	0	0	0	0	1	0	1	4276	3.84
00000010000	0	0	0	0	0	0	1	0	0	0	0	3921	3.52
00000000001	0	0	0	0	0	0	0	0	0	0	1	3029	2.72
00000110101	0	0	0	0	0	1	1	0	1	0	1	2628	2.36
00000000110	0	0	0	0	0	0	0	0	1	1	0	2319	2.08
00000020000	0	0	0	0	0	0	2	0	0	0	0	1850	1.66
00000010010	0	0	0	0	0	0	1	0	0	1	0	1757	1.58
00000110001	0	0	0	0	0	1	1	0	0	0	1	1517	1.36
00000210001	0	0	0	0	0	2	1	0	0	0	1	1455	1.31
00000100000	0	0	0	0	0	1	0	0	0	0	0	1383	1.24
00000100001	0	0	0	0	0	1	0	0	0	0	1	1253	1.13
00000101010	0	0	0	0	0	1	0	1	0	1	0	1141	1.03
00000001001	0	0	0	0	0	0	0	1	0	0	1	1135	1.02
00000120101	0	0	0	0	0	1	2	0	1	0	1	817	0.73
00000010011	0	0	0	0	0	0	1	0	0	1	1	505	0.45
00000010110	0	0	0	0	0	0	1	0	1	1	0	495	0.44
00000120001	0	0	0	0	0	1	2	0	0	0	1	464	0.42
00000000111	0	0	0	0	0	0	0	0	1	1	1	452	0.41
00000100011	0	0	0	0	0	1	0	0	0	1	1	417	0.37
00000220001	0	0	0	0	0	2	2	0	0	0	1	402	0.36
00000011000	0	0	0	0	0	0	1	1	0	0	0	401	0.36
00000001000	0	0	0	0	0	0	0	1	0	0	0	398	0.36
00000010100	0	0	0	0	0	0	1	0	1	0	0	384	0.35
00000101011	0	0	0	0	0	1	0	1	0	1	1	345	0.31
00000111010	0	0	0	0	0	1	1	1	0	1	0	333	0.30
00000011011	0	0	0	0	0	0	1	1	0	1	1	209	0.19
00000020100	0	0	0	0	0	0	2	0	1	0	0	179	0.16
00000101110	0	0	0	0	0	1	0	1	1	1	0	158	0.14
00000201010	0	0	0	0	0	2	0	1	0	1	0	140	0.13
00000011100	0	0	0	0	0	0	1	1	1	0	0	130	0.12
00000001100	0	0	0	0	0	0	0	1	1	0	0	111	0.10

Appendix I

BARCODED YEAST GFP FUSION PROTEINS STUDIED AND PERTURBATION CONDITIONS USED

Table I.1: Barcoded yeast GFP fusion clones used in this study

GFP Clone ORF	GFP Clone Name	Barcode	T7	V5	AU5	AcV5	E2	HA	HSV	HIS	AU1	GLU	FLAG	Description
YBL023C	MCM2	10011000000	1	0	0	1	1	0	0	0	0	0	0	Protein involved in DNA replication
YBL024W	NCL1	01011000000	0	1	0	1	1	0	0	0	0	0	0	S-adenosyl-L-methionine-dependent tRNA: m5C-methyltransferase
YBR067C	TIP1	10011200000	1	0	0	1	1	2	0	0	0	0	0	Major cell wall mannoprotein with possible lipase activity
YBR072W	HSP26	10011010000	1	0	0	1	1	0	1	0	0	0	0	Small heat shock protein (sHSP) with chaperone activity
YBR208C	DUR1,2	00010000000	0	0	0	1	0	0	0	0	0	0	0	Urea amidolyase
YCR016W		01100100000	0	1	1	0	0	1	0	0	0	0	0	Putative protein of unknown function
YDL133C-A	RPL41B	00000220100	0	0	0	0	0	2	2	0	1	0	0	Ribosomal 60S subunit protein L41B
YDL168W	SFA1	01000220000	0	1	0	0	0	2	2	0	0	0	0	Bifunctional alcohol dehydrogenase and formaldehyde dehydrogenase
YDR023W	SES1	10000200011	1	0	0	0	0	2	0	0	0	1	1	Cytosolic seryl-tRNA synthetase
YDR070C	FMP16	00000220000	0	0	0	0	0	2	2	0	0	0	0	Protein of unknown function
YEL001C	IRC22	00110000000	0	0	1	1	0	0	0	0	0	0	0	Protein of unknown function
YER027C	GAL83	00001000010	0	0	0	0	1	0	0	0	0	1	0	One of three possible beta-subunits of the Snf1 kinase complex
YER178W	PDA1	00001100000	0	0	0	0	1	1	0	0	0	0	0	E1 alpha subunit of the pyruvate dehydrogenase (PDH) complex
YFL036W	RPO41	00100000000	0	0	1	0	0	0	0	0	0	0	0	Mitochondrial RNA polymerase
YGL035C	MIG1	00100100000	0	0	1	0	0	1	0	0	0	0	0	Transcription factor involved in glucose repression
YGL048C	RPT6	01000010000	0	1	0	0	0	0	1	0	0	0	0	ATPase of the 19S regulatory particle of the 26S proteasome
YGL135W	RPL1B	10011020000	1	0	0	1	1	0	2	0	0	0	0	Ribosomal 60S subunit protein L1B
YGL147C	RPL9A	01001020000	0	1	0	0	1	0	2	0	0	0	0	Ribosomal 60S subunit protein L9A
YGL207W	SPT16	10000000000	1	0	0	0	0	0	0	0	0	0	0	Subunit of the heterodimeric FACT complex (Spt16p-Pob3p)
YGL234W	ADE5,7	01010000000	0	1	0	1	0	0	0	0	0	0	0	Enzyme of the 'de novo' purine nucleotide biosynthetic pathway
YGR012W	MCY1	10000000010	1	0	0	0	0	0	0	0	0	1	0	Putative cysteine synthase
YGR019W	UGA1	10000010000	1	0	0	0	0	0	1	0	0	0	0	Gamma-aminobutyrate (GABA) transaminase
YHL017W		00000020000	0	0	0	0	0	0	2	0	0	0	0	Putative protein of unknown function
YIL127C	RRT14	10011100000	1	0	0	1	1	1	0	0	0	0	0	Putative protein of unknown function
YIL137C	TMA108	01000000000	0	1	0	0	0	0	0	0	0	0	0	Ribosome-associated, nascent chain binding factor
YJR003C	MRX12	11100000000	1	1	1	0	0	0	0	0	0	0	0	Protein that associates with mitochondrial ribosome
YKL096W	CWP1	11000020000	1	1	0	0	0	0	2	0	0	0	0	Cell wall mannoprotein that localizes to birth scars of daughter cells
YKL096W-A	CWP2	00000200010	0	0	0	0	0	2	0	0	0	1	0	Covalently linked cell wall mannoprotein
YKL142W	MRP8	01001010000	0	1	0	0	1	0	1	0	0	0	0	Protein of unknown function
YKL183W	LOT5	01000220000	0	1	0	0	0	2	2	0	0	0	0	Protein of unknown function
YLR064W	PER33	11001010000	1	1	0	0	1	0	1	0	0	0	0	Protein that localizes to the endoplasmic reticulum
YLR438W	CAR2	00001000100	0	0	0	0	1	0	0	0	1	0	0	L-ornithine transaminase (OTAse)
YML100W	TSL1	00001000000	0	0	0	0	1	0	0	0	0	0	0	Large subunit of trehalose 6-phosphate synthase/phosphatase complex
YMR099C		11000010000	1	1	0	0	0	0	1	0	0	0	0	Glucose-6-phosphate 1-epimerase (hexose-6-phosphate mutarotase)
YMR120C	ADE17	10010000000	1	0	0	1	0	0	0	0	0	0	0	Enzyme of 'de novo' purine biosynthesis
YNL181W	PBR1	00100000010	0	0	1	0	0	0	0	0	0	1	0	Putative oxidoreductase
YNL212W	VID27	00000020011	0	0	0	0	0	0	2	0	0	1	1	Cytoplasmic protein of unknown function
YOL143C	RIB4	01001000000	0	1	0	0	1	0	0	0	0	0	0	Lumazine synthase (DMRL synthase)
YOL144W	NOP8	00100220000	0	0	1	0	0	2	2	0	0	0	0	Nucleolar protein required for 60S ribosomal subunit biogenesis
YOL151W	GRE2	11001110000	1	1	0	0	1	1	1	0	0	0	0	3-methylbutanal reductase and NADPH-dependent methylglyoxal reductase
YOR083W	WHI5	11000100000	1	1	0	0	0	1	0	0	0	0	0	Repressor of G1 transcription
YOR182C	RPS30B	01001000011	0	1	0	0	1	0	0	0	0	1	1	Protein component of the small (40S) ribosomal subunit
YOR216C	RUD3	10000000100	1	0	0	0	0	0	0	0	1	0	0	Golgi matrix protein
YOR312C	RPL20B	11001100000	1	1	0	0	1	1	0	0	0	0	0	Ribosomal 60S subunit protein L20B
YPL079W	RPL21B	11001000000	1	1	0	0	1	0	0	0	0	0	0	Ribosomal 60S subunit protein L21B
YPR160W	GPH1	11000000000	1	1	0	0	0	0	0	0	0	0	0	Glycogen phosphorylase required for the mobilization of glycogen

Table I.2: Stress perturbations, responses, effective concentrations, and positive control proteins.

Perturbation	Agent	Mechanism of action	Concentration(s) used in literature	Selected Control GFP fusion proteins (abundance)	Citations
Oxidative stress	menadione (superoxide), H2O2, diamide	Menadione generates superoxide through futile redox cycling. Diamide is a thiol specific oxidant that promotes disulfide bond formation. Oxidative stress can act on proteins, DNA, and lipid, and interefere with cellular signalling	Menadione: 0-40 mM diamide: 1.5mM. H2O2: 0.3-4mM	CTT1 (319), CTA1 (623), SOD1 (519000), SOD2 (11000), TRX2 (17200), GRX1 (3000), GRX2 (31000),TRR1 (290000)	Jamieson 1998 Yeast, SGD
DNA replication stress	hydroxurea (HU)	HU Limits dNTP pools by inhibition of ribonucleotide reductase (enzyme catalyzes formation of dNTPs from ribonucleotides)	0.2M up to 8h, 0.1M 4h	CDC19 (291000) TPS1 (YBR126C) (11000), BMH2	Tkasch 2012 Nat Cell Biol
DNA replication stress	MMS (methyl methanesulfonate)	Alkylating agent that damages DNA (methylation) preventing DNA polymerase from replicating DNA (stalls replication forks)	0.03% 2h, 0.05-0.1%, 0.05% up to 4 hours	(YDR099W) (48000)	Tkasch 2012 Nat Cell Biol
Loss of proteostasis	MG132	Inhibits proteasome activity by binding to the active site beta subunits on the 20S core	50 uM up to 4h, 20uM up to 3h	HSP104 (32800), HSP70 (6440), SIS1 (20300), YDJ1 (119000)	Lee & Goldberg Mol Cell Biol 1998
Loss of proteostasis	Cycloheximide	Inhibits protein synthesis in eukaryotes by blocking translational elongation	100ug/mL up to 4h	PDR5 (42000)	Katzmann Mol Cell Bio 1994
Heat stress	37C incubation	Heat shock proteins (HSPs) act to repair, refold, or degrade damaged proteins, also induces metabolic changes including increased trehalose synthesis, cell cycle arrest, transcriptional program activated by Hsf1 transcription factor	N/A	HSPs	SGD, Verghese J. 2012 Microbiol. and Mol. Bio. Rev.
Nutrient starvation response	rapamycin	TOR inhibitor protein kinase that influences cell growth (starvation response, amino acid biosynthesis, transcription and translation, stress response genes, TCA cycle, rRNA. ribosomal protein synthesis)	200ng/ml	TDH1 (120000), QCR2 (35000), HSP26 (19300)	Chong 2015 Cell
Reducing stress	DTT	Membrane permeable reducing agent that typically acts to reduce thiol groups or prevents cysteine residues form forming disulfide bonds	2mM	PUN1 YLR414C (1600), YIP3 YNL044W (5500), YNL162W RPL42A (13600)	Breker 2012 JCB
Osmotic stress	KCl, NaCl, sorbitol	Causes water loss resulting in cell shrinkage, cell growth arrest, and activation of the high-osmolarity glycerol (HOG) MAPK pathway which promotes glycerol synthesis and sodium/potassium efflux pump activity	KCl 0.2-0.8M, Sorbitol: 1M, NaCl: up to 1M	YMR251W-A HOR7 (6000), YER062C HOR2 (5000), GPP1 (190000)	Chen 2003 Mol. Biol. Cell, Zhang 2017 Sci. Rep., Mager 2002 FEMS Yeast Res.
Alcohol stress	EtOH	Increases plasma membrane fluidity by increased production of ergosterol and unsaturated fatty acids. Factors that improve protein stability or repair are upregulated including trehalose, heat shock proteins, transcription factors under stress-responsive element	5%	HSP12 (4000), HSP26 (19000), HSP30 (7800), HSP42 (1400), HSP78 (3000), HSP82 (445000), HSP104 (32800), CTT1 (319), DDR2 (400), SSA4 (16000),	Stanley J Appl Microbiol 2010, Ding 2009 Appl. Microbio. Biotech.
Membrane stress	Clotrimazole	Inhibits ergosterol biosynthesis causing increased membrane permeability	10uM	ERG11 (73000), ERG13 (35000), ERG3 (36000)	Sud I.J. 1981 J. Investig. Derm.

Appendix J

PYTHON SCRIPT USED TO ANALYZE GFP FLUORESCENCE DISTRIBUTIONS

```
from collections import defaultdict
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import os
import scipy.stats as stats
import matplotlib.mlab as mlab
import json

protein_fname = '/Users/Stefanie/Desktop/Protein_Names.txt'
condition_fname = '/Users/Stefanie/Desktop/Condition_Names.txt'
data_directory = '/Users/Stefanie/Desktop/07112017_GFPBAR_EXPORTS'

print 'loading data'
#read protein names and append to list
proteins_list = []
with open(protein_fname,'rU') as f:
    p_list = f.readlines()
    for line in p_list:
        line = line.strip('\n')
        proteins_list.append(line)
#print proteins_list
print '# proteins',len(proteins_list)

#read condition names and append to list
conditions_list = []
with open(condition_fname,'rU') as f:
    c_list = f.readlines()
    for line in c_list:
        line = line.strip('\n')
        conditions_list.append(line)
#print conditions_list
#print len(conditions_list)


#read data file names and append to list
data_names_list = []
for file in os.listdir(data_directory):
    if file.endswith('.csv') and file != 'BY4741.csv':
        data_names_list.append(file)
print '# data files',len(data_names_list)
#print data_names_list[1:10]

#load BY4741 background data
```

```
background_fname = data_directory+'/BY4741.csv'
BY4741_data = []
with open(background_fname,'rU') as f:
    back_data = f.readlines()[1:]
    for row in back_data:
        row.strip('\n')
        dpoint = float(row)
        if dpoint >1:
            BY4741_data.append(dpoint)
BY4741_data = np.log10(BY4741_data)

linear_BY4741_data = np.power(BY4741_data,10)
median_BY4741 = np.median(linear_BY4741_data)
std_BY4741 = np.std(linear_BY4741_data)
#print BY4741_data


print 'storing data in dictionary'
data_dict = defaultdict(lambda: defaultdict(list))

#store data in dictionary with keys protein name, condition name, value list
of GFP fluorescence values
for name in data_names_list:
    #print name
    for protein in proteins_list:
        #print protein
        for condition in conditions_list:
            #print condition
            if protein in name and condition in name:
                data_list = []
                with open(data_directory+'/'+str(name),'rU') as f:
                    data = f.readlines()[1:]
                    for row in data:
                        row.strip('\n')
                        dpoint = float(row)
                        if dpoint > 1:
                            data_list.append(dpoint)
                data_list_log = np.log10(data_list)
                data_dict[protein][condition] = data_list_log


#print data_dict['YMR099C']['37C'][0:10]
#print 'banana'
#print data_dict['SES1']['37C'][0:10]
#plot data
plot_var = 'no'
if plot_var == 'yes':
    print 'plotting data'
    colors =
['gray','silver','g','b','m','y','orange','dodgerblue','aqua','lime','maroon'
,'hotpink','darkviolet']
    p_counter = 1
    for protein in data_dict.keys():
        print 'protein is:',protein
        graph = 1
        #plt.figure()
        f,ax =
```

```
plt.subplots(len(conditions_list)+1,1,sharex=True,sharey=True,figsize=(4,10))

        #plot background histogram

ax[0].hist(BY4741_data,bins=500,normed=True,fc='k',histtype='stepfilled')
        ax[0].get_yaxis().set_visible(False)
        if p_counter ==1:
            ax[0].legend(['Background'],fontsize='medium',loc=7)

        #plot condition histograms
        for condition in data_dict[protein]:
            if condition == 'None_0h':
                r = 1
            elif condition == 'None_2h':
                r = 2
            else:
                r = 2+graph
            #print condition
            curr_data = data_dict[protein][condition]
            #print 'r is:',r

            est_data_len = int(np.log10(len(curr_data)))
            #print 'approx log10 length data:',est_data_len

            #estimate number bins based on data set size
            if est_data_len == 2:
                b = 50
            elif est_data_len == 3:
                b = 500
            elif est_data_len >= 4:
                b = 1000
            elif est_data_len == 1:
                b = 50
            elif est_data_len == 0:
                b = 1

            #plot histogram GFP data
            [n,bins,patches] =
ax[r].hist(curr_data,bins=b,normed=True,fc=colors[r-1],histtype='stepfilled')
            #ax[r].set_yticks(np.arange(0,2,1))
            ax[r].get_yaxis().set_visible(False)
            plt.xlim((2,5))
            plt.ylim(0,2)
            if p_counter == 1:
                ax[r].legend([condition],fontsize='medium',loc=7)

            #plot best fit normal curve
            [mu,sigma] = stats.norm.fit(curr_data)
            #[a,l,b] = stats.gamma.fit(curr_data)
            y = mlab.normpdf(bins,mu,sigma)
            ax[r].plot(bins, y, 'r--', linewidth=2)
            #ax[r].plot(stats.gamma.pdf(bins,a,l,b))

            if condition not in ['None_0h','None_2h']:
                graph+=1

        p_counter+=1
```

```
        f.suptitle(protein,fontsize=20)
        plt.ylabel('Normalized Count')
        plt.xlabel('Log10 GFP Fluorescence (afu)')
        #plt.show()
        save_file = '/Users/Stefanie/PycharmProjects/GFPplots/pdffits/'
        plt.savefig(save_file+protein+'.png',dpi=300)


print 'calculating statistics'
#calculate median and CV for each protein and condition, store in stats_dict
#calcuate 95% CI for control conditions, store in control_stats_dict
stats_dict = defaultdict(lambda: defaultdict( lambda:defaultdict(float)))
control_stats_dict = defaultdict(lambda: defaultdict(float))

for protein in data_dict.keys():
    for condition in data_dict[protein]:
        #print condition
        #linear transformation
        curr_data = np.power(data_dict[protein][condition],10)
        curr_med = np.median(curr_data)
        curr_sd = np.std(curr_data)
        curr_cv = float(curr_sd)/float(curr_med)
        #print curr_med,curr_sd,curr_cv

        stats_dict[protein][condition]['median'] = curr_med
        stats_dict[protein][condition]['sd'] = curr_sd
        stats_dict[protein][condition]['cv'] = curr_cv

    control_med_r1 = stats_dict[protein]['None_0h']['median']
    control_med_r2 = stats_dict[protein]['None_2h']['median']

    control_cv_r1 = stats_dict[protein]['None_0h']['cv']
    control_cv_r2 = stats_dict[protein]['None_2h']['cv']

    control_sd_r1 = stats_dict[protein]['None_0h']['sd']
    control_sd_r2 = stats_dict[protein]['None_2h']['sd']

    #calculate average median and CV
    #calculate standard deviation between two replicates for median and CV
    control_median_avg = np.mean([control_med_r1,control_med_r2])
    control_median_sd = np.std([control_med_r1,control_med_r2])
    control_cv_avg = np.mean([control_cv_r1,control_cv_r2])
    control_cv_sd = np.std([control_cv_r1,control_cv_r2])

    control_median_CI_upper = control_median_avg+3*control_median_sd
    control_median_CI_lower = control_median_avg-3*control_median_sd

    control_cv_CI_upper = control_cv_avg+3*control_cv_sd
    control_cv_CI_lower = control_cv_avg-3*control_cv_sd

    control_stats_dict[protein]['avg_median'] = control_median_avg
    control_stats_dict[protein]['avg_cv'] = control_cv_avg
    control_stats_dict[protein]['99CI_median_upper'] =
control_median_CI_upper
    control_stats_dict[protein]['99CI_median_lower'] =
control_median_CI_lower
    control_stats_dict[protein]['99CI_cv_upper'] = control_cv_CI_upper
```

```
    control_stats_dict[protein]['99CI_cv_lower'] = control_cv_CI_lower

fold_change_dict = defaultdict(lambda: defaultdict(
lambda:defaultdict(float)))
significant_changes_dict = defaultdict(lambda: defaultdict(
lambda:defaultdict(lambda: defaultdict(float))))
number_significant_median = 0
number_significant_cv = 0
#for each protein, check if any medians or CVs fall outside of normal 95% CI
for protein in stats_dict:
    median_upper_CI = control_stats_dict[protein]['99CI_median_upper']
    median_lower_CI = control_stats_dict[protein]['99CI_median_lower']
    cv_upper_CI = control_stats_dict[protein]['99CI_cv_upper']
    cv_lower_CI = control_stats_dict[protein]['99CI_cv_lower']
    control_median = control_stats_dict[protein]['avg_median']
    control_cv = control_stats_dict[protein]['avg_cv']

    for condition in stats_dict[protein]:
        condition_median = stats_dict[protein][condition]['median']
        condition_cv = stats_dict[protein][condition]['cv']

        #calculate log2 fold change between non-stress and stress condition
        fold_median = np.log2(float(condition_median)/float(control_median))
        fold_cv = np.log2(float(condition_cv)/float(control_cv))
        fold_change_dict[protein][condition]['median'] = fold_median
        fold_change_dict[protein][condition]['cv'] = fold_cv

        #if distribution is above background/noise
        if condition_median > median_BY4741+std_BY4741:
            #if condition_median < median_lower_CI or condition_median >
median_upper_CI:
            if fold_median > 0.58 or fold_median < -0.58:
                #print 'median outside 95% CI:',protein, condition
                #print 'median before,after
is:',control_median,condition_median
                #print 'fold change median is:',fold_median
                #print '---'

significant_changes_dict[protein][condition]['median']['before'] =
control_median

significant_changes_dict[protein][condition]['median']['after'] =
condition_median

significant_changes_dict[protein][condition]['median']['fold'] = fold_median
                number_significant_median+=1
            #if condition_cv < cv_lower_CI or condition_cv > cv_upper_CI:
            if fold_cv > 0.58 or fold_cv < -0.58:
                #print 'CV outside 95% CI:',protein,condition
                #print 'CV before,after is:',control_cv,condition_cv
                #print 'fold change CV is:',fold_cv
                #print '---'
                significant_changes_dict[protein][condition]['cv']['before']
= control_cv
                significant_changes_dict[protein][condition]['cv']['after'] =
condition_cv
                significant_changes_dict[protein][condition]['cv']['fold'] =
```

```
fold_cv
                number_significant_cv+=1


print
json.dumps(significant_changes_dict,sort_keys=True,indent=4,separators=(',','
:'))
print 'number significant median changes:',number_significant_median
print 'number significant CV changes:',number_significant_cv


print 'making 2D array fold change'
#make graphical heat-map representation of fold changes
#make 2D array of rows = proteins, columns = conditions, value = fold-change

#dictionary of condition:column number
conditions_dict = {'Rapamycin':0,'H2O2':1,'Clotrimidazole':2,'MG-
132':3,'Cycoheximide':4,

'MMS':5,'Diamide':6,'ETOH':7,'HydroxyUrea':8,'37C':9,'DTT':10}
row = 0
fold_change_median_array =
np.zeros((len(fold_change_dict.keys()),len(conditions_dict.keys())))
fold_change_cv_array =
np.zeros((len(fold_change_dict.keys()),len(conditions_dict.keys())))

pro_list = []
cond_list = ['Rapamycin','H2O2','Clotrimidazole','MG-
132','Cycloheximide','MMS','Diamide','ETOH','Hydroxyurea','37C','DTT']
for protein in fold_change_dict:
    pro_list.append(protein)
    for condition in conditions_dict.keys():
        column = conditions_dict[condition]
        #print condition,column

        curr_fc_med = fold_change_dict[protein][condition]['median']
        curr_fc_cv = fold_change_dict[protein][condition]['cv']

        fold_change_median_array[row,column] = curr_fc_med
        fold_change_cv_array[row,column] = curr_fc_cv
    row+=1

#plot median fold changes

print cond_list
print pro_list

plt.figure(figsize=(12,12))
ax = plt.gca()
cmap =
mpl.colors.ListedColormap(['maroon','red','lightyellow','lightgreen','green']
)
bounds=[-4,-1,-0.5,0.5,1,4]
norm = mpl.colors.BoundaryNorm(bounds,cmap.N)
plt.imshow(fold_change_median_array,cmap=cmap,norm=norm,aspect='auto',interpo
lation='nearest')
plt.xticks(np.arange(0,len(conditions_dict.keys()),1),cond_list,rotation='ver
```

```
tical',fontsize=10)
ax.set_xticks(np.arange(0.5,len(conditions_dict.keys())+0.5,1),minor=True)
plt.yticks(np.arange(0,len(fold_change_dict.keys())),pro_list,fontsize=10)
ax.set_yticks(np.arange(0.5,len(fold_change_dict.keys())+0.5,1),minor=True)
plt.title('Protein Abundance',fontsize=18)
cb = plt.colorbar()
cb.set_label(label='Log2 Fold Change',fontsize=18)
plt.grid(True,which='minor',color='k',linestyle='solid',linewidth=2)
plt.savefig('/Users/Stefanie/PycharmProjects/GFPplots/median_fold_change.png'
,dpi=300)

#plot CV fold change
plt.figure(figsize=(12,12))
ax = plt.gca()
cmap =
mpl.colors.ListedColormap(['maroon','red','lightyellow','lightgreen','green']
)
bounds=[-4,-1,-0.5,0.5,1,4]
norm = mpl.colors.BoundaryNorm(bounds,cmap.N)
plt.imshow(fold_change_cv_array,cmap=cmap,norm=norm,aspect='auto',interpolati
on='nearest')
plt.xticks(np.arange(0,len(conditions_dict.keys()),1),cond_list,rotation='ver
tical',fontsize=10)
ax.set_xticks(np.arange(0.5,len(conditions_dict.keys())+0.5,1),minor=True)
plt.yticks(np.arange(0,len(fold_change_dict.keys())),pro_list,fontsize=10)
ax.set_yticks(np.arange(0.5,len(fold_change_dict.keys())+0.5,1),minor=True)
plt.title('Protein CV',fontsize=18)
cb = plt.colorbar()
cb.set_label(label='Log2 Fold Change',fontsize=18)
plt.grid(True,which='minor',color='k',linestyle='solid',linewidth=2)
plt.savefig('/Users/Stefanie/PycharmProjects/GFPplots/cv_fold_change.png',dpi
=300)
```