

TOOLS FOR MAXIMIZING THE EFFICIENCY OF PROTEIN ENGINEERING

A Thesis
Presented to
The Academic Faculty

by

Karen Marie Polizzi

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Chemical & Biomolecular Engineering

Georgia Institute of Technology
December 2005

TOOLS FOR MAXIMIZING THE EFFICIENCY OF PROTEIN ENGINEERING

Approved by:

Dr. Andreas S. Bommarius, Advisor
School of Chemical & Biomolecular
Engineering
Georgia Institute of Technology

Dr. Charles A. Eckert
School of Chemical & Biomolecular
Engineering
Georgia Institute of Technology

Dr. Christopher W. Jones
School of Chemical & Biomolecular
Engineering
Georgia Institute of Technology

Dr. David Lynn
Department of Chemistry
Emory University

Dr. Ichiro Matsumura
Department of Biochemistry
School of Medicine
Emory University

Date Approved: August 30, 2005

$\sin^2 \phi$ is odious to me

-Carl Friedrich Gauss

ACKNOWLEDGEMENTS

It is customary to begin the acknowledgements by thanking your advisor, and so, I would like to thank Andy who most of all taught me how to draw attention to the important parts of a sentence. I am also grateful to my thesis committee members: Ichiro Matsumura, Chuck Eckert, Chris Jones, and David Lynn for their time and patience. In particular, Ichiro was always looking out for my well being, suggesting projects and ideas, and contributing to my overall development as a scientist. I can't thank him enough for caring about me as if I was one of his own graduate students. Chuck has also been extremely helpful, trying to draw me out of my shell, offering words of encouragement when things were not working out as planned, and always advising the right course of action. I consider both Chuck and Ichiro my adopted advisors.

I would also like to thank the members of my lab group. Tracey Thaler had the burden of being the first student in the Bommarius laboratory and did a wonderful job setting everything up for us future students. She is also a great and caring friend both inside and outside of the laboratory and formerly (pre-Lily) my partner in crime. Eduardo Vazquez brings light and sunshine into my life everyday. He is a great counterpoint to my Cassandra. Bernard Loo (dude) is always willing to lend a helping hand. He also makes me laugh with his geeky jokes; great minds think alike. Javier Chaparro-Riggers is like a taller, male, older, German version of myself. He has been so supportive of me that I can never begin to repay him. James Broering, especially when he is at his most cynical, makes life in the lab more enjoyable for me. And, he is the most intelligent person I know. I would like to thank Phillip Gibbs for pointing out to me

that the technical support department of a company is a great resource and for always laying out ideas, but letting me choose my own course. I would also like to thank Rongrong Jiang for being a wonderful colleague. Finally, Jessica St. John, the very first undergraduate I advised, deserves my thanks for being the most wonderful, hard-working, and bubbly undergrad ever. I wish her luck at Michigan State.

The Doyle lab, just down the hall in Wing 3-D, has been a source of many wonderful and supportive friendships. Bahareh Azizi is a wonderful listener and gives great advice—even if I don't always follow it. Kenyetta Johnson is the most entertaining and sweet person I know. Priyanka Rohatgi (and her cellphone), Lauren Schwimmer, and Terry Watt (whose cynicism sometimes rivals James') are also deserving of my deepest thanks.

On a more serious note, the pooling work presented in Chapters III and IV would not have been possible without the hard work of many people. Here, I have to thank Ichiro once again for donating so many constructs and putting so much thought into the project. The Java programming was done by Cody Spencer. Anshul Dubey did the Matlab programming for the direct Poisson calculations. The fucosidase library was constructed by Monal Parikh of the Matsumura laboratory. I would also like to thank Phillip Gibbs, Jay Lee and Matthew Realff for their participation in the DE group, particularly, Matthew Realff for using dry, British humor to make the unbearable meetings more bearable. I would also like to thank Johnafel Crowe in the IBB Core Laboratory for doing the flow cytometry.

DNA sequencing is of vital importance to all of the projects we do. I would like to thank Brian Lynch in the School of Biology at Georgia Tech for always going out of

his way to provide sequences quickly and for trying different conditions for different problems. I would also like to thank Perry Mars of the Fundamental and Applied Molecular Evolution (FAME) sequencing center at Emory University for the hard work he has done with sequencing.

Finally, I would like to thank my non-GT friends and family. Without my parents' love and support there is no way I would be here (both literally and figuratively). Jonathan Liff, my former boss in the Epidemiology department never laughed at my career change and has supported me through trials and tribulations along the way. Laura Flowers, my great friend and neighbor, really knows her way around the kitchen and has kept me well fed for years. Last, but certainly not least, I would like to thank my fiancé, Jason Hallett, for absolutely everything. He encouraged me to come to Georgia Tech, supported me all the way through, and comforted me every single day. He has listened to a lifetime of woes over the past four years; I couldn't have done it without him.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	xiv
LIST OF FIGURES	xv
SUMMARY	xvii
 <u>CHAPTER</u>	
1 INTRODUCTION: TOOLS AND TRICKS TO ACCELERATE THE DEVELOPMENT OF NOVEL BIOCATALYSTS	5
2 BACKGROUND: A NEW PARADIGM FOR BIOCATALYSIS	6
2.1 Biocatalysis	6
2.1.1 Definition	6
2.1.2 Advantages	6
2.1.3 Disadvantages	7
2.1.4 Uses	7
2.1.4.1 Enzymes in Detergents	8
2.1.4.2 Enzymes in Textile Processing	8
2.1.4.3 Enzymes to Produce Food and Fragrance Compounds	9
2.1.4.4 Enzymes in the Cosmetic Industry	10
2.1.4.5 Enzymes in Pharmaceutical and Fine Chemical Industries	10
2.1.4.6 Enzymes in Commodity Scale Processes	13
2.1.4.7 Enzymes in the Petroleum Industry	13
2.2 Changing the Paradigm of Biocatalyst Process Development	14
2.3 Protein Engineering	15

2.3.1 Protein Engineering Definition	15
2.3.1.1 Rational Methods of Protein Engineering	16
2.3.1.2 Combinatorial Methods of Protein Engineering	16
2.3.2 Screening and Selection to Identify Improved Mutants	17
2.3.2.1 Advantages and Disadvantages of Selection	18
2.3.2.2 Advantages and Disadvantages of Screening	18
2.4 Summary	19
REFERENCES	20
3 CONSTRUCTION AND VALIDATION OF A MONTE-CARLO SIMULATION MODEL OF POOLNG	23
3.1 Introduction	23
3.2 Materials and Methods	26
3.2.1 Computer Simulations	26
3.2.2 Simulation Software	28
3.2.3 Verification of Simulation Results Through Analytical Probabilistic Model	29
3.2.4 Materials	30
3.2.5 Co-Culturing Experiments	31
3.2.6 Cell Growth and Assay Conditions	31
3.2.7 Cell Dilution Check with Coulter Counter	33
3.2.8 Mutant with 15-fold Difference in Activity Level	33
3.2.9 Standard Directed Evolution Activity Curve Experiments	34
3.3 Results	36
3.3.1 Co-Culturing Experiments	36
3.3.2 Flow Cytometry Check of Dilution	36
3.3.3 Simulations and Direct Calculations	37

3.3.4	Initial Experimental Validation	38
3.3.5	Forced Dilution Experiments	41
3.3.6	Experiments at Lower Activity Level Differences	43
3.3.7	Standard Activity Distribution	46
3.4	Discussion	47
3.4.1	Simulation Results	47
3.4.2	Model Validation	48
3.4.3	The Utility of Pooling in Directed Evolution Experiments	49
3.5	Summary and Conclusions	50
	REFERENCES	51
4	EXTENSION OF THE POOLING MODEL TO AN ACTUAL DIRECTED EVOLUTION LIBRARY	54
4.1	Introduction	54
4.2	Materials and Methods	55
4.2.1	Materials	55
4.2.2	Simulation Model	55
4.2.3	Experimental System	56
4.2.4	Model Inputs	56
4.2.5	Model Sensitivity Analysis	58
4.2.6	Library Construction Methods	59
4.2.7	Cell Growth, Induction, and Assay Procedures	59
4.2.8	Deconvolution of Winning Wells	60
4.2.9	Verification of Identity of “Winners”	60
4.3	Results	61
4.3.1	Sensitivity to Activity Distribution Input	61

4.3.2 Analysis of the Interdependence of Assay Accuracy and Activity Ratio	63
4.3.3 Threshold for Detection	65
4.3.4 Prediction of the Number of Supermutants Detected with Fucosidase Library	67
4.4 Discussion	70
4.4.1 Increase in the Number of Supermutants Detected Using Pooling	70
4.4.2 When is Pooling Useful?	71
4.4.3 Estimating the Parameters	72
4.4.3.1 Activity Distribution	72
4.4.3.2 Supermutant-to-Dud Ratio	72
4.4.3.3 Activity Ratio	72
4.4.3.4 Threshold for GREATER_THAN function	73
4.4.4 Extension of the Model to Other Disciplines	73
4.5 Conclusions	74
REFERENCES	75
5 DEVELOPMENT OF CLONING VECTORS TO AID IN HIGH-THROUGHPUT SCREENING	77
5.1 Introduction	77
5.2 Discussion of Vector Features	78
5.2.1 Basic Elements Necessary for Protein Expression	78
5.2.1.1 Origin of Replication	78
5.2.1.2 Promoter and Operator	78
5.2.1.3 Constitutive versus Inducible Expression	79
5.2.1.4 Ribosome Binding Site	80
5.2.1.5 Base Vector to Provide These Elements	80

5.2.2 Selection for Plasmid Retention	80
5.2.2.1 Antibiotic Resistance Genes	81
5.2.2.2 Growth Inhibitors	81
5.2.2.3 Triclosan	82
5.2.3 Mechanism of DNA Insertion	82
5.2.3.1 Ligation-Independent Cloning	83
5.2.4 Expression Reporters	83
5.2.5 Folding Reporters	84
5.2.5.1 Fluorescent Proteins	84
5.2.5.2 Antibiotic Resistance Markers	84
5.2.5.3 Intein-Mediated Fusions	85
5.3 Materials and Methods	85
5.3.1 Materials	85
5.3.2 Test of LIC Efficiency	86
5.3.3 Insertion of Triclosan into pPROTet	86
5.3.4 Design of LIC Inserts	87
5.3.5 Randomization of the lac promoter	87
5.3.6 Fluorescent Reporter Tests	87
5.3.7 Intein-Mediated Fusion Reporters	88
5.4 Results	89
5.4.1 Insertion of the Triclosan Gene into pPROTet	89
5.4.2 Test of LIC Efficiency	89
5.4.3 Design of LIC Inserts	89
5.4.4 Constitutive Expression by Randomization of the lac Operator	90
5.4.5 Fluorescent Reporter Tests	91

5.4.6 Intein-Mediated Fusion Reporters	93
5.5 Summary and Conclusions	94
REFERENCES	96
6 CLONING AND OVEREXPRESSION OF ENOLASE TEMPLATE ENZYMES	98
6.1 Introduction	98
6.1.1 α/β -barrel enzymes	98
6.1.2 Superfamilies	100
6.1.3 Enolase Superfamily	101
6.1.4 Enolase	102
6.2 Materials and Methods	104
6.2.1 Materials	104
6.2.2 Isolation of the Enolase Genes	105
6.2.3 Overexpression and Purification of the Enolase Enzymes	107
6.2.4 Synthesis of N-acetyl-(R)-Phenylglycine	107
6.2.5 Activity Assays	108
6.3 Results	109
6.3.1 Isolation of the Enolase Genes	109
6.3.2 Enolase Overexpression and Purification	110
6.3.3 N-acetyl-(R)-Phenylglycine Synthesis	112
6.3.4 Activity Assays	113
6.4 Discussion	113
REFERENCES	115
7 CONCLUSIONS AND FUTURE WORK	118
REFERENCES	123

LIST OF TABLES

	Page
TABLE 4.1 Sequences of mutants identified in the 384-well plate screen	68
TABLE 6.1 Summary of mutations in the various enolase enzymes	110
TABLE 6.2 Activity tests of purified enolases	113

LIST OF FIGURES

	Page
FIGURE 2.1 A comparison of conventional biocatalyst process development and the ideal development	15
FIGURE 3.1 Finding the optimal level of pooling	24
FIGURE 3.2 The substrates of β -galactosidase and β -glucuronidase	26
FIGURE 3.3 Pooling simulation model flowsheet	27
FIGURE 3.4 The substrates <i>ortho</i> -nitrophenylgalactopyranoside and resorufin-b-D-galactopyranoside	33
FIGURE 3.5 Standard activity distribution from a directed evolution experiment	35
FIGURE 3.6 Results of the co-culturing experiment	36
FIGURE 3.7 Check of cell dilution procedures with flow cytometry	37
FIGURE 3.8 Example simulation data	38
FIGURE 3.9 Comparison of simulated and experimental results	40
FIGURE 3.10 Assay of artificial pools of cells expressing β -glucuronidase with cells expressing β -galactosidase	42
FIGURE 3.11 Full color photos of the assay results	43
FIGURE 3.12 Assay of artificial pools of cells expressing β -glucuronidase with cells expressing the three mutants	44
FIGURE 3.13 Comparison of simulation and experimental data for pooling with mutants	46
FIGURE 3.14 Predicted and actual numbers of AR=15 mutants detected when pooling with a more typical activity distribution	47
FIGURE 4.1 The activity curve derived from a single 384-well plate screen	58
FIGURE 4.2 Analysis of dependence of model on the shape of the activity distribution	63

FIGURE 4.3	Predicted number of supermutants at varying activity ration and coefficient of variation	64
FIGURE 4.4	The influence of the threshold for detection in the GREATER_THAN function	66
FIGURE 4.5	A comparison of the predicted and experimentally obtained number of supermutants	69
FIGURE 5.1	Intein-mediated fusion reporter construct	88
FIGURE 5.2	Designed LIC site allowing for C-terminal His-tag option	90
FIGURE 5.3	The second designed LIC site without His-tag	90
FIGURE 5.4	Results of the variable expression by randomization of the lac operator region	91
FIGURE 5.5	Fluorescent reporter tests	92
FIGURE 5.6	Diagnostic PCR of intein fusion constructs	93
FIGURE 6.1	The structure of an α/β -barrel enzyme	99
FIGURE 6.2	The reaction catalyzed by enolase	103
FIGURE 6.3	N-acetyl-(R)-phenylglycine	108
FIGURE 6.4	Initial amplification of the enolase genes	109
FIGURE 6.5	Expression of the <i>S.cerevisiae</i> and <i>E. coli</i> enolases	110
FIGURE 6.6	Expression of the <i>L. lactis</i> enolase	111
FIGURE 6.7	Expression of the <i>L. plantarum</i> enolase	112

SUMMARY

Biocatalysts offer advantages over their chemical counterparts in terms of their high enantioselectivity and the opportunity to develop more environmentally friendly processes. However, the widespread adoption of biocatalytic processes is hampered by the long development times for enzymes with novel and sufficient activity and adequate stability under operating conditions. Protein engineering, while extremely useful for modifying the properties of protein catalysts in select cases, still cannot be performed rapidly enough for many applications. In order for biocatalysts to become a competitive alternative to chemical catalysts, new tools to make the tailoring of biocatalysts by protein engineering methods speedier and more efficient are necessary. The aim of this work was to develop methods to aid in the faster production of novel biocatalysts.

Protein engineering involves two steps: the generation of diversity and the screening or selection of variants with the desired properties. Both of these must be targeted to create a faster protein engineering process. In the case of the former, this work sought to clone and overexpress some template enzymes which would create smaller, more manageable libraries of mutants with a higher likelihood of function by the manipulation of a few focused amino acid residues. For the latter, this work developed and validated a Monte-Carlo simulation model of pooling to increase screening throughput and created a set of vectors to aid in high-throughput screening by eliminating unwanted mutants from the assay procedure entirely.

CHAPTER I

INTRODUCTION: TOOLS AND TRICKS TO ACCELERATE THE DEVELOPMENT OF NOVEL BIOCATALYSTS

Biocatalysis, or the use of whole cells or enzymes to perform chemical reactions, is becoming increasingly useful on a large scale. Processes utilizing biocatalysts are often more environmentally friendly than the chemically catalyzed alternatives and the selectivity of biocatalysts (both regioselectivity and enantioselectivity) leads to a reduction in both starting material required and waste produced. However, biocatalysts were evolved to work in the cellular environment; aqueous medium at low temperatures and near neutral pH, and these conditions may not be identical to the conditions necessary to carry out the desired reaction on a large scale. Thus, the field of protein engineering has emerged to improve the properties of biocatalysts to become more compatible with operating conditions.

Protein engineering involves both rational and combinatorial methods of changing a protein sequence to achieve a desired result, such as a change in substrate specificity, or increased stability to temperature, organic solvents, and/or extremes of pH. Rational methods, such as site-directed mutagenesis, require targeted amino acid substitutions, and therefore, require a large body of knowledge about the biocatalyst being improved, including the three-dimensional structure and the chemical mechanism of the reaction.

The main advantage to rational design is that a very small number of protein variants are created, meaning that very little effort is necessary to screen for improved properties. Combinatorial methods, on the other hand, create a large number of variants that must be assayed, however they have the advantage of not requiring such extensive knowledge about the protein. In addition, often times non-obvious changes in the protein sequence will lead to large improvements in their properties, these are extremely hard to predict rationally, and thus, can only be identified by combinatorial methods.

Although protein engineering, particularly via combinatorial methods, has become a useful tool to improve the utility of biocatalysts, one of the challenges hampering the widespread implementation of biocatalytic processes is the development time necessary to tailor the biocatalyst to the operating conditions. Many of the biocatalytic processes currently in operation, such as the biological route to acrylamide and similar processes (discussed briefly in Chapter II) took decades to develop. Such a large resource investment, coupled with the fact that the lifetime of patent protection is only about as long as the development time, has hindered development of biocatalytic processes for many industrially relevant problems. The work presented here seeks to decrease the development times for new biocatalysts by accelerating the process of protein engineering. In the future, these tools can be used to create biocatalytic routes to products formerly inaccessible by biological routes and will increase the efficiency of protein engineering efforts.

Combinatorial methods of protein engineering involve two main steps, the generation of diversity, followed by the screening or selection of the variants created to identify which have the improved properties of interest. Chapters III through V focus on

shortening the development time of new biocatalysts by focusing on improving the identification of improved mutants. The screening process involves the individual assay of each mutant created, and as a consequence, is very resource- and time-intensive. Chapters III and IV deal with the development and experimental validation of a Monte-Carlo simulation model of pooling to increase the throughput in screening assays. Pooling, a concept used in the screening of combinatorial chemical libraries, is a two-stage process. In the first stage, multiple cells are combined into a single assay well, forming a pool. The best pools are chosen for the second stage where they are separated into individual cell types and assayed to extract the improved mutant. Pooling for combinatorial protein engineering problems depends on several factors including the frequency with which good mutants are created, the increase in activity level of the most highly improved mutants, the level of accuracy of the assay, and the distribution of activity levels in the population. Chapter III focuses on the initial validation of the model using highly controlled systems designed to test the simulations at various levels of the factors mentioned previously by mixing known entities with varying levels of activity towards the hydrolysis of substrates containing a galactoside moiety. The first system tested was a binary mixture of a very highly active mutant (the "supermutant") with an ancestral cell (the "dud"). Because the difference in activity level of the supermutant and dud was far larger than the error in assay detection we were able to provide additional validation by using the Poisson distribution to directly calculate the number of active mutants for this case. We then moved to a more realistic case where the population remained binary, but the difference in activity level was much lower, more realistic for a protein engineering experiment. Finally, we tested the performance of the model on a

simulated activity curve using cells with activity level ranging from zero to 15-fold over the ancestral cell.

Following the initial validation of the Monte Carlo model in Chapter III, Chapter IV extends the prediction to an actual directed evolution library, created by randomization of the active site in three amino acid residues. Here, we also use a system that requires induction of protein expression (rather than the constitutively expressing system in Chapter III), a situation more common to protein engineering projects. The use of pooling in this more realistic example increases the number of good mutants detected, and the simulation model accurately predicts the number of mutants expected. Pooling is a valid method to increase the throughput of high throughput screening assays.

The process of analyzing the protein variants for the activity of interest requires the transformation of cells with the DNA of interest usually carried on a plasmid (vector), which also serves as a means to express the protein. Chapter V discusses the creation of vectors designed to aid protein engineering efforts by combining state-of-the-art features that have been shown in the literature to increase the efficiency of high throughput screens and selections. Among these is the use of reporter proteins to indicate which mutants are successfully expressed as well as to provide a basis for normalization by expression levels, the use of ligation-independent cloning to insert the gene of interest with high efficiency and regardless of the restriction pattern of the gene, the use of the bacterial inhibitor triclosan to provide stringent selection for plasmid retention, and protein expression controlled by the Tn(10) tet promoter/operator which is orthogonal to the metabolism of the traditional *E. coli* expression strains. The combination of these

features into one vector allows for the optimal expression and screening of protein variants in a high throughput manner.

The focus of Chapter VI is to increase the efficiency of protein engineering experiments by impacting the diversity generation steps. We introduce the concept of the α/β -barrel protein as a generic scaffold for enzyme evolution and present the cloning and overexpression of enolase from three organisms as a template for further evolution. The α/β -barrel is a ubiquitous fold found in nature, consisting of eight alternating alpha helices and beta strands which pack to form an inner core of beta strands surrounded by an outer core of alpha helices. All catalytic residues are found in the loop regions. This fold is especially suited as a scaffold template because a large portion of the amino acid residues can be eliminated from diversity generation because they form the secondary structure of the enzyme and, in addition, the loop regions which are responsible for catalysis are localized at the C-terminal end of the barrel. Thus, protein engineering efforts can be focused in one particular region, which increases the likelihood for activity in each individual mutant, as well as limits the number of mutants that must be tested.

CHAPTER II

BACKGROUND: A NEW PARADIGM FOR BIOCATALYSIS

2.1 Biocatalysis

2.1.1 Definition

Biocatalysis, the use of whole cells or isolated enzymes for chemical transformations, has been employed for thousands of years, beginning with early civilizations which used yeast to leaven bread and ferment beverages. During the past few decades the industrial use of ‘white biotechnology’ is increasing substantially and the trend is expected to continue.²

2.1.2 Advantages

The biggest advantage of biocatalysts involves the synthesis of products containing chiral centers. Here, biocatalysts not only have an advantage in that they are highly regio- and stereo- selective, but they also avoid the need of multiple protection and deprotection steps commonly found in organic syntheses and are less likely to produce unwanted side products than traditional chemical catalysts. In particular, the elimination of the protection/deprotection steps increases the atom efficiency of a process and decreases the amount of waste generated.³ An additional attractive feature of enzyme catalysts is that they work under very gentle conditions (aqueous media at ambient temperature); however, this is not always feasible for the process.³⁻⁵ Finally, with the

cost of drug development skyrocketing and an FDA mandate that each enantiomer of a compound be tested for safety and efficacy, biocatalysts are becoming increasingly important to the pharmaceutical and agrochemical industries.⁶

2.1.3 Disadvantages

The main disadvantages of biocatalysts stem from the natural evolution of their activity. Enzymes have evolved to work in the cellular environment, and therefore are not usually tolerant to the presence of organic solvents, extremes of pH, or extremes of temperature. Also, because of regulation mechanisms within cells, they may be subject to substrate or product inhibition. Additionally, they have evolved to catalyze reactions which are useful to living organisms—these may not be the same reactions or substrates that are industrially useful.³⁻⁵

2.1.4 Uses

Biocatalysis is being used increasingly in various industrial applications, ranging from adding pure enzyme to a product, as is the case with detergents, or using whole cells to do complex syntheses in the pharmaceutical and fine chemical industry. Overall, the biggest successes have come where the required enzyme is easily obtainable from the cell lysate, easily immobilized, catalyzes a reaction which does not require a cofactor, and the reaction conditions for the process are similar to the reaction conditions *in vivo*.⁵

In the manufacturing arena, biocatalysts are used extensively in the textile industry, for leather processing, and as additives to detergents.⁷ The majority of these enzymes are hydrolases, hydrolyzing the bonds of proteins, lipids, and starches for various purposes. In a few instances, oxidoreductases are also being employed, primarily as bleaching agents. The main reason for the use of enzymes in these manufacturing

areas is their ease of use (e.g. in the leather industry the hide can be simultaneously softened, degreased, dehaired, and rehydrated in a single step) and the reduction of waste from chemical processes (e.g. enzymatic removal of excess dyes in the textile industry reduces the water usage associated with repeated rinsing).

2.1.4.1 Enzymes in Detergents

The addition of enzymes to detergent formulations has been in practice since the 1960s. Here, the function of the enzymes is to degrade the insoluble, usually polymeric, stains on clothing to smaller, more water-soluble monomers that can be removed into the washing water. The most prevalent and first enzyme group to be added to detergents is the protease, which hydrolyzes proteinaceous stains, although increasingly lipases (which degrade fats), cellulases (for grass stains), and amylases (which degrade starches) are also being used. The primary challenge in the use of enzymes as detergent additives is their stability in the washing environment which includes a fairly alkaline pH, the presence of bleaching agents and chelators, and strong surfactants.⁷ Therefore, a lot of time and effort has gone into engineering more stable versions of these enzymes with improved washing performance.

2.1.4.2 Enzymes in Textile Processing

The most ubiquitous enzyme in the textile industry is cellulase, which hydrolyzes the endo, β - linkages of glucose that form the cellulose polymer. The primary focus is to improve the characteristics of the cloth by polishing stray cotton fibers to decrease pilling, increase the softness and flexibility, and to prepare the cloth to receive dyes. An additional use of cellulases is in enzymatic stone-washing, whereby fibers of the denim cloth that has already been dyed with indigo are selectively removed to create the mottled

finish characteristic of stone-washed jeans. In 80% of cases, this process has replaced the original method of stone-washing that involved laundering denim fabric with pumice stones. One other use of enzymes in the textile industry is the removal of excess dye and bleaching agents by peroxidases and catalases, respectively.⁷

In the tanning industry, enzymes are used to degrease and soften the hides to make them ready for the tanning process. Degreasing is performed by lipases, whereas softening, dehairing, and pickling are done by proteases. These are used in conjunction to efficiently process the leather in a minimum number of steps.⁷

2.1.4.3 Enzymes to Produce Food and Fragrance Compounds

Many food and fragrance compounds are synthesized biocatalytically. Due in part to the ‘chemophobic’ nature of the public when it comes to articles associated with their food source, naturally derived flavor compounds are sold for substantially higher prices (up to two orders of magnitude compared to their chemically synthesized counterparts).⁸ However, their extraction from natural sources is time-consuming and expensive.⁹ One advantage of biocatalytic routes to these compounds stems from the definition of the word ‘natural’ in the regulation of food products by the United States and the European Union which allows for the labeling of a product as natural as long as it is made via natural routes, regardless of whether this takes place in a living organism or in a reactor. Additionally, biocatalytic processes are highly regio- and stereo- selective and can be run under fairly mild conditions to protect the often unstable compounds from degradation, all of which is important for maintaining their flavoring properties.⁸ While mixed microbial cultures have the tendency to promote spoilage and undesirable off-flavors, many successful processes have been developed using bacterial monoculture and

purified enzymes.⁹ Compounds such as vanillin⁸⁻¹⁰, γ -decalactone (peach)⁸, and raspberry ketone⁹ are all accessible via biocatalytic routes. A major continuing challenge in the use of biocatalysis for the production of natural products is the time and effort necessary for the discovery of natural pathways of production and the recombinant production of the enzymes involved.^{8, 9} Future possibilities for growth in this industry involve the discovery of novel flavor compounds by the biocatalytic generation of combinatorial libraries of functionalized natural products.¹¹

2.1.4.4 Enzymes in the Cosmetic Industry

Biocatalysts are also being employed in the cosmetic industry in the synthesis of emollients, emulsifiers, and thickening agents. Here, the main advantage is that the use of enzymes can vastly reduce the number of steps required to make the product. In addition, minimizing the use of organic solvents is desirable because residual solvent causes an undesired skin drying effect. The main disadvantage to enzymatic routes is their cost effectiveness; although this can be improved via catalyst recycle. One example of a competitive process is the production of cetyl ricinoleate via esterification of cetyl alcohol with ricinolic acid using an immobilized lipase as the catalys.¹²

2.1.4.5 Enzymes in the Pharmaceutical and Fine Chemical Industries

Enzymes have found extensive use in the chemical industry. For example, there are numerous processes in pharmaceutical and fine chemical industries that utilize biocatalysts. The compounds used in these industries must be of high regio- and optical purity, thus the high selectivity of biocatalysts is their main advantage. A wide range of biocatalysts are employed for both direct synthesis and the resolution of racemic mixtures.^{10, 13-16} Examples of industrially viable processes using biocatalysts include the

production of α - and β - amino acids, enantiomerically pure amines and alcohols, regioselective reactions involving carbohydrates, and many highly complex pharmaceutical intermediates.

Biocatalytic process development in the pharmaceutical industry faces particular challenges with regard to throughput and downstream processing, based on the particular phase of trial that the compound is currently in. In early phases, such as compound discovery and lead optimization, synthesizing the compound quickly is the most important factor, and the process productivity and cost are largely ignored. It is also important, in the early stages, to be able to synthesize both enantiomers of chiral molecules to test their effects independently. As the drug molecule moves closer to the market, emphasis shifts to developing a scalable, cost-effective process. Downstream processing issues become important, particularly in determining the competitiveness of the biocatalytic process compared to alternate chemical synthesis routes because the cost and ease of separation impact the overall economic viability of the synthetic route ultimately chosen.¹⁷

With regard to throughput, to be competitive a process utilizing a biocatalyst must have a high volumetric productivity. Therefore, the enzyme must be highly active and not subject to substrate inhibition at high concentrations of reactant.^{17, 18} The ability to react with less expensive starting materials is also desired.¹⁷

The main challenge in the downstream processing of products of enzymatic reactions is the difficulty with extractive recovery of the products. When organic products are being extracted from a mostly aqueous reaction medium into an organic layer, the whole cells or isolated enzymes tend to aggregate into an emulsion which is not

easily broken. Some processes use the addition of de-emulsifying agents, adsorption, or other techniques to circumvent this problem; however, there are many instances where this tendency to emulsify will render a process infeasible. The ability to use immobilized enzymes or whole cells has some benefit in this instance as filtration can be used to recover the catalyst pellets.¹⁷

Further challenges will emerge as renewable feedstocks (e.g. corn stover) are used as precursors for synthesis. Most research is conducted on clean samples and assumes idealized kinetics. In reality, many of these crude substrates are complex mixtures containing inhibitors which impede the desired reaction, but allow side reactions to continue.¹⁹

In particular, the high degree of complexity of pharmaceuticals with a large number of functional groups and chiral centers requires catalysts that can be highly selective without the need of protection and deprotection steps that add to cost and waste generation.¹³ Biocatalysts offer an advantage in the case of highly complex syntheses. Here, it is desirable to reduce number of steps to product formation, replace difficult steps with more simple ones, improve optical purity, to improve reaction conditions to eliminate potentially unsafe steps, and to reduce waste.²⁰ An example is the production of penicillin-derived antibiotics by various β -lactam acylases.²¹ In this case, the complex antibiotic structure is created in two steps. First, the component 6-aminopenicillanic acid is obtained via fermentation and the enzymatic hydrolysis of penicillin G. This core structure is then reacted with a side chain compound, which is added via acylation. The chemical synthesis of penicillin-like structures is far more complex, with higher energy

consumption (taking place at -40 °C) and generating much more waste.²² Thus, the biocatalytic method has gained favor.

2.1.4.6 Enzymes in Commodity Scale Processes

The use of biocatalysts in the chemical industry is not limited to the pharmaceutical and fine chemical industries, but has also been used on the bulk chemical scale. For instance, high fructose corn syrup, a sweetener present in sodas and many other products is produced by the enzymatic hydrolysis of starch followed by the isomerization of glucose to fructose by using another enzyme, glucose isomerase. More than one million tons of high fructose corn syrup are produced annually by this biocatalytic process.²³ In addition, acrylamide, the monomer unit of a mass produced, large volume, commodity polymer is produced on the 11 million ton scale using the nitrilases from various *Rhodococcus* species.²⁴

2.1.4.7 Enzymes in the Petroleum Industry

One industry which may benefit from the implementation of biocatalysis in the future is the petroleum industry. In the past, the main focus was on microbiologically enhanced oil recovery, treatment of waste streams, and bioremediation of contaminated soils. However, new research toward the biologically-based desulfurization, denitrogenation, and demetallation of fuels and the lightening of crude oil via whole cell biocatalysts is increasing. The main challenge facing the implementation of biocatalysts for these purposes is the enhancement of their activity and stability in the highly hydrophobic, complex environments characteristic of crude oils.²⁵

2.2 Changing the Paradigm of Biocatalyst Process Development

Current methods of process development for biocatalysts usually attempt to circumvent the weaknesses of the biocatalyst by changing the process conditions. For example, if the reactants are sparingly soluble in water, but the enzyme is not tolerant to high levels of organic solvents, the reactor volume will be increased. This results in a compromised process with low volumetric productivity which is usually not economically viable. A better solution would be to determine the optimal processing parameters based on the particular constraints of the *reaction*. Here, the solubility of the reactants and products must be considered in choosing a reaction medium: their stability with respect to temperature and pH will, in part, dictate process operating conditions; and the equilibrium of the underlying reaction may suggest an operating temperature. Additionally, the type of reactor must be considered. If an immobilized catalyst is used, there may be diffusion effects. If a plug flow reactor is packed with catalyst beads, there might be concentration gradients—therefore a catalyst with a lack of substrate and product inhibition and a low K_m (substrate concentration to obtain a rate one-half of the maximum) is preferred. Once these parameters are set, a biocatalyst can be engineered to work within them. The shift from changing the process to suit the catalyst to improving the catalyst to work in the best process represents a new paradigm in biocatalyst development. Figure 2.1 illustrates the difference between conventional and ideal process development.

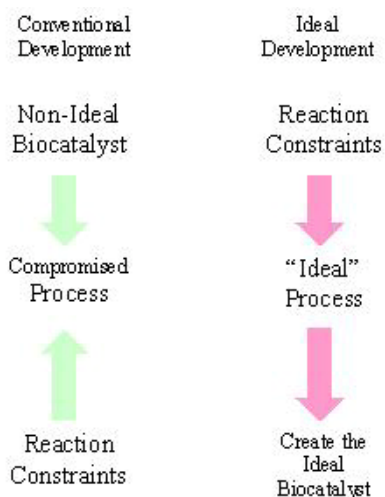


Figure 2.1: A comparison of conventional biocatalyst process development and the ideal development.¹

As discussed above, the current challenges facing the widespread adoption of biocatalytic processes center on their instability in certain situations, namely at high temperature and in the presence of organic solvents. Additionally, there is not an existing biocatalyst for every desired reaction.⁴ Both of these problems can be alleviated by improving the stability or changing the substrate specificity of the biocatalyst via protein engineering (for examples, see ^{5, 26-29}).

2.3 Protein Engineering

2.3.1 Protein Engineering Definition

Protein engineering is the creation of new or improved enzymes or proteins by altering an enzyme or a protein with inadequate properties. Many specific methods for protein engineering exist, but they can be grouped into two major categories: those involving rational design of protein changes and combinatorial methods which make changes in a more random fashion.

2.3.1.1 Rational Methods of Protein Engineering

Rational methods of protein engineering involve the selection of particular amino acid residue to target for alteration to achieve the desired outcome based on initial knowledge of the protein structure, function, and chemical mechanism. Usually, a limited number of variants are explored and the residues that are targeted are often those involved in contact with the substrate. Changes can be made via site-directed mutagenesis, in which the investigator selects both which amino acids to change, and what specific changes should be made.³⁰ A related method is site-saturation mutagenesis, where specific amino acid residues are targeted, but all 20 amino acids are tried at each location.³¹ More recently, the complete *de novo* design of protein backbones based on the combination of various optimization algorithms with protein predictive force fields has been applied,³² though this field is still in its infancy. As more knowledge is gained as to how the protein structure affects its function, rational design of proteins with particular functions will become easier to perform.

2.3.1.2 Combinatorial Methods of Protein Engineering

Combinatorial methods of protein engineering can be further subdivided into methods that generate point mutations, insertions, or deletions and methods involving recombination of one or more parental genes. The former includes error-prone polymerase chain reaction (PCR), a process by which errors are introduced into the growing DNA chain during normal replication. This can be accomplished by exploiting the natural errors produced by Taq DNA polymerase in the presence of manganese and unbalanced concentrations of dNTPs, using a polymerase which has been modified to increase its error rate (such as Mutazyme® from Stratagene), or by including a universal

base such as inosine that hydrogen bonds with any of the normal nucleotides.^{33, 34} Another method for introducing random errors into the gene is the use of a mutator strain of *E. coli* in the transformation step, which has been engineered for decreased fidelity of replication.³⁵

There are a very large number of recombination protocols available in the literature. Some, such as DNA shuffling and RACHITT (Random Chimeragenesis on Transient Templates), rely on annealing of mixed DNA fragments from different parents followed by replication of the newly constructed chimeras. Others, like the Staggered Extension Process (StEP) and RDA-PCR (Recombination-Dependent Exponential Amplification Polymerase Chain Reaction) are based on PCR with short extension time and template switching between parents to create diversity.³³⁻³⁵ Both the annealing and PCR- based methods require a certain degree of sequence homology for recombination to take place.³⁵ For a third group of methods, including ITCHY (incremental truncation for the creation of hybrid enzymes) and sequence-independent site-directed chimeragenesis (SISC), this is not a requirement. These methods rely on the directing of crossovers via ligation-mediated assembly, and thus are not bounded by the homology of the sequence in question. One drawback, however, is that the number of crossovers obtained via these methods is limited.^{33, 34}

2.3.2 Screening and Selection to Identify Improved Mutants

Following diversity generation, each member of the library must be evaluated to determine whether it has the properties of interest. There are two main ways of achieving this, tying the survival of the host organism to the function of the protein of interest (selection), or individually assaying each protein (screening).

2.3.2.1 Advantages and Disadvantages of Selection

Selection involves transforming the library into the host cell and then plating the transformants onto some sort of selective medium which only allows survival of the organism if the protein is present. It is much less time-intensive, allowing for the analysis of 10^9 - 10^{11} colonies per round of evolution. Additionally, selection is very sensitive—less than 1% activity over background is required for organism survival. However, due to the adaptive nature of the host organism, selection can sometimes result in false positives (i.e. the bacterium changes in ways not related to the evolved protein to prevent its death) and it merely provides a yes/no answer—catalytic rates must still be measured in a separate experiment. Finally, not all activities of interest can be tied to host cell survival.³⁶

2.3.2.2 Advantages and Disadvantages of Screening

Screening entails analyzing each colony for its level of the desired reaction. Screening involves three steps: the separation of individual colonies into their own assay compartment, the generation of signal (reaction), and the detection of the signal (analysis).³⁷ The process is much more resource- and time- intensive, so only about 10^5 - 10^7 colonies can be screened in each round of evolution.^{36, 37} The capacity limitation means that only a fraction of the generated library is screened in any round, making it likely that some highly active mutants may be missed.³⁷

One problem with current directed evolution experiments is that high throughput screening conditions do not mirror the way in which the final enzyme will be used industrially because they take place in the growth medium in the presence of cell lysate,

rather than with isolated enzyme which may be immobilized and will be subject to recycle.³

2.4 Summary

In summary, biocatalysis is gaining favor in a variety of industries due to its selective nature and the potential for more environmentally favorable processing conditions. The main challenge to developing economically viable biocatalytic processes is the lack of stability of these catalysts to many conditions such as the presence of organic solvents or extremes of temperature, which are often necessary for large-scale production. Protein engineering is a valuable tool to improve these properties, and therefore the usefulness of biocatalysts. In the future, the tailoring of biocatalyst properties to the process at hand (rather than the reverse) will make biocatalytic processes more competitive alternatives for many chemical applications.

REFERENCES

1. Burton, S. G.; Cowan, D. A.; Woodley, J. M., The search for the ideal biocatalyst. *Nature Biotechnology* **2002**, 20, 37-45.
2. Jaeger, K.-E., Protein technologies and commercial enzymes White is the hype-biocatalysts on the move. *Current Opinion in Biotechnology* **2004**, 15, 269-271.
3. Lye, G. J.; Dalby, P. A.; Woodley, J. M., Better biocatalytic processes faster: new tools for the implementation of biocatalysis in organic synthesis. *Organic Process Research and Development* **2002**, 6, 434-440.
4. Bommarius, A. S.; Riebel, B. R., *Biocatalysis: Fundamentals and applications*. Wiley VCH: Weinheim, Germany, 2004; p 611.
5. Powell, K. A.; Ramer, S. W.; del Cardayre, S. B.; Stemmer, W. P.; Tobin, M. B.; Longchamp, P. F.; Huisman, G. W., Directed evolution and biocatalysis. *Angewandte Chemie International Edition* **2001**, 40, 3948-3959.
6. <http://www.fda.gov/cder/guidance/stereo.htm>.
7. Galante, Y. M.; Formantici, C., Enzyme applications in detergency and in manufacturing industries. *Current Organic Chemistry* **2003**, 7, 1399-1422.
8. Schrader, J.; Etchmann, M.; Sell, D.; Hilmer, J.; Rabenhorst, J., Applied biocatalysis for the synthesis of natural flavour compounds- current industrial processes and future prospects. *Biotechnology Letters* **2004**, 26, 463-472.
9. Vandamme, E. J.; Soetaert, W., Bioflavours and fragrances via fermentation and biocatalysis. *Journal of Chemical Technology and Biotechnology* **2002**, 77, 1323-1332.
10. Liese, A.; Filho, M. V., Production of fine chemicals using biocatalysis. *Current Opinion in Biotechnology* **1999**, 10, 595-603.
11. Khan, J. A.; Vulfson, E. N., Combinatorial chemistry in food research. *Combinatorial Chemistry and High Throughput Screening* **2003**, 6, 569-574.
12. Veit, T., Biocatalysis for the production of cosmetic ingredients. *Engineering and Life Sciences* **2004**, 4, (6), 508-511.
13. Panke, S.; Wubbolts, M., Advances in biocatalytic synthesis of pharmaceutical intermediates. *Current Opinion in Chemical Biology* **2005**, 9, 188-194.

14. Rasor, J. P.; Voss, E., Enzyme-catalyzed processes in pharmaceutical industry. *Applied Catalysis A: General* **2001**, 221, 145-158.
15. Schulze, B.; Wubbolts, M., Biocatalysis for industrial production of fine chemicals. *Current Opinion in Biotechnology* **1999**, 10, 609-615.
16. Panke, S.; Held, M.; Wubbolts, M., Trends and innovations in industrial biocatalysis for the production of fine chemicals. *Current Opinion in Biotechnology* **2004**, 15, 272-279.
17. Yazbeck, D. R.; Martinez, C. A.; Hu, S.; Tao, J., Challenges in the development of an efficient enzymatic process in the pharmaceutical industry. *Tetrahedron: Asymmetry* **2004**, 15, 2757-2763.
18. Ikunaka, M., Biocatalysis from the perspective of an industrial practitioner: let a biocatalyst do a job that no chemocatalyst can. *Catalysis Today* **2004**, 96, 93-102.
19. Wiseman, A., "Biosustainability-limit" to industrial biocatalysis: if no supersedence of bioprocess-perturbation by resilience? *Enzyme and Microbial Technology* **2003**, 32, 635-637.
20. Bhattacharya, S. K., The use of enzyme mixtures for complex biosyntheses. *Current Opinion in Biotechnology* **2004**, 15, 449-455.
21. Sio, C. F.; Quax, W. J., Improved b-lactam acylases and their use as industrial biocatalysts. *Current Opinion in Biotechnology* **2004**, 15, 349-355.
22. Sheldon, R. A.; van Rantwijk, F., Biocatalysis for sustainable organic synthesis. *Australian Journal of Chemistry* **2004**, 57, 281-289.
23. Crabb, W. D.; Shetty, J. K., Commodity scale production of sugars from starches. *Current Opinion in Microbiology* **1999**, 2, 252-256.
24. Hughes, J.; Armitage, Y. C.; Symes, K. C., Application of whole cell rhodococcal biocatalysts in acrylic polymer manufacture. *Antonie van Leeuwenhoek* **1998**, 74, 107-118.
25. Le Borgne, S.; Quintero, R., Biotechnological processes for the refining of petroleum. *Fuel Processing Technology* **2003**, 81, 155-169.
26. Bornscheuer, U. T.; Pohl, M., Improved biocatalysts by directed evolution and rational protein design. *Current Opinion in Chemical Biology* **2001**, 5, 137-143.
27. Jaeger, K.-E.; Eggert, T., Enantioselective biocatalysis optimized by directed evolution. *Current Opinion in Biotechnology* **2004**, 15, 305-313.

28. Cherry, J. R.; Fidantsef, A. L., Directed evolution of industrial enzymes: an update. *Current Opinion in Biotechnology* **2003**, 14, 438-443.
29. Zhao, H.; Chockalingham, K.; Chen, Z., Directed evolution of enzymes and pathways for industrial biocatalysis. *Current Opinion in Biotechnology* **2002**, 13, 104-110.
30. Costa, G.; Bauer, J.; McGowan, B.; Angert, M.; Weiner, M., Site-directed mutagenesis using a rapid PCR-based method. *Methods in Molecular Biology* **1996**, 57, 239-48.
31. O'Donohue, M. J.; Kneale, G. G., A method for introducing site-specific mutations using oligonucleotide primers and its application to site-saturation mutagenesis. *Molecular Biotechnology* **1996**, 6, (2), 179-189.
32. Bolon, D. N.; Voigt, C. A.; Mayo, S. L., De novo design of biocatalysts. *Current Opinion in Chemical Biology* **2002**, 6, 125-129.
33. Lutz, S.; Patrick, W. M., Novel methods for directed evolution of enzymes: quality, not quantity. *Current Opinion in Biotechnology* **2004**, 15, 291-297.
34. Neylon, C., Chemical and biochemical strategies for the randomization of protein encoding DNA sequences: library construction methods for directed evolution. *Nucleic Acids Research* **2004**, 32, (4), 1448-1459.
35. Farinas, E. T.; Bulter, T.; Arnold, F. H., Directed enzyme evolution. *Current Opinion in Biotechnology* **2001**, 12, 545-551.
36. Steipe, B., Evolutionary Approaches to Protein Engineering. *Current Topics in Microbiology and Immunology* **1999**, 243, 55-86.
37. Cohen, N.; Abramov, S.; Dror, Y.; Freeman, A., In vitro enzyme evolution: the screening challenge of isolating the one in a million. *Trends in Biotechnology* **2001**, 19, (12), 507-510.

CHAPTER III

CONSTRUCTION AND EXPERIMENTAL VALIDATION OF A MONTE-CARLO SIMULATION MODEL OF POOLING

3.1 Introduction

Pooling is the simultaneous analysis of multiple samples followed by the deconvolution of pools that show promising results. Similar techniques have been used to increase throughput in combinatorial chemistry for drug discovery,^{2, 3} genome sequencing and chromosome mapping,⁴⁻⁷ oligonucleotide microarrays,^{8,9} and in testing of biological samples such as blood for the presence of organisms, contaminants, or characteristics of interest.¹⁰⁻¹³ As noted by Bruno et. al., “Whenever the objective is to find ‘needles in a haystack’ a reliable test indicating whether at least one needle occurs in a specific part of the haystack can greatly facilitate the isolation of the needles“.⁶ In each case, the object is to balance the increase in sample size obtained by pooling with the decrease in ability to detect the analyte of interest. If pools become too large, the signal from the organism in question will be lost in the background of the assay (Figure 3.1).

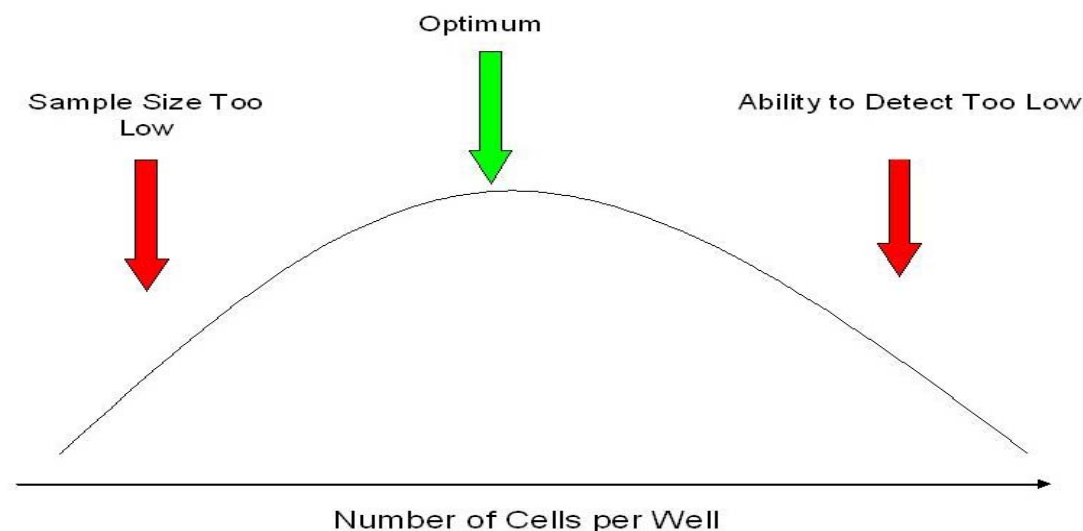


Figure 3.1: Finding the optimal level of pooling

As discussed in Chapter II, directed evolution is an iterative method of protein engineering where one or more genes that code for the protein of interest are subjected to random mutagenesis or recombination.^{14, 15} The resulting variants, or mutant proteins, must then be sorted into good and bad performers. If the activity of interest cannot be tied to host survival (selection), then each mutant must be assayed individually, a process called screening. Because of the time- and resource- intensive nature of screening, laboratories are typically limited to screening 10^5 to 10^7 mutants per round of directed evolution.¹⁶ This capacity limit impacts the way error-prone PCR experiments are conducted: since higher mutation frequencies yield larger numbers of inactive mutants, experimenters typically do not generate more than 1-2 mutations per gene. This ensures that the majority of mutants that are screened have at least some activity.¹ While this policy means that most of the screened library will be active, it decreases the accessible portion of sequence space. Therefore, if large improvements in the protein of interest

require many mutations, the probability that this solution will be tested is very low. Additionally, when recombination methods are used to generate diversity, the proteins that are actually assayed represent a very small portion of the ones that are produced. Therefore, it is likely that many improved proteins are missed due to the limitations of assay throughput.

We propose to extend the concept of pooling to the directed evolution of proteins. However, the level of pooling which can be tolerated in any system will depend on several factors, such as the background level of activity that the parental strain possesses, the accuracy level of the assay in question, how frequently improved mutants are produced, and what level of improvement is achieved by these mutants. The optimal level of pooling will vary with the particular situation at hand; therefore it would be useful to have a simulation model to test different conditions *in silico* before planning experiments.

The initial validation of the proposed model required a highly controlled experimental system with a colorimetric reporter assay. A first level test system was comprised of two enzymes, β -galactosidase (LacZ) and β -glucuronidase (Gus), which are thought to be evolutionarily related glycosyl hydrolases (Figure 3.2).^{17, 18} β -glucuronidase does catalyze the hydrolysis of the substrate of β -galactosidase, but at an approximately 10^6 -fold lower level. A mixture of LacZ and Gus thus comprises a binary system for which direct calculations of the probability of finding an improved enzyme can be made, providing further corroboration of the accuracy of the model. In addition, directed evolution experiments to increase this activity have been conducted,¹⁹⁻²¹ yielding mutants of various levels of galactosidase activity with which we could test more

complex activity distributions. Additionally, several reporter compounds are available to detect galactosidase activity; most of these allow for visual detection of the presence of a supermutant. Thus, these enzymes and their mutants comprise a system that can be used to validate the model in a controlled manner.

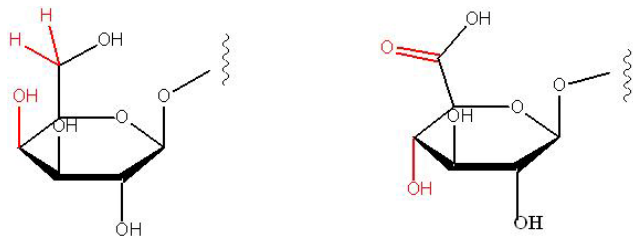


Figure 3.2: The substrates of β -galactosidase and β -glucuronidase. Left: a galactoside. Right: a glucuronide

3.2 Materials and Methods

3.2.1 Computer Simulations

The pooling experiments are complex enough that an analytical probability model of the system is unlikely to yield tractable closed-form results when considering complex assay accuracies or distributions of underlying population activity. This led to the development of a stochastic simulation model using Monte-Carlo methods.^{22, 23} The simulation essentially maps an idealized experimental procedure into an executable algorithm (see Figure 3.3). Monte-Carlo sampling is required whenever there is an underlying non-deterministic choice to be made. The Jet Random Number Generator from the European Center for Nuclear Research (CERN²⁴) was used to create random

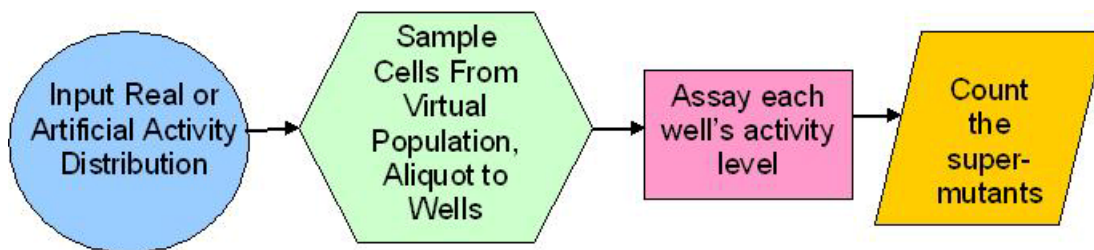


Figure 3.3: Pooling simulation model flowsheet

numbers from different probability distributions. Sampling is required during the following points in the simulation:

1. The pipetting of cells is assumed to draw a certain quantity of liquid whose volume is significantly greater than the volume of the cells that is expected to be within it. Thus the probability that any subvolume contains a cell is thus very small, a rare event, and the number of subvolumes very large. This enables the use of the Poisson distribution to represent a single pipette action. The mean of the Poisson distribution is taken to be the intended target number of cells per well. The concentration of the cells and the quantity of the original mixture are considered to be uniform and significantly greater than the amount that is being pipetted, so the overall process is represented by a series of identical, independently distributed trials. For example, if one cell per well is the target, then the probability of a given cell being empty is equal to $1/e$, or 0.368.
2. The activity curve for a population of cells is obtained by plotting the catalytic activity against the corresponding number of cells. A directed evolution experiment would sample for this activity curve by creating variants. The probability of a cell having an activity within a given range is dependent on the underlying population and processes that have been used to select it. For the

purposes of early model validation, cells with only two different levels of activity form the population. Thus, the activity curve was assumed to be a discrete probability distribution, with two masses, the probability of a supermutant and the probability of the dud, summing to unity. Although the activities of the supermutant and dud are assumed to be fixed, it is actually an average activity of the population and a finite spread would occur around it. However, it was unnecessary to represent this finite spread because it was assumed that it could be reflected in the assay accuracy. In later simulations, a more complex activity curve involving several mutants of varying activity levels was input. In practice, the model can handle any type of activity curve.

3. A normal distribution is used to model the assay error; however, in simulation runs involving a large activity level difference, the accuracy was sufficiently high that there was little chance that the two cell types could be confused because of assay error.

The computer code was constructed to allow for tracking of the true identity of cells in each well at the end of the simulation, allowing the identification of false positives.

3.2.2 Simulation Software

The simulation code (Appendix 1) is implemented in Java for ease-of-use, portability, and extensibility. The object-oriented nature of Java is a perfect fit for the experiments because each physical part of the process is described by a class. These classes can be individually tested and are easy to replace if the model needs to be changed. Once the code is compiled, it can be used on any platform that supports a Java

Virtual Machine, eliminating the need to implement different versions for different operating systems and making distribution simple.

The large scale of data being processed requires a fast computer with significant storage capability. A typical experiment has 80 well plates with 384 wells each, and each well with 1-100 cells. Each of those 1-2 million cells has data that needs to be stored, manipulated, and compared for each round of directed evolution. The running time for simulations on a Pentium 4 2.4 GHz processor with 512 Mb of RAM is on the order of a few minutes, depending on the number of cells sampled per well.

3.2.3 Verification of Simulation Results through Analytical Probabilistic Model

For the simulations involving a binary activity distribution, the modeling of the experimental results can also be approached through an analytical probabilistic model that captures the basic features of the pooling procedure. It is important to note that the assay accuracy also cannot be accounted for in the calculations, thus this calculation could only be made for the binary system involving the wild-type LacZ versus the wild-type Gus (AR=38,000). This model was used as further corroboration of the simulation procedure.

The number of cells sampled from the initial pool is assumed to follow a Poisson distribution, for the same reason as its use in the simulation. For an average j cells/well, the probability of sampling i cells is defined as $P_{i,j}$ and given by:

$$P_{i,j} = \frac{j^i e^{-j}}{i!} \quad (1)$$

Assuming that the initial pool of cells is infinite in number and the cells taken out in samples do not affect the supermutant:dud ratio (SR), the probability of picking out i duds in a sample can be written as:

$$p_i^D = (f_D)^i \quad (2)$$

where f_D is the fraction of duds in the initial pool. Thus, the probability of sampling only duds or no cells when the number of cells sampled is given by a Poisson distribution with an average j is:

$$\mathbf{P}_j^D = P_{o,j} + \sum_{i=1}^{n \gg j} P_{i,j} (f_D)^i \quad (3)$$

where n is a number much larger than the average cells per well j such that the term $P_{i,j}(f_D)^i$ is negligible. The probability of sampling one or more supermutants in a well can now be written as:

$$\mathbf{P}_j^S = 1 - \mathbf{P}_j^D \quad (4)$$

The probability of finding k wells with one or more supermutants when sampling a total of N wells with an average of j cells per well is written as:

$$\mathbf{P}_j^S(k) = (\mathbf{P}_j^D)^{N-k} (1 - \mathbf{P}_j^D)^k \frac{N!}{(N-k)!k!} \quad (5)$$

Note that the assay accuracy is assumed to be 100% in this derivation. From Eq. (5) the average number of supermutant cells detected for a dud ratio of f_D in the pool when sampled with an average of j cells/well can be written as:

$$\mathbf{P}_j^S = \sum_{k=0}^{\infty} (\mathbf{P}_j^D)^{N-k} (1 - \mathbf{P}_j^D)^k \frac{N!}{(N-k)!k!} \times k \quad (6)$$

3.2.4 Materials

Immunopure™ *o*-nitrophenylgalactopyranoside was obtained from Pierce Biotechnology (Rockford, IL). All other chemicals were obtained from Sigma Aldrich (St. Louis, MO). Molecular biology reagents and LB medium were purchased through VWR International

(Suwanee, GA). The genetic constructs for the wild-type β -galactosidase and β -glucuronidase were a kind gift of Dr. Ichiro Matsumura (Emory University).

3.2.5 Co-culturing Experiments

It was essential to determine whether cells expressing β -galactosidase and β -glucuronidase grow at the same rate, because if they do not the pooling ratio set at the beginning of the experiment would not be retained at the time of the assay. Therefore, cells expressing β -galactosidase and β -glucuronidase were grown separately in glass culture tubes at 37°C with 250 rpm shaking, and in a 1:1 co-culture under the same conditions, and growth curves based on optical density readings at 600 nm were constructed to compare the behavior of the mono- and co- cultures.

3.2.6 Cell Growth and Assay Conditions

Screening was conducted using cells expressing β -galactosidase as the supermutant and cells expressing β -glucuronidase as the dud. Genetic constructs for each enzyme are described in detail elsewhere.^{19, 25} Briefly, the gene encoding for each enzyme was cloned from *E. coli* genomic DNA into a library of pET20b vectors in which the T7 promoter has been removed and the lac promoter randomized to produce varying levels of constitutive expression,²⁶ transformed into *E. coli* INV α F'(lacZ Δ M15) cells and selected for the optimal and most stable level of expression.

Screening assay procedures were derived from Geddie et. al.²⁷ Initial stocks of each cell type were created by growing a single colony in Luria-Bertani (LB) medium supplemented with 100 μ g/mL of ampicillin to mid-log phase, diluting 50:50 with 33% glycerol, aliquotting into microcentrifuge tubes, and snap freezing with liquid nitrogen.

One tube of each type was then thawed and titered by serial dilution to determine the concentration prior to cell dilution.

For each screening run the supermutant and dud were mixed in a given ratio, the supermutant:dud ration (SR) (1 supermutant per 10^4 , 10^5 , or 10^6 duds), and then diluted to the appropriate number of cells per 5 μ l (1, 10, 40, or 100). Cells were allocated to 80 384-well plates using a Multidrop 384 (Thermo LabSystems, Waltham MA), sealed with silicon mats (Specialty Silicon, Rochester NY), and grown overnight at 37°C in a rotating incubator (Environmental Express, Mt. Pleasant SC). The mats were removed and 50 μ l of substrate solution (0.5 mM *ortho*-nitrophenyl-galactopyranoside (oNPG- Figure 3.4) in 1X Phosphate Buffered Saline, pH 7.3) was added to each well. The plates were incubated at room temperature for 3 hours and visually inspected for the appearance of bright yellow color indicating presence of the supermutant. For tests of assay limitations and comparison of assays, forced dilutions of cells expressing the “supermutant” and the “dud” (1:10 through 1:10⁵) as well as pure cultures of each type were grown overnight in glass culture tubes, 5 μ l of these cultures were dispensed into each well of a 384-well plate, 50 μ l of oNPG substrate solution or 25 μ l of 0.1 mM resorufin- β -D-galactopyranoside (ResGal-Figure 3.4) in 10 mM Tris-HCl pH 7.5 were added, and plates were incubated at 30°C. After 3 hours, the absorbance of the plates incubated with the oNPG substrate was read at 405 nm using a Fluorostar Galaxy (BMG Technologies, Offenburg, Germany). Plates used in the resorufin assay were incubated for 1.5 hours and assayed in fluorescence mode (excitation wavelength 544 nm, emission wavelength 590 nm). Because the use of pooling in a directed evolution experiment would require a second stage to isolate the true supermutants that were present in the winning well (two

sets of 80 plates would be necessary for a single set of pooled experiments), experiments of unpooled conditions were increased to a sample size of 160 plates to keep the comparison even in total effort.

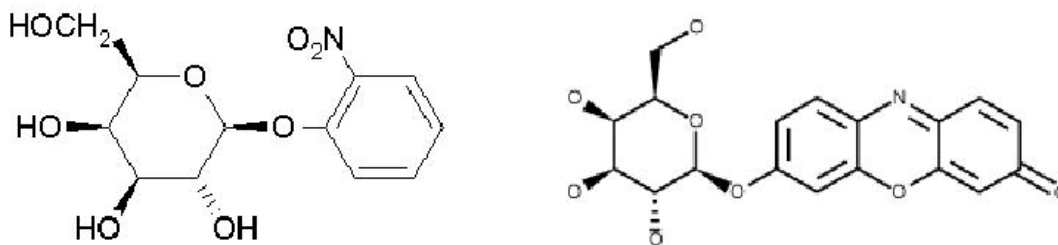


Figure 3.4: The substrates *ortho*-nitrophenylgalactopyranoside (left) and resorufin- β -D-galactopyranoside (right)

3.2.7 Cell Dilution Check with Coulter Counter

To determine whether the final dilution to the proper number of cells per well was accurate, the cell counts in samples of final solution were checked using flow cytometry (MultisizerTM 3, Beckman Coulter, Fullerton, CA). Several routes to the final dilution were examined to determine whether there was variation in the final number based on the method of dilution.

3.2.8 Mutant with 15-fold difference in activity level

The S557P, T509A/S557P, and T509A/S557P/N566S/K568Q mutants^{19, 20} were created using the QuikChange Site Directed Mutagenesis Kit (Stratagene, La Jolla CA) and sequenced to confirm the changes (Perry Mars, Fundamental and Applied Molecular Evolution Center, Emory University). The mutant and wild-type cultures were grown in Luria-Bertani medium to saturation in 10 mL cultures and the activity of the mutants, as well as the native β -glucuronidase and β -galactosidase, was verified in cell lysate (to

represent pooling conditions) and in purified form following his-tag purification with Talon Metal Affinity Resin (BD Biosciences, San Jose CA) using continuous monitoring of the change in absorbance at 405 nm. Forced dilutions to determine the limits of the assay were performed as for the wild-type, but incubation times were increased to allow for the slower reaction. Screening runs were conducted with 30 μ l of the resorufin substrate solution (0.1 mM in 10 mM Tris-HCl pH 7.5) and an incubation time of 9 hours.

The assay of pools of T509A/S557P/N566S/K568Q as the supermutant and β -glucuronidase as the dud (activity ratio, AR=15) were conducted at a SR of 1:10⁴ and pooling levels of 1, 5, and 10 cells per well. Due to the cost of the resorufin substrate, sample sizes were limited to 10 plates per experiment. Growth conditions were as described for the β -galactosidase / β -glucuronidase experiments and assay conditions were as described for the forced dilution experiments.

3.2.9 Standard Directed Evolution Activity Curve Experiments

The T509A/S557P/N566S/K568Q mutant (AR=15), the S557P/T509A mutant (AR=2), wild-type β -glucuronidase (AR=1), and cells harboring an empty plasmid (AR=0) were mixed in appropriate ratios to reproduce a typical activity curve (Figure 3.5). Stage one experiments were conducted as described above for cell dilution procedures, using a pooling level of five cells per well and an unpooled control. After assay, the top 15 wells from the pooled experiment were streaked onto LB medium containing ampicillin, and grown overnight. Six of the resulting colonies were chosen for each winning well from stage one and picked into 96-well plates for stage two. As additional verification, three samples from each of the top five wells were made from the

unpooled plate and picked into 96-well plates. The stage two colonies were grown overnight with agitation to stationary phase, 20 μ l of each well was transferred to fresh medium, and cells were grown for an additional 17 hours. Assays were conducted as described previously.

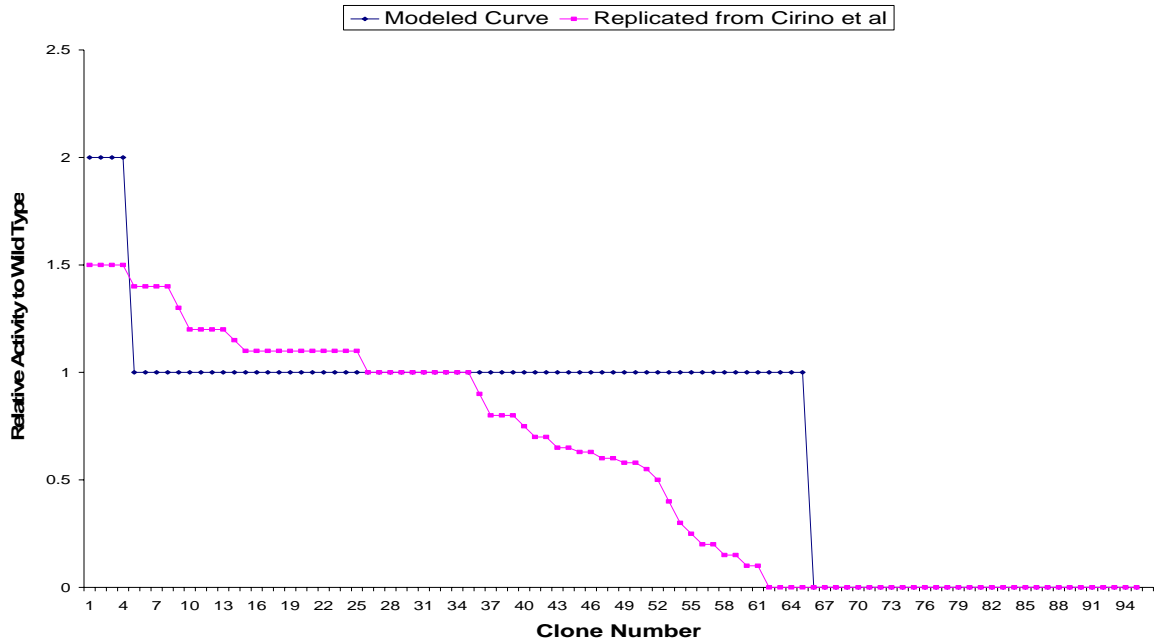


Figure 3.5: Standard Activity Distribution from a directed evolution experiment by Cirino et al.¹ with very few improved mutants, a large plateau of wild-type activity level, and about 30% inactives. Our smoothed approximation consisting of the mutants with 15-fold, 2-fold, wild-type, and no activities. The 15-fold mutant is included at a frequency of about 0.25 per 96

3.3 Results

3.3.1 Co-culturing Experiments

The results of the co-culturing experiment show that LacZ and Gus suggest that LacZ and Gus grow at a similar rate to each other and that the mono-cultures behave similarly to the 1:1 co-culture (Figure 3.6).

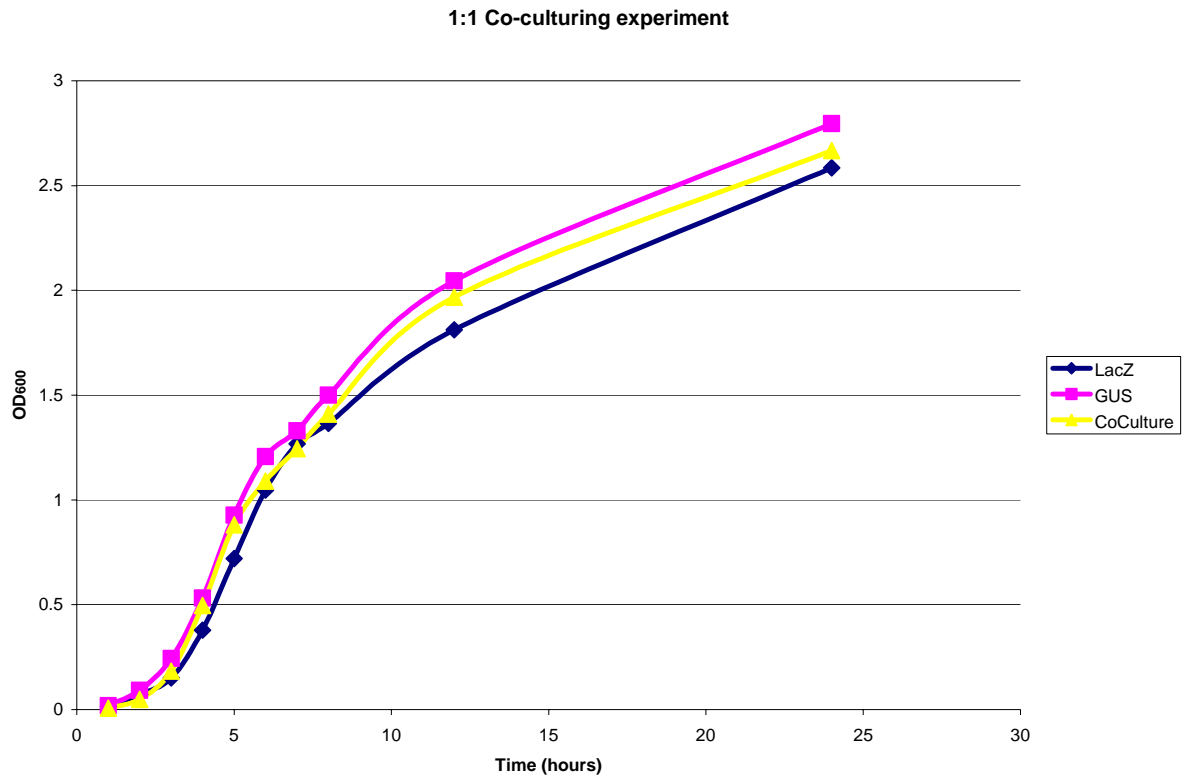


Figure 3.6: Results of the co-culturing experiment. Growth was done at 37°C with 250 rpm shaking in LB medium.

3.3.2 Flow Cytometry Check of Dilution

Flow cytometry was used to check the dilution procedures. Figure 3.7 shows the actual versus expected number of cells obtained via three different dilution procedures. The discrepancy between the predicted concentration of cells per well and

the counted one differed by only 5-10%. The agreement was far closer when flow cytometry was used as opposed to when spread plates were used to determine cell concentration. Note that Run B has large error bars due to problems with the orifice clogging in this condition.

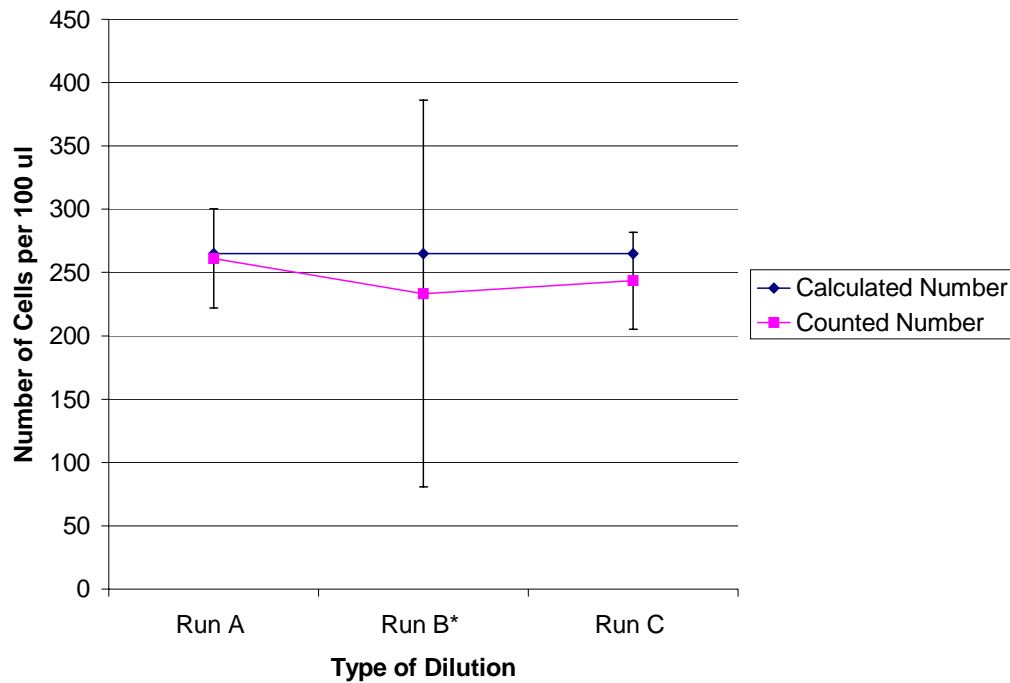


Figure 3.7: Check of cell dilution procedures with flow cytometry

3.3.3 Simulations and Direct Calculations

Simulations were conducted using activity ratios (AR) of 15, 38000, and 10^6 and supermutant:dud ratios (SR) of $1:10^4$, $1:10^5$, and $1:10^6$ respectively. At the lowest activity ratio (15), pooling conditions beyond 5 cells per well predicted finding supermutants in virtually all wells at all supermutant:dud ratios, a result of false positives. At activity ratios of 38000 and above, the number of supermutants found increases linearly with the number of pooled cells per well, and more supermutants are found as the

supermutant:dud ratio increases (example: Figure 3.8). The predicted number of supermutants for an activity ratio of 38000 and of 10^6 is nearly identical, indicating that once the activity ratio is far above the threshold of assay detection, the true number of supermutants is always detected. Theoretical calculations based on the Poisson distribution (see developments above) agree with the simulated results, for the most part within a single standard deviation of the simulation (Figure 3.8).

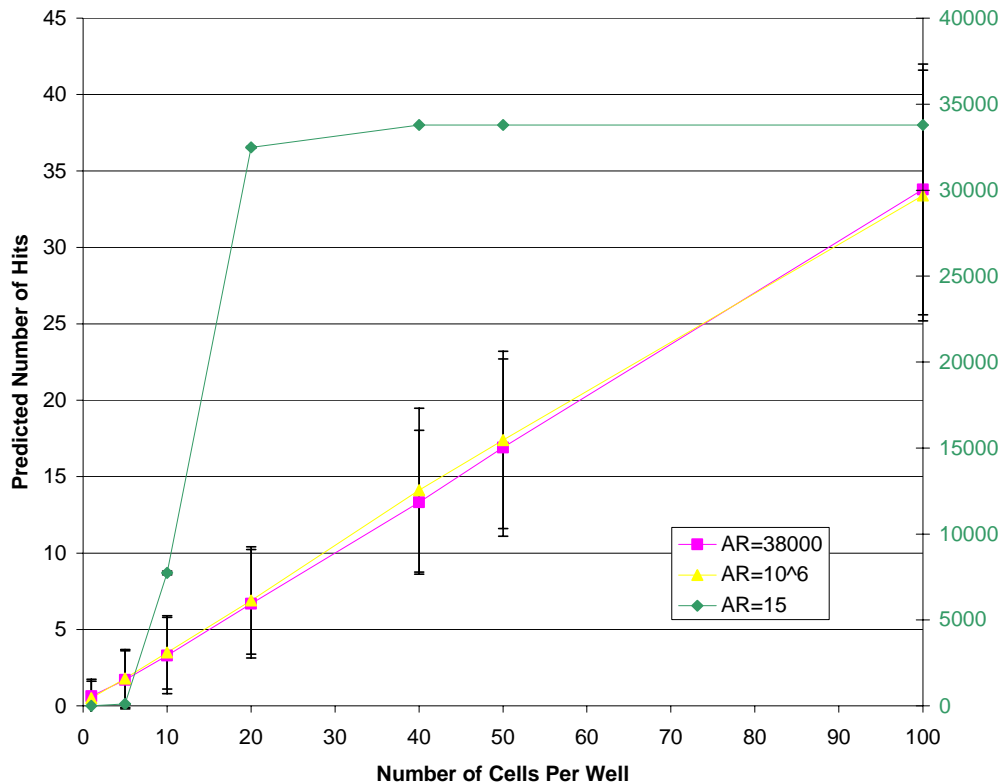


Figure 3.8: Example simulation data. The number of supermutants detected as a function of cells per well is plotted for all activity ratios at a supermutant:dud ratio of $1:10^5$. The data for activity ratio (AR) of 15 is plotted on a secondary axis to aid in interpretation.

3.3.4 Initial Experimental Validation

Initial experiments focused on two populations with a large difference in activity level. The activity ratio of cells expressing β -galactosidase to cells expressing native β -glucuronidase on oNPG in cell lysate was measured to be 38000; thus, this number was used in simulations. Published estimates of the activity difference range up to 10^6 , but these are for pure enzymes and do not represent the screening conditions of a whole-cell assay. Additionally, as mentioned above, simulations do not substantially differ for these activity ratios.

The model was tested at supermutant:dud ratios of $1:10^4$ - $1:10^6$. Most experiments were conducted at a supermutant:dud ratio of $1:10^5$, because the largest difference in detection ability between pooled and unpooled experiments occurs here (central line of Figure 3.9). Additional experiments were conducted at the interesting endpoints of $1:10^4$ and $1:10^6$ supermutant:dud ratio. In the former case, the model always predicts that a supermutant will be found, even if one does not do pooling. In the latter, the model predicts a very low probability of finding the supermutant, even at pooling levels of 100 cells per well. Figure 3.10 shows the comparison of the theoretical calculation, the simulation model and the experimental results. For the supermutant:dud ratio of $1:10^5$, the calculations, simulations, and experimental results agree very closely, indicating that the model fits well. For the SR of $1:10^4$ and $1:10^6$ the simulated and calculated results are slightly higher than the experimentally observed ones.

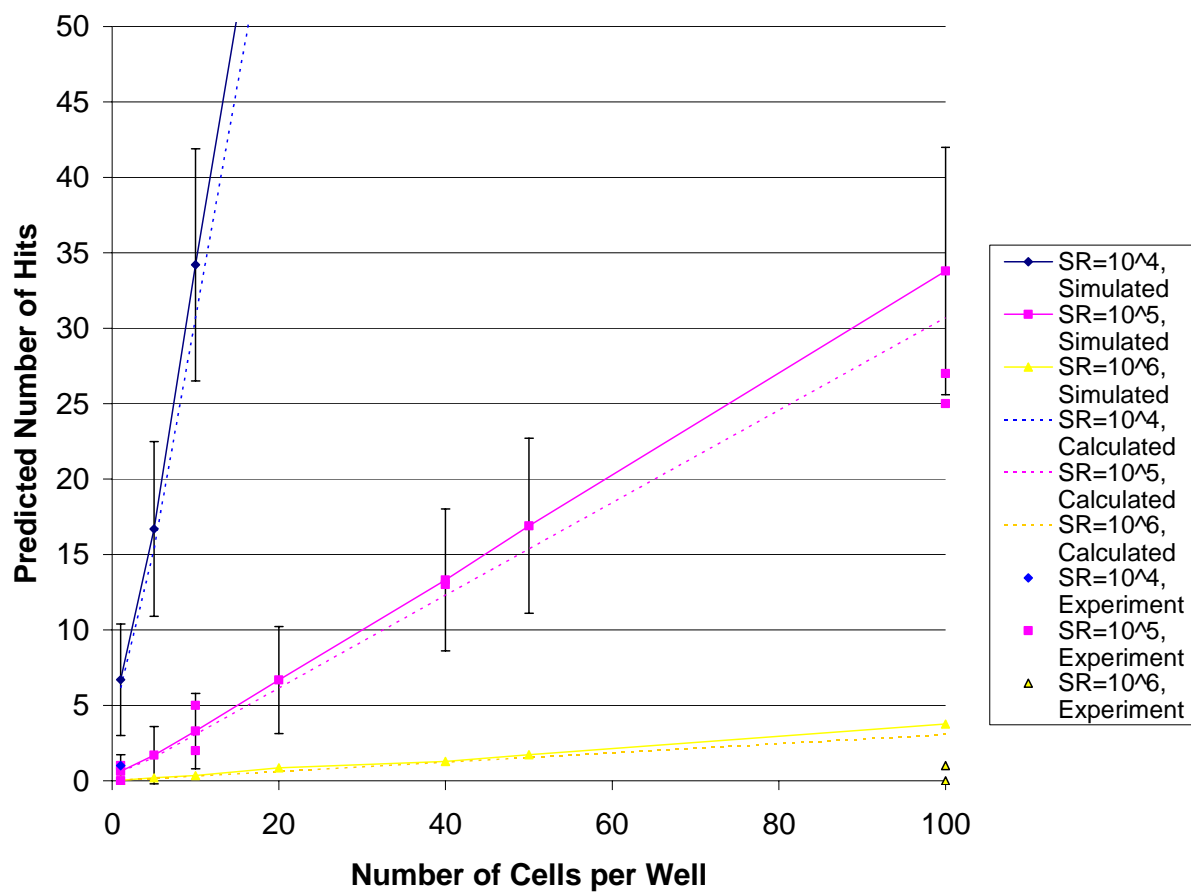


Figure 3.9: Comparison of simulated and experimental results for pooling with cells expressing β -galactosidase as the supermutant and cells expressing β -glucuronidase as the dud. The number of supermutants detected is plotted versus the number of cells pooled per well.

3.3.5 Forced Dilution Experiments

Both to test the limits of pooling and to compare two assays of varying accuracy, artificial pools of cells expressing β -galactosidase and cells expressing β -glucuronidase were tested in the oNPG and resorufin assays. Figure 3.10 indicates that pools of $1:10^3$ and higher do not allow for the detection of the supermutant from the background of the assay. This is true even when incubation times are increased (data not shown) and is probably a function of the relative rates of hydrolysis of the supermutant and dud. Since our method involves visual inspection, it was also important that we be able to distinguish wells containing a supermutant by eye. This was much easier in the case of the resorufin assay where colors ranged from orange (unreacted) to magenta (fully reacted), versus in the oNPG assay which calls for distinction between various shades of yellow (Figure 3.11).

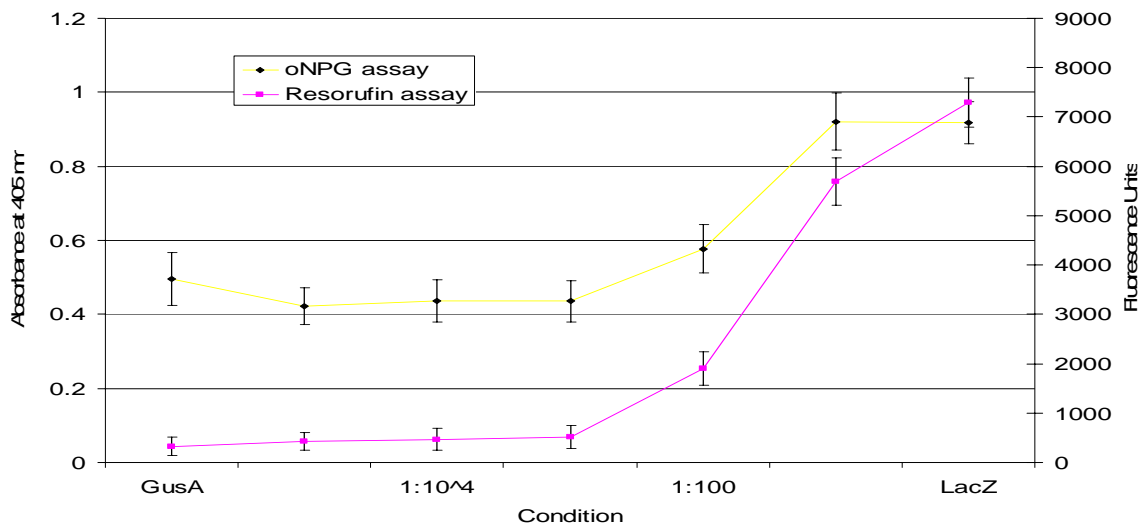


Figure 3.10: Assay of artificial pools of cells expressing β -glucuronidase and cells expressing β -galactosidase. For the oNPG assay 50 μ l of 0.5 mM oNPG in 1X phosphate buffered saline were added to each well and well plates were incubated for 4 hours at 30°C before being assayed spectrophotometrically at 405 nm. For the ResGal assay 25 μ l of 0.1 mM Resorufin- β -D-galactopyranoside in 10 mM Tris-HCl were added to each well and plates incubated at 30°C for 1.5 hours. Fluorescence units were determined using a fluorimeter with excitation at 544 nm and emission monitored at 590 nm.

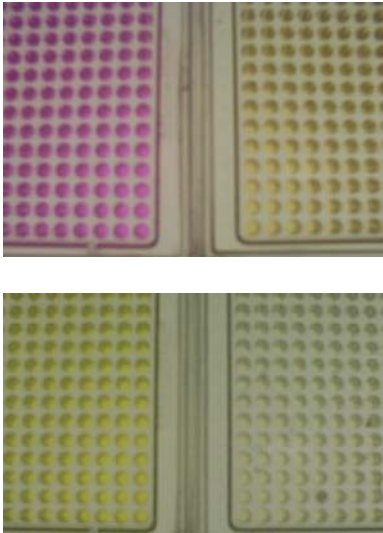


Figure 3.11: Full color photos of the assay results for cells expressing β -galactosidase (left) and cells expressing β -glucuronidase (right). The top plates are the resorufin assay, the bottom are the oNPG assay

3.3.6 Experiments at Lower Activity Level Differences

Since differences in activity level of 38000-fold are rare for mutants created by directed evolution, particularly in initial rounds, pooling experiments with a mutant of β -glucuronidase (T509A/S557P/N566S/K568Q) with 15-fold activity level difference in cell lysate were conducted^{19,20}.

Pools of this mutant with cells expressing wild-type β -glucuronidase were tested with both assays to determine the limits of pooling for spectrophotometric and visual detection. For the oNPG assay, even pools of 1 mutant: 2 wild-type were barely distinguishable spectrophotometrically from the wild-type background (Figure 3.12). However, for the resorufin assay, ratios of up to 1 mutant: 10 wild-type were distinguishable both with the fluorimeter (Figure 3.12) and visually (data not shown).

Two additional mutants created in the process of making the quadruple mutant, S557P and T509A/S557P, both with approximately two-fold increased specific activity over the wild-type GUS were also assayed to determine whether they could be detected in pools. Neither of the mutants was distinguishable from wild-type GUS in either assay (Figure 3.12). Simulations results incorporating the coefficient of variation of both assays agree that pooling of these mutants with only wild-type β -glucuronidase is not worthwhile.

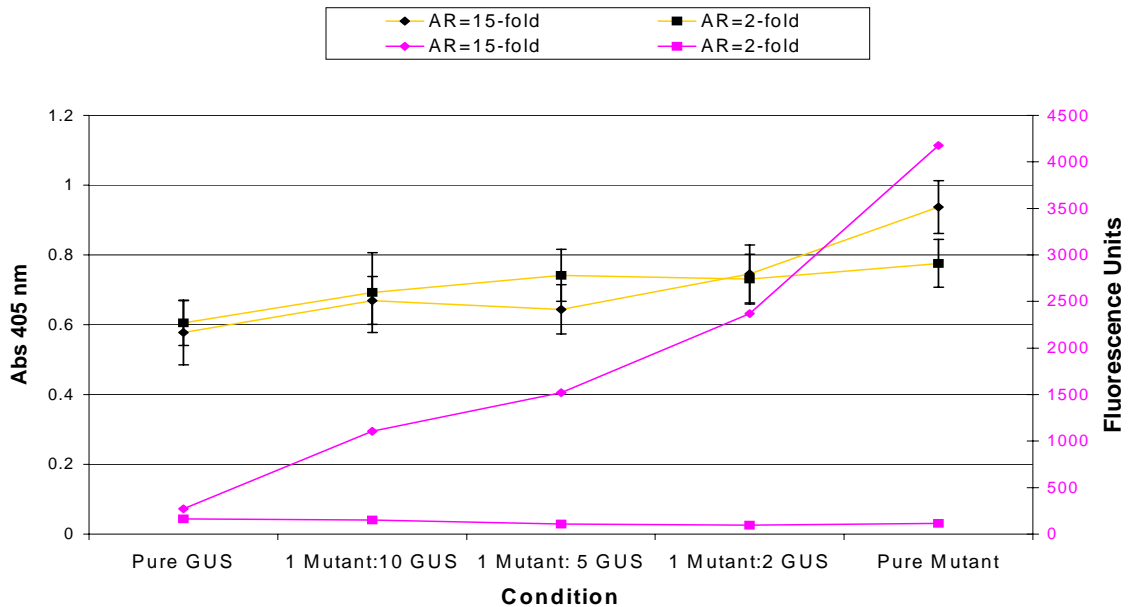


Figure 3.12: Assay of forced pools of cells expressing β -glucuronidase with cells expressing the 3 mutants: S557P, T509AS557P, and T509A/S557P/N566S/K568Q. The S557P and T509AS557P mutants have 2-fold increased specific activity over wild-type GUS in cell lysate assays, while the T509A/S557P/N566S/K568Q mutant has 15-fold increased specific activity in cell lysate assays. The oNPG assay was conducted as for the β -galactosidase and β -glucuronidase case, but incubation time was increased to 6 hours. The resorufin assay was conducted as for the β -galactosidase and β -glucuronidase case, but incubation time was increased to 9 hours (secondary axis).

To test the validity of the model at low activity ratios, experiments using the resorufin assay, a supermutant:dud ratio of $1:10^4$, and pools of 1, 5, and 10 cells per well were conducted with the quadruple mutant. Figure 3.13 compares the simulation and experimental results for these experiments. At an activity ratio of 15, there are a substantial number of false positives occurring in the simulation, particularly at and above 10 cells per well (axis has been truncated). Once these are taken into account, the model predicts the number of supermutants detected fairly well. When an activity ratio of 35 (green line) is used to run the simulation, there are negligible numbers of false positives. Note that this simulation predicts the same number of supermutants as the previous simulation when the false positives are subtracted. Other supermutant:dud ratios were not tested because too few supermutants were expected in the small sample size necessary to use the resorufinated substrate.

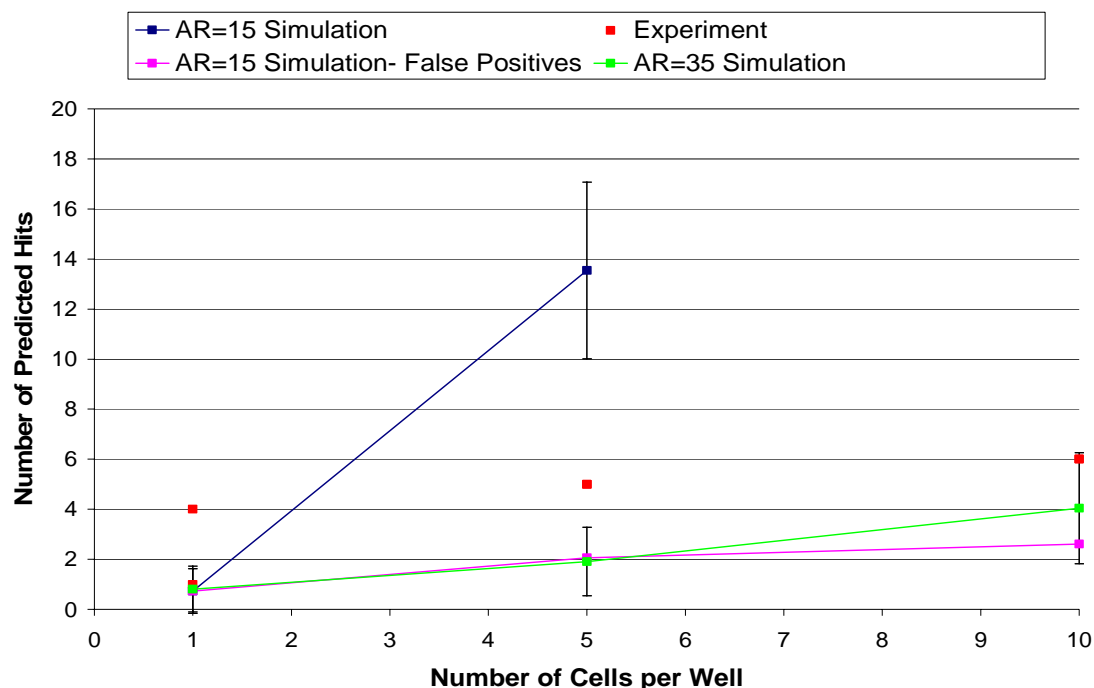


Figure 3.13: Comparison of simulation and experimental data for pooling with cells expressing β -glucuronidase and cells expressing the mutant with 15-35-fold increased activity level (T509A/S557P/N566S/K568Q). Experiments were conducted at a SR of $1:10^4$ using a 10 plate sample size.

3.3.7 Standard Activity Distribution

Figure 3.14 compares the number of T509A/S557P/N566S/K568Q (AR=15) mutants predicted to be present by the simulation with the number experimentally detected for pooled and unpooled conditions after the second stage. The simulation accurately predicts the number detected regardless of the assay employed. This result may seem to be in contrast with the forced dilution experiments, however the presence of the inactive mutant means that the T509A/S557P/N566S/K568Q (AR=15) mutant can be detected with some probability (when these end up in the same well).

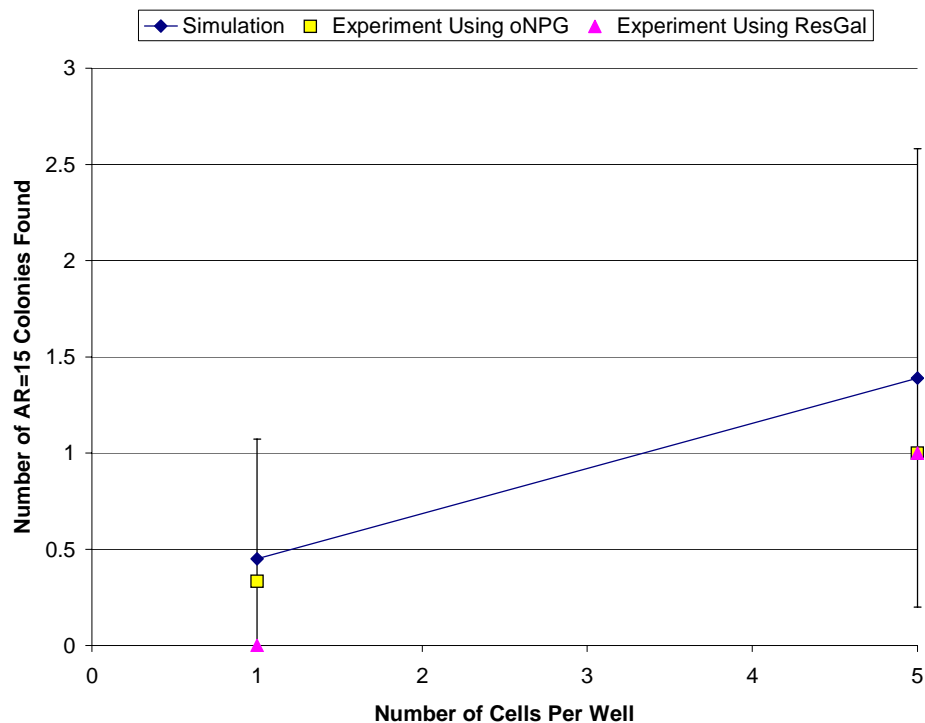


Figure 3.14: Predicted and actual number of AR=15 mutant detected when pooling with a more typical activity distribution¹. Experiments were conducted using one 96-well plate for each stage.

3.4 Discussion

3.4.1 Simulation Results

The simulation results reflect the physical phenomena that the experimenter witnesses. For instance, when the activity level difference between the supermutant and dud is much larger than the assay accuracy, the simulation always detects the true number of supermutants present. Hence, the simulated numbers of supermutants for the activity level ratio of 38000 and 10^6 are almost identical. Additionally, simulations at low activity ratio (e.g. 15) and high pooling level predict the occurrence of a supermutant in

every well. The physical manifestation of this result would be the inability to distinguish between any of the wells because the pooling level is too high to allow for the detection of the supermutant within the error of the assay. The simulation procedure allowed us to record which “supermutants” were false positives and which were false negatives. At low activity ratios, a substantial number of false positives were detected as evidenced by Figure 3.14. The numbers of false positives were negligible at higher activity ratios, when the difference between improved and unimproved mutants is far larger than the assay error in detection. The number of false negatives was insignificant in all cases.

3.4.2 Model Validation

Overall, the experiments support that the model accurately predicts the number of supermutants that will be found for many cases. Among the variables accounted for in the Monte-Carlo model, the supermutant:dud ratio has the largest impact on the number of supermutants detected. This is logical, because the more supermutants produced in the diversity generation step of directed evolution, the higher the probability that one will be sampled for assay. In our experiments, when one supermutant was generated for every 10^4 duds, supermutants were detected even in the unpooled conditions. On the other hand, when one supermutant was generated for every 10^6 duds, even pooling 100 cells per well does not guarantee the detection of a supermutant in every experiment. The results indicate that the amount of diversity generated will greatly impact the success of the screening and there is a minimum number of supermutants that must be generated before screening is even worthwhile. Note that although supermutants are found in the $1:10^4$ SR condition when pooling is not employed, the number of supermutants detected still increases when pooling is used. In an actual directed evolution experiment where

there is not a single type of improved variant generated, it would still be beneficial to use pooling to bias the populations in future rounds towards higher activity mutants.

3.4.3 The Utility of Pooling in Directed Evolution Experiments

Pooling is most suited to cases where an investigator is looking for a large increase in activity over the background level, such as the case where one is attempting to evolve a new function in a protein. In this case, the number of mutants with the desired activity will be very few; therefore, pooling will be very useful in finding these few winners among the vast number with no activity. In addition, the wild-type parental gene is unlikely to have a high level of new desired activity in such an evolution experiment. Thus, the noise in the assay is minimized, increasing the number of cells that can be pooled before the background overwhelms the ability to detect. Preliminary evidence of this can be seen in the experiments approximating a standard activity distribution. Here, despite what would have been predicted from the forced dilution experiments, we were able to find the 15-fold more active mutant with both assays as predicted by the model. Further, we expect that the use of pooling will allow for a higher mutation load in error-prone PCR experiments, as more samples can be screened and high numbers of inactive mutants are actually beneficial, as they will not contribute to the background of the assay.

To use the model to guide directed evolution experiments, an investigator can input the assay accuracy, an estimate of the supermutant:dud ratio, and the desired activity increase that they wish to detect. The model will output the proper level of pooling to be used. In practice, the assay accuracy is most easily represented by the coefficient of variation seen when replicates are assayed. The supermutant:dud ratio can be estimated from initial library generation experiments, or alternatively, can be varied

among likely values to determine the effect on the model. The desired activity level increase can also be varied, but our simulation results show that as long as the activity level increase is larger than the assay error, the supermutant will always be detected when present.

3.5 Summary and Conclusions

In summary, pooling is a method to increase the capacity of high-throughput screening assays. The Monte-Carlo simulation model we have developed and validated can be used to predict the expected number of winners in directed evolution experiments, given a putative distribution of activity over the population (for further discussion of the activity distribution of actual directed evolution experiments, please see Chapter IV). This allows researchers to test several different scenarios before beginning experimentation. Additionally, the end user can input his/her plate capacity, the number of wells in each plate, an estimated frequency that good mutants will be created, an estimation of the accuracy of the assay, and the activity level increase that he/she hopes to find and use the model to determine the number of cells that should be pooled into each well for that experiment. The use of pooling will greatly increase the capacity of a laboratory to perform screening in directed evolution experiments.

REFERENCES

1. Cirino, P. C.; Mayer, K. M.; Umeno, D., Generating Mutant Libraries Using Error-Prone PCR. In *Directed Evolution Library Creation: Methods and Protocols*, Arnold, F. H.; Georgiou, G., Eds. Humana Press Inc: Totowa, 2003; Vol. 231, pp 3-9.
2. Venton, D. L.; Woodbury, C. P., Screening Combinatorial Libraries. *Chemometrics and Intelligent Laboratory Systems* **1999**, 48, 131-150.
3. Hwang, F.; Liu, Y., Error-Tolerant Pooling Designs with Inhibitors. *Journal of Computational Biology* **2003**, 10, (2), 231-236.
4. Balding, D. J.; Torney, D. C., The Design of Pooling Experiments for Screening a Clone Map. *Fungal Genetics and Biology* **1997**, 21, 302-307.
5. Barillot, E.; Lacroix, B.; Cohen, D., Theoretical Analysis of Library Screening Using a N-Dimensional Pooling Strategy. *Nucleic Acids Research* **1991**, 19, (22), 6241-6247.
6. Bruno, W.; Knill, E.; Balding, D.; Bruce, D.; Doggett, N.; Sawhill, W.; Stallings, R.; Whittaker, C.; Torney, D., Efficient Pooling Designs for Library Screening. *Genomics* **1995**, 26, 21-30.
7. Cai, W.-W.; Chen, R.; Gibbs, R. A.; Bradley, A., A Clone-Array Pooled Shotgun Strategy for Sequencing Large Genomes. *Genome Research* **2001**, 11, (10), 1619-23.
8. Peng, X.; Wood, C. L.; Blalock, E. M.; Chen, K. C.; Landfield, P. W.; Stromberg, A. J., Statistical Implication of Pooling RNA samples for Microarray Experiments. *BMC Bioinformatics* **2003**, 4, no pp. given.
9. Han, E.-S.; Wu, Y.; McCarter, R.; Nelson, J. F., Reproducibility, Sources of Variability, Pooling, and Sample Size: Important Considerations for the Design of High-Density Oligonucleotide Array Experiments. *The Journals of Gerontology* **2004**, 59A, (4), 306-315.
10. Sham, P.; Bader, J. S.; Craig, I.; O'Donovan, M.; Owen, M., DNA Pooling: A Tool For Large-Scale Association Studies. *Nature Reviews Genetics* **2002**, 3, (11), 862-871.
11. Andersson, S.; Gessain, A.; Taylor, G. P., Pooling of Samples for Seroepidemiological Surveillance of Human T-cell Lymphotropic Virus Types I and II. *Virus Research* **2001**, 78, (1-2), 101-106.

12. Panchagnula, R.; Sharma, A.; Agrawal, S., Plasma Pooling Methodology as a Faster and Cheaper Tool to Evaluate Bioequivalence of Rifampicin Component of FDCs of Antitubercular Drugs. *Pharmacological Research* **2003**, 48, 655-663.
13. Brookmeyer, R., Analysis of Multistage Pooling Studies of Biological Specimens for Estimating Disease Incidence and Prevalence. *Biometrics* **1999**, 55, (2), 608-612.
14. Stemmer, W. P. C., Rapid Evolution of a Protein in vitro by DNA Shuffling. *Nature* **1994**, 370, (4), 389-391.
15. Arnold, F. H.; Volkov, A. A., Directed Evolution of Biocatalysts. *Current Opinion in Chemical Biology* **1999**, 3, 54-59.
16. Steipe, B., Evolutionary Approaches to Protein Engineering. *Current Topics in Microbiology and Immunology* **1999**, 243, 55-86.
17. Henrissat, B., A Classification of Glycosyl Hydrolases Based on Amino Acid Sequence Similarities. *Biochemistry Journal* **1991**, 280, (Part 2), 309-316.
18. Huber, R.; Gupta, M.; Khare, S., The Active Site and Mechanism of the beta-galactosidase from Escherichia coli. *International Journal of Biochemistry* **1994**, 26, (3), 309-318.
19. Rowe, L. A.; Geddie, M. L.; Alexander, O. B.; Matsumura, I., A Comparison of Directed Evolution Approaches Using the b-Glucuronidase Model System. *Journal of Molecular Biology* **2003**, 332, (4), 851-860.
20. Matsumura, I.; Ellington, A. D., In vitro Evolution of Beta-glucuronidase into a Beta-galactosidase Proceeds Through Non-specific Intermediates. *Journal of Molecular Biology* **2001**, 305, (2), 331-339.
21. Geddie, M. L.; Matsumura, I., Rapid Evolution of b-Glucuronidase Specificity by Saturation Mutagenesis of an Active Site Loop. *Journal of Biological Chemistry* **2004**, 279, (25), 26462-26468.
22. MacKay, D. J. C., Introduction to Monte Carlo Methods. In *Learning in Graphical Models*, Jordan, M. I., Ed. Kluwer Academic Press: 1998; pp 175-204.
23. Chandler, D., *Introduction to Modern Statistical Mechanics*. Oxford University Press: 1987.
24. <http://hoschek.home.cern.ch/hoschek/colt/V1.0.3/doc/overview-summary.html>
25. Matsumura, I.; Rowe, L. A., Whole Plasmid Mutagenic PCR For Directed Protein Evolution. *Biomolecular Engineering* **2004**, Accepted.

26. Matsumura, I.; Olsen, M.; Ellington, A., Optimization of Heterologous Gene Expression for In Vitro Evolution. *BioTechniques* **2001**, 30, (3), 474-476.
27. Geddie, M. L.; Rowe, L. A.; Alexander, O. B.; Matsumura, I., High Throughput Microplate Screens for Directed Evolution. *Methods in Enzymology* **2004**, 388, 134-145.

CHAPTER IV

EXTENSION OF THE POOLING MODEL TO AN ACTUAL DIRECTED EVOLUTION LIBRARY

4.1 Introduction

In Chapter III, we introduced the concept of pooling, an established method to increase the throughput in screening assays for lead optimization in combinatorial chemistry¹⁻³ and in other disciplines (e.g. epidemiology, genomic mapping) where the object of an assay is to determine whether a certain agent or sequence is present in a group or not.⁴⁻⁷ Screening of combinatorial libraries from protein evolution experiments shares the same ‘needle in a haystack’ situation, however, there are several factors which complicate the ability to apply pooling to protein engineering. Thus, we created and experimentally validated a Monte Carlo simulation model of pooling on a very controlled system created by mixing known entities in various ratios (see Chapter III for details). Using this controlled system, we were able to show that pooling increases the assay throughput and can be used to increase the number of high activity variants identified in the screening step.⁸ In this chapter, we show that the concept of pooling can be extended to a real directed evolution library created in the course of evolution of β -galactosidase to β -fucosidase.⁹ We also extend the system to evolutionary projects involving induction of

protein expression, a situation that is very common in the high throughput screening of combinatorial libraries.

4.2 Materials and Methods

4.2.1 Materials

Molecular biology reagents and microbial medium supplies were purchased through VWR International (Suwanee, GA). The substrate *p*-nitrophenyl-fucopyranoside was purchased from Sigma Chemicals (St Louis, MO). The QiaSpin Miniprep kit was obtained from Qiagen (Valencia, CA). The randomized β -galactosidase library was a kind gift of Dr. Ichiro Matsumura (Emory University).

4.2.2 Simulation Model

The simulation procedures were conducted as described in Chapter III. The model can be set to provide one of three types of output. SUPERMUTANT allows for the evaluation of the number of highest activity mutants detected in the pooling scenario, as well as an estimation of the number of false positives in that total. TOP_PCT outputs the level of activity of each of the top X percent of wells specified by the user (the PRINT_ALL function can be used to output the exact values of each of the wells). Finally, the GREATER_THAN function can be used to obtain a count of all mutants with activity greater than the specific ratio, with annotation of the number of false positives. In all cases, a false positive occurs when a well is marked as having a supermutant within it, but really does not. A false negative occurs when a well containing a supermutant is not marked as such. Quantification of false positives and negatives can be done in the simulation since the true identity of every cell is known.

To solve for the number of supermutants that were detected in each case, the model requires an input of the expected frequency of occurrence of the improved mutant (supermutant to dud ratio), the improvement in activity level compared to wild-type (activity ratio), an activity distribution, an assay accuracy, the number of plates in each stage, and the pooling level of each experiment.

4.2.3 Experimental System

A 1997 paper by Zhang et al showed β -galactosidase, an enzyme that hydrolyzes the sugar moiety galactoside from a substrate, could be evolved to increased activity on a related sugar, fucoside, by successive rounds of DNA shuffling and screening with colorimetric substrates.¹⁰ Subsequent work by Parikh and Matsumura showed that greater gains in activity and selectivity could be achieved by randomization of three active site residues, D201, H540, and N604, which were identified using a semi-rational strategy.⁹ The best mutant obtained in this study was D201/H540V/N604T which is identified as the supermutant of interest in our study.

4.2.4 Model Inputs

The supermutant:dud ratio, in our case the frequency of occurrence of the D201D/H540V/N604T mutant, was estimated from the probability of obtaining these codons at each of the three positions based on the codon randomization scheme ($2/32*2/32*3/32=1/8192$). The activity ratio of the D201D/H540V/N604T mutant to the wild-type was determined using whole cells expressing both enzymes. The activity distribution was obtained from a preliminary screen of a single 384-well plate (Figure 4.1). Each value obtained was normalized by the average value of the wild-type β -galactosidase obtained in a parallel experiment, so that data could be input in the form of

an activity ratio. The coefficient of variation of the *p*-nitrophenyl assay was measured in duplicate experiments both on cells grown and induced directly in the 384-well plate and on cells grown and induced separately and added to the 384-well plate just prior to addition of substrate. The former was used to estimate the assay accuracy under the conditions of the screen, while the latter was used to estimate the assay accuracy in the absence of the variability caused by induction. Parikh and Matsumura screen 10,000 colonies using a filter lift assay,⁹ therefore, we screened approximately the same number of colonies in 384-well plates without pooling as a basis for comparison (26 plates). For pooled conditions, the same number of plates was used over the two stages for consistency, giving a total of 13 plates per stage. The optimal pooling level was determined by running simulations at increasing numbers of cells per well until the number of false positives detected was more than the true number of supermutants detected.

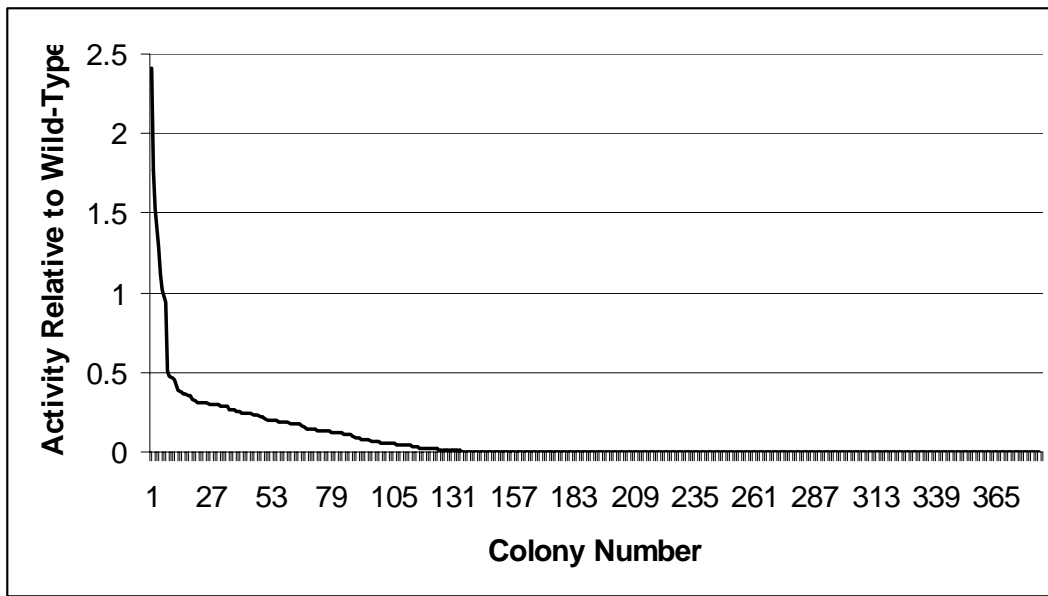


Figure 4.1: The activity curve derived from a single 384-well plate screen of the library using a substrate solution consisting of 0.5 mM *p*-nitrophenyl- β -D-fucopyranoside in 10 mM Tris-HCl pH 7.6. The absorbance at 405 nm was taken after 2 hours of incubation at 30 °C. Absorbance values were divided by the average value of absorbance for a plate consisting entirely of the wild-type β -galactosidase assayed in the same manner.

4.2.5 Model Sensitivity Analysis

Because the activity distribution was estimated from a very small sample of the actual library, a sensitivity analysis was conducted to determine the effects of errors in the activity distribution. Various activity curves with the same mean, but different shapes were used as input, with all other parameters being held constant and the number of supermutants detected in each case was compared. Additional simulations were conducted at varying assay accuracies (coefficient of variation 5%-50%) and varying activity ratios (5, 50, 500, and 5000), to understand the interdependence of the activity level and the assay accuracy.

4.2.6 Library Construction Methods

The library construction procedures have been described elsewhere.⁹ Briefly, three active site residues were chosen for site-saturation mutagenesis based on the crystal structure of the E537Q mutant of β -galactosidase with *p*-nitrophenyl-galactoside. Primers were designed based on an NNK scheme (32 possibilities for each residue, all 20 amino acids, but only one stop codon) with an additional pair used for vector amplification. Whole plasmids containing the library of genes were assembled from overlap extension PCR of the four segments, followed by intramolecular ligation. Plasmids were transformed into *E. coli* DH5 α Δ lac cells by electroporation. Following electroporation, cells were grown for 1 hour at 37°C in 1 mL of SOC medium to allow for recovery and then mixed with an equal volume of 33% glycerol and snap frozen using liquid nitrogen. One aliquot was thawed and titered by serial dilution for an approximate concentration of successful transformants.

4.2.7 Cell Growth, Induction, and Assay Procedures

Frozen aliquots of transformation mixture were thawed on ice and diluted to the appropriate number of cells per 5 μ l of LB medium supplemented with 30 μ g/mL of kanamycin (volume of one well) and aliquotted to the wells of ten 384-well plates using a Multidrop 384 dispenser (Thermo Labsystems, Waltham MA). The plates were sealed with silicon mats (Specialty Silicon, Rochester NY), and grown for 17 hours at 37° C with shaking at 140 rpm. Subsequently, a single drop of overnight culture was used to inoculate 5 μ l of fresh LB-kanamycin in a new 384-well plate using a multi-blot replicator (V&P Scientific, Inc, San Diego, CA). Cells were grown for an additional 3 hours at 37°C with shaking to allow for recovery and 5 μ L of LB-kanamycin plus 0.2 mM

IPTG were added to induce protein expression. After 1 hour of induction, 50 μ L of substrate solution (0.5 mM p-nitrophenyl-fucopyranoside in 10 mM Tris-HCl pH 7.6) was added.

The level of substrate hydrolysis was assayed after 4 hours of incubation at 30°C using a Fluorostar Galaxy (BMG Technologies, Offenburg, Germany) and a wavelength of 405 nm.

4.2.8 Deconvolution of Winning Pools

Wells which turned visibly yellow were picked for deconvolution. Ten μ l of the well was inoculated into 5 mL of LB plus kanamycin (30 μ g/mL) and grown at 37°C with shaking for 6 hours. These were mixed 50:50 with 33% glycerol and snap frozen. One aliquot of each well was thawed, diluted, and spread on LB-kanamycin agar plates. Twenty colonies were randomly picked (2-fold oversampling) and subjected to a secondary assay to identify the good variants. Variants with activity higher than 1.2-fold times wild-type LacZ, but less than 0.8 times the D201/H540V/N604T mutant were identified as improved, but not supermutants. Those with 0.8-1.2 times the activity of D201/H540V/N604T mutant were identified as supermutants.

4.2.9 Verification of Identity of “Winners”

All improved and supermutants identified from the unpooled condition, as well as all supermutants identified in the pooled conditions, were grown in a 5 mL culture, the plasmid retrieved by miniprep, and the colony sequenced (Fundamental and Applied Molecular Evolution Center, Emory University) to determine the amino acid sequence. A small sample of the improved mutants and those identified as wild-type equivalents from the pooled condition were also sequenced to get an estimate of false positive rates.

4.3 Results

The first goal of this work was to evaluate the sensitivity of the model predictions to parameters that might be hard to estimate, or control, for a combinatorial protein engineering system. We were interested both in understanding how changes in the estimation of the activity distribution would affect the accuracy of the prediction, as well as how the activity ratio and the coefficient of variation of the assay, two parameters which are interdependent, would affect the ability to accurately predict the number of supermutants detected after pooling.

The second goal was to use the model in both pooled and unpooled conditions to predict the number of supermutants that would be found after screening the fucosidase library. In doing so, we would need to set a threshold level for the activity of the supermutant (in effect, define what a supermutant is for the model). To determine whether the threshold should be set equal to the activity ratio, or slightly below, we explored the effect of both definitions on the predictions. We then compared the predicted and experimentally determined number of supermutants at both thresholds.

4.3.1 Sensitivity to Activity Distribution Input

Diversity generation by error-prone polymerase chain reaction (epPCR) results in sigmoidal shaped curves with a large number of mutants displaying wild-type activity and a large number of inactive mutants, the exact shape and steepness of which depend on the mutation frequency. The wild-type activity results from random mutations in tolerated positions, or screening of unmutated sequences when the mutation rate is very low. Inactivation is a result of mutation in residues which cannot tolerate substitution and can be the result of misfolding or destruction of catalytically important residues. In

contrast to epPCR, diversity generation by active site randomization is very disruptive to activity because of the proximity of the targeted residues to the catalytic sites. Thus, activity curves from randomization experiments resemble exponential decay curves. Three different sigmoidal curves to mimic the type of activity curve obtained from epPCR experiments and three different exponential decay curves to mimic the data obtained from active site randomization experiments were computationally created (Figure 4.2a and Figure 4.2b). Within the sets of curves, we chose to change the maximum value obtained and the steepness of the slopes. For comparison purposes, the average activity level was kept the same for the three curves of each type. All other simulation input parameters were those derived for this study. Figure 4.2c shows the predicted number of supermutants detected for the epPCR curves. The three curves are very similar, with overlapping error bars, indicating that the variance in the activity distribution does not have a great effect. The same similarity in the prediction is seen in the estimation of false positives and negatives (data not shown). Figure 4.2d shows the expected number of supermutants predicted by the model for the three curves approximating data from an active site randomization experiment. Here, the three curves differ in their predictions much more than with the epPCR curves. Curve C, which has more colonies with zero activity level and a higher activity level of improved mutants, predicts far more supermutants than the other two curves; curve B is intermediate. The negligible effect of the shape of the curve in epPCR experiments is expected since all three curves have a large mass of activities equivalent to wild-type and completely inactive mutants. These are most often sampled in the random allocation of cells to wells, regardless of the other mutants present. In the case of the exponential decay

curves, there are very few identical activities, other than the inactive mutants, so the individual identities of the cells have a greater effect on the number of supermutants obtained.

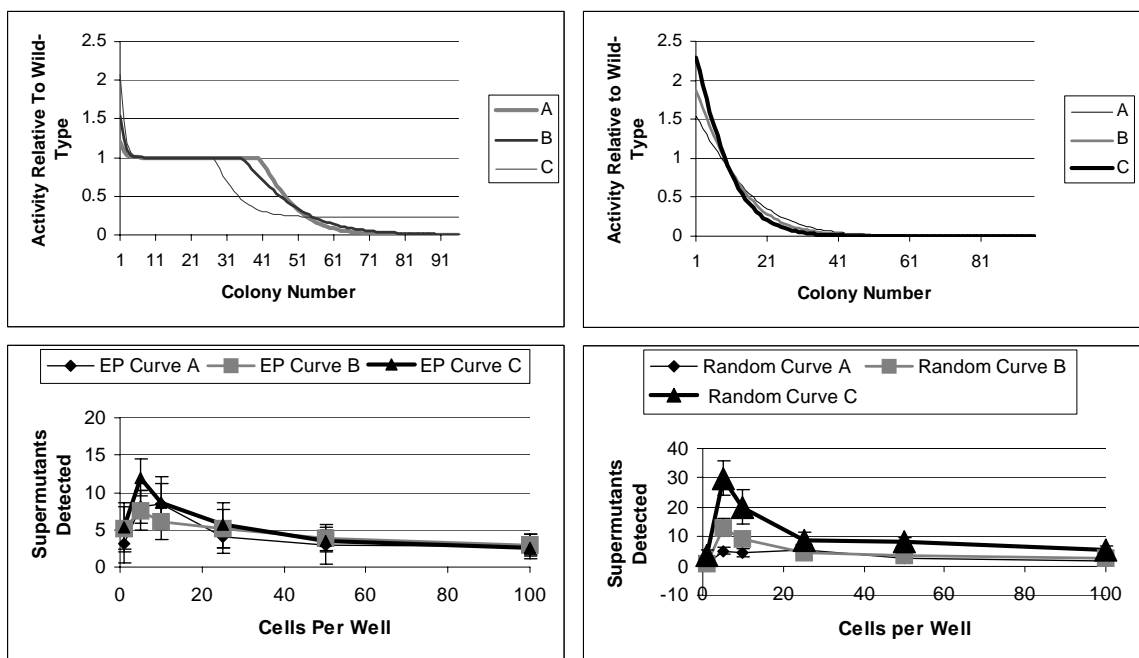


Figure 4.2: Analysis of dependence of model on shape of activity distribution. Top left: a) Three sigmoidal curves used to simulate epPCR experiments. Top right: b) Three exponential decay curves used to simulate active site randomization experiments. Other parameters used as input: activity ratio (5), assay coefficient of variation (24%), threshold for detection of mutant (5), supermutant:dud ratio (1/8192), number of plates per stage (13), all results are the average of 10 simulation experiments. Bottom left: c) Number of supermutants predicted by the model with each of the epPCR curves. Bottom right: d) Number of supermutants predicted by the model with each of the active site randomization curves.

4.3.2 Analysis of the Interdependence of Assay Accuracy and Activity Ratio

Two important factors in the ability to detect improved mutants while pooling are the increase in activity level of the supermutant (activity ratio) and the accuracy of the assay. These are highly interdependent. For example, if an assay has no error in detection (coefficient of variation, $CV=0$), then the supermutant can always be detected regardless of the level of improvement in activity. On the other hand, if the supermutant

has an infinite level of improvement then it will always be detected even if the assay has a high coefficient of variation.

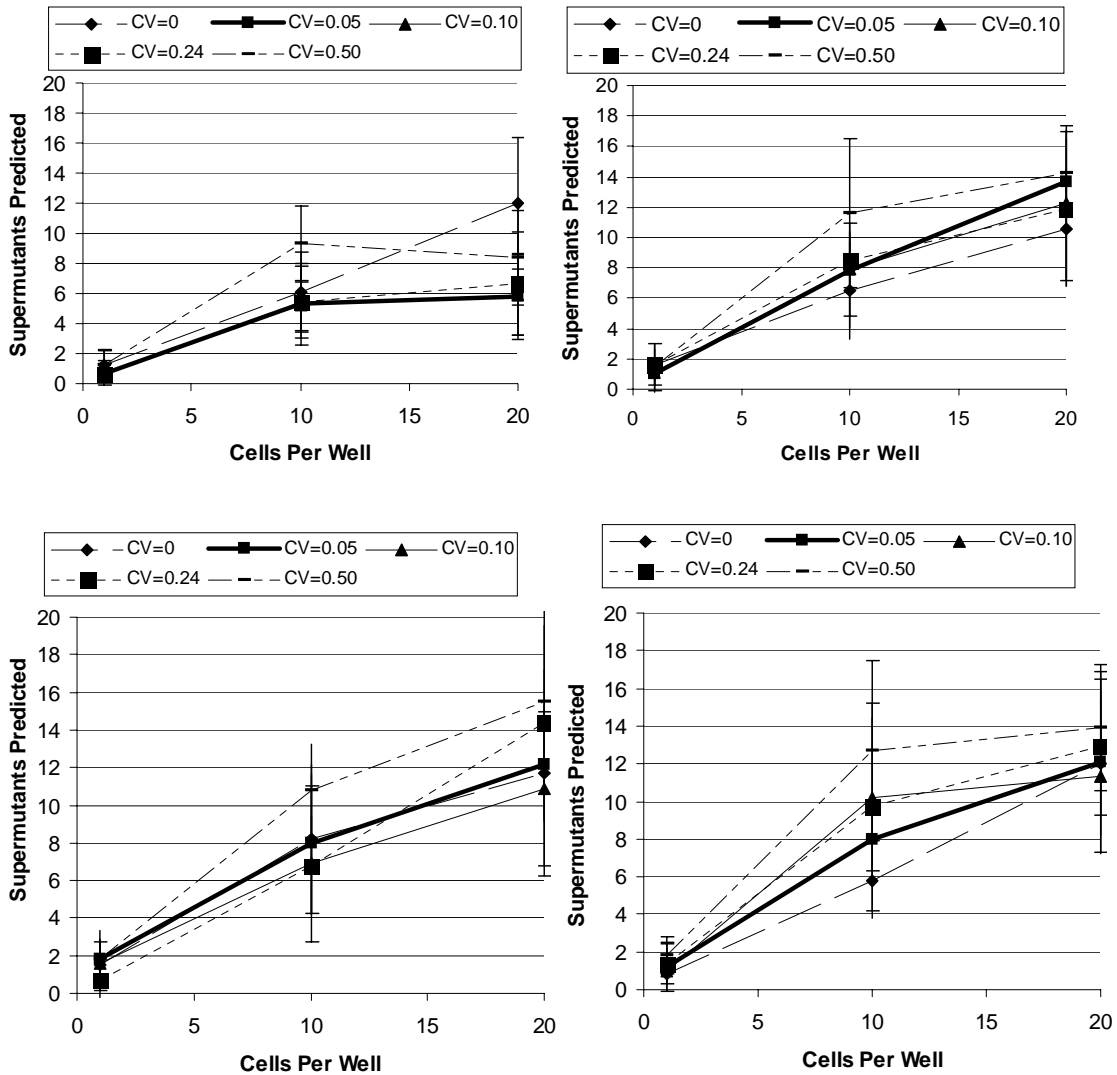


Figure 4.3: Predicted number of supermutants at varying activity ratio and coefficient of variation. Top (left to right): Activity ratio of 5, Activity ratio of 50 Bottom (left to right): Activity ratio of 500, Activity ratio of 5000. Each point represents an average of 10 simulations.

Figure 4.3 shows the simulated number of supermutants detected at activity ratios ranging from 5 to 5000 and CVs from 0 to 50%. At a CV of zero (long dashed line), the assay has no error, so false positives and negatives do not exist. This represents the true

number of supermutants present for that experiment. At the lowest activity ratio and 20 cells per well, the model underpredicts the number of supermutants, at all non-zero values of CV, as a result of a large number of false negatives. At the next level of activity ratio and above, the number of false negatives diminishes to below one in all cases because the threshold is much lower than the activity of the improved mutant and the error in detection of the true number of supermutants is due to the detection of false positives. CVs of 5, 10, and 24% give similar predictions at all activity ratios. The highest CV of 50% tends to lead to overprediction of the number of supermutants present at all activity ratios.

4.3.3 Threshold for Detection

When using the GREATER_THAN function, it is necessary to set a threshold for detection of the supermutant. Lower thresholds will mean fewer false negatives, but more false positives, while higher thresholds have the opposite effect. In general, false negatives are more detrimental in the screening of a directed evolution library than false positives. For simulation of detection of the supermutant in our library we chose to compare two cutoff points, a threshold of five, which is equal to the activity ratio of the supermutant to the wild-type, and a threshold of four to allow for detection of supermutants when the assay error causes a decrease in signal from the supermutant.

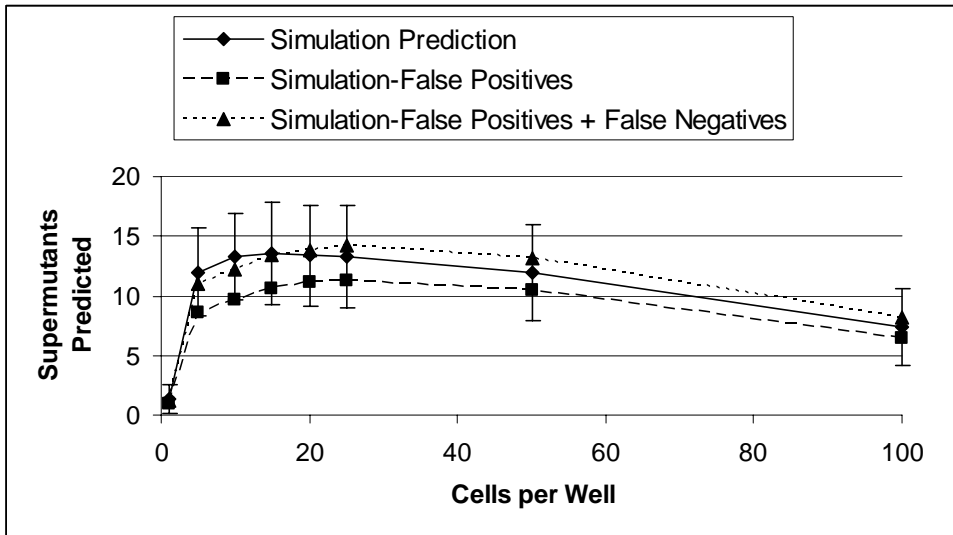
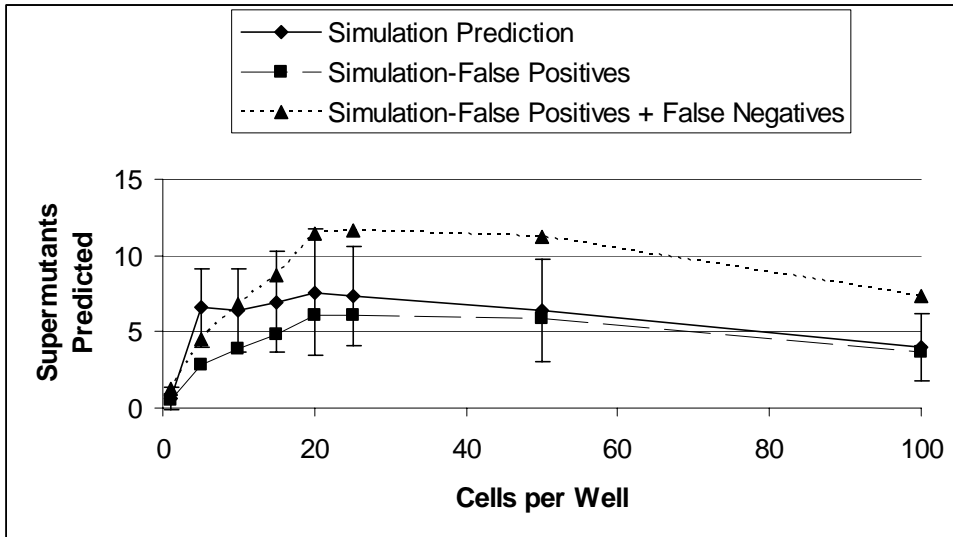


Figure 4.4: The influence of the threshold for detection in the GREATER_THAN function. Top: Threshold of 5. Bottom: Threshold of 4. The data points are the mean of 100 simulations.

Figure 4.4 illustrates the effect of changing the threshold from five to four. At a threshold of five (top curve) there are a substantial number of false negatives. The number of false negatives decreases significantly when the threshold is lowered to four (bottom curve), but the number of false positives also increases. The decline in the number of supermutants predicted at 50 and 100 cells per well is the result of the capacity

of the second stage. Only a certain number of wells can be sampled from the first stage to the second because the number of wells in the second stage is fixed. Since each well must be sampled enough times to retrieve all mutants present (example at least 50 times for 50 cells per well), at very high pooling levels some wells that contain a supermutant are left behind once the second stage fills up.

4.3.4 Prediction of Number of Supermutants Detected with Fucosidase Library

As an equivalent to the filter-lift screening done by Parikh and Matsumura, we screened 26 384-well plates, approximately 10,000 colonies, without pooling to create a baseline for comparison. Ten wells turned visibly yellow and were picked for confirmation. In the secondary assay, three of the 10 had activity that was equivalent to the supermutant control, three had activity that was higher than wild-type LacZ, but not as high as the supermutant, and four had activity equivalent to the wild-type.

Based on the simulation results in Figure 4.4, we chose to pool 10 cells per well to validate the model prediction. After the first stage assay, 26 wells turned visibly yellow and were picked for the secondary assay. Following the second stage, all 26 wells contained a mutant with wild-type or higher activity and the majority of wells (85%) had several active variants. If variants are identified singularly, a total of 8 supermutants, 23 improved mutants, and 26 wild-type equivalents were obtained after the secondary screen. If the wells are identified by the mutant with the highest level of activity in them, 8 wells contained supermutants, 14 wells contained improved, but not super-, mutants and 4 wells contained wild-type equivalents.

Sequencing data was used to refine the identification of improved mutants (Table 4.1). In the unpooled condition, the six mutants with activity higher than wild-type were

sequenced. Three of these were identified as supermutants by the secondary assay. All three had sequences different than wild-type at the three randomized positives, two had sequences identical to the supermutant (D201/H540V/N604T) identified by Parikh and Matsumura.⁹ Three additional wells from the unpooled assay were identified as improved, but not supermutants. Upon sequencing, however, these wells actually contained wild-type enzyme, meaning that they were false positives identified in the unpooled screen.

Table 4.1: Sequences of mutants identified in the 384-well plate screen.

Unpooled Screen

Secondary Assay Identification	D201	H540	N604	Comment
Supermutant	D	H	P	New Sequence
Supermutant	D	V	T	Best mutant from ref 9
Supermutant	D	V	T	Best mutant from ref 9
Improved	D	H	N	False Positive
Improved	D	H	N	False Positive
Improved	D	H	N	False Positive

Pooled Screen

Secondary Assay Identification	D201	H540	N604	Comment
Supermutant	D	A	S	New Sequence
Supermutant	D	E	T	New Sequence
Supermutant	D	E	T	New Sequence
Supermutant	D	V	T	Best mutant from ref 9
Supermutant	D	H	N	False Positive
Supermutant	D	M	E	New Sequence
Supermutant	D	E	T	New Sequence
Supermutant	D	A	S	New Sequence
Improved	D	V	T	Best mutant from ref 9
Improved	D	H	N	False Positive
Improved	D	H	N	False Positive
Improved	D	H	N	False Positive
Wild-Type	D	H	N	Confirmed WT
Wild-Type	D	H	N	Confirmed WT

From the pooled experiment, eight wells were identified as having contained supermutants at the end of the secondary assay. The variants from these eight wells were all sequenced. Of these, one was the wild-type enzyme (a false positive), one contained the D201/H540V/N604T supermutant, and the additional six wells contained other non-wild-type sequences. Of the 14 improved, but not supermutants, a random sample of four variants was sequenced. Three of these were wild-type enzyme false positives, the fourth was an improved enzyme. Two of the four wells identified as having contained wild-type enzyme were confirmed to be equivalent to wild-type by sequencing.

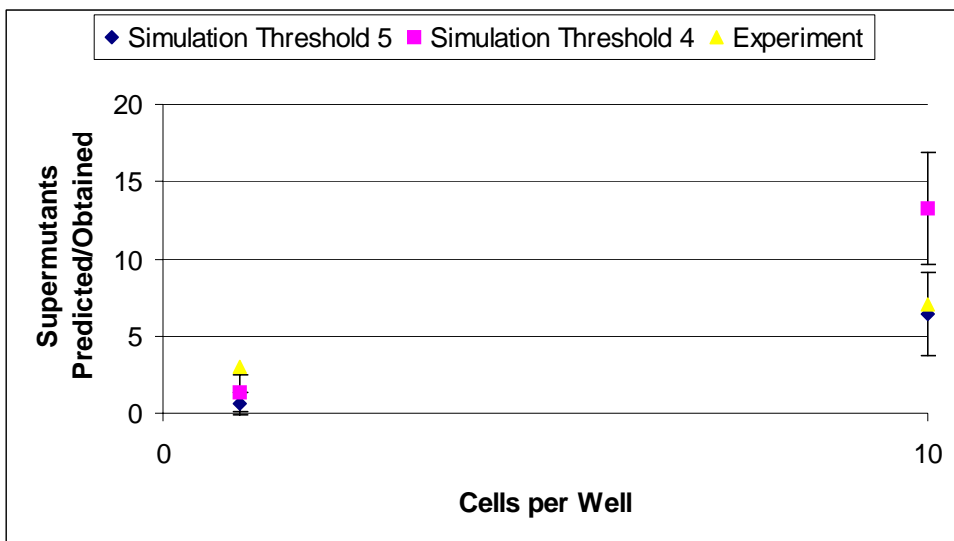


Figure 4.5: A comparison of the predicted and experimentally obtained number of supermutants. The simulation predictions using a threshold of 5 and a threshold of 4 are both included for comparison purposes. The experimental results have been adjusted for false positives using sequencing data.

Figure 4.5 compares the number of supermutants predicted by the simulation model with those obtained through experiments at 1 and 10 cells per well. The model

accurately predicted the number of supermutants obtained through pooling when a threshold of five was used for the GREATER_THAN function. When four was used as the threshold, the model overpredicted the number of supermutants that would be detected, largely because of the presence of about twice as many false positives with the lower threshold. The number of supermutants obtained without pooling is higher than predicted.

4.4 Discussion

We have shown that it is feasible to use our Monte-Carlo simulation model of the pooling process to guide screening experiments with an actual directed evolution library. The library, which was randomized in three active site residues, has a broad spectrum of activity levels, which prohibits the direct calculation of the probability of finding a particular mutant. Monte-Carlo simulation methods were developed specifically for situations such as this, where complexity prohibits direct calculation. Additionally, we have shown that our model can be extended to screening cases that require an induction step to express the enzyme being engineered. Currently, this represents the vast majority of cases, although systems that allow for constitutive expression have been developed.¹¹ In our case, the induction step adds variability to our assay accuracy (a coefficient of variation of 24% versus 10%), but we were still able to successfully employ the pooling strategy predicted by the simulation model.

4.4.1 Increase in Number of Supermutants Detected Using Pooling

Using pooling we obtained more than twice the number of supermutants (seven) as in the unpooled condition (three). In addition, we attained more diversity in the sequences obtained with pooling (four different enzymes versus two). When trying to

identify variants that were improved over wild-type, but not supermutants, we found a high number of false positives. All three such variants identified in the unpooled screen were false positives, and approximately 75% of those identified when pooling were false positives. Possibly, we set our criteria for improvement (1.2-fold increased activity over wild-type) too low given that the coefficient of variation of our assay was 24%.

4.4.2 When Is Pooling Useful?

Pooling is especially suited to cases where the average background level of activity is very low. We can envision several scenarios that produce such an activity curve. In the particular instance under study, the library was created by randomization of amino acid residues very close to the active site. This results in an extremely sharp activity curve where a large percentage of the mutants are completely inactive. Thus, wells containing good mutants are readily distinguished from those that do not. A second scenario under which one might expect a very sharp activity distribution is when a novel function is to be created. Here, the background level of activity is essentially zero. In addition, the creation of new activity is a very rare event, so many, many samples will need to be screened in order to identify those with the created activity. Thus, pooling will be a highly beneficial strategy to increase the efficiency of the process. Finally, error-prone polymerase chain libraries with very high mutation rates also produce the same type of sharp activity distribution. In a recent paper, Chodorge et al published data on mutation rate versus active fraction for green fluorescent protein and *Candida antarctica* lipase B.¹² These could be used as estimates to guide inputs to our pooling model.

4.4.3 Estimating the Parameters

4.4.3.1 Activity Distribution

The activity distribution can be measured by performing a small preliminary screen. In the case of error-prone polymerase chain reaction curves, errors in the estimation of the distribution will have little effect on the predicted outcomes. For active site randomization, the impact of errors is higher, however if a random sample is taken, this should suffice for prediction purposes.

4.4.3.2 Supermutant to Dud Ratio

An additional benefit of diversity generation by active site residue randomization is that it provides a simple way to calculate the frequency of occurrence of each mutant, thereby allowing you to input the expected value of the rarest mutant as your supermutant to dud ratio. This is a much easier parameter to estimate than the true supermutant to dud ratio, which may not be known for a particular experimental system unless prior experiments have been conducted. In the case of variants generated by error-prone polymerase chain reaction or by DNA recombination protocols, the frequency of the most rarely occurring mutants can also be calculated using formulas such as those derived in Patrick et al.¹³

4.4.3.3 Activity Ratio

Another parameter that needs to be estimated is the activity ratio. In cases where prior engineering of the same protein has been done, this parameter can be estimated from these experiments. However, in the case where this is the first attempt to evolve the protein, a substitute measure, the minimum activity level that is an acceptable outcome, may be used.

4.4.3.4 Threshold for GREATER_THAN Function

The threshold of detection for the GREATER_THAN function can be raised and lowered to aid in choosing a pooling level. In general, the higher thresholds yield more false negatives and lower thresholds yield more false positives.

One additional note in the optimization of the pooling level is that the more false positives the experimenter is willing to accept, the more cells that can be pooled into each well. In early rounds of evolution, it might be beneficial to accept more false positives so that a larger portion of the library can be screened in order to find very rare mutants with the activity of interest. On the other hand, there may be less tolerance for false positives in later rounds when fine-tuning of the protein properties is the most important factor.

4.4.4 Extension of Model to Other Disciplines

The Monte-Carlo simulation model of pooling that we developed is adaptable for design of experiments in other disciplines as well. In the example of pooled blood samples being tested for a foreign agent, the situation is analogous to the search for a supermutant among the population using the SUPERMUTANT function. The inputs here would be the estimated frequency of occurrence of tainted blood (such as a prevalence rate), the level of signal achieved by a positive sample, the number of samples that will be pooled together, and an estimate of the variation in the assay procedure. Similarly, a combinatorial library of chemicals derived from a lead structure is similar to a library of enzymes being evolved towards higher function. The inputs would be similar to those required in this study. In such experiments, often the level of activity of the base compound is already known. Therefore, one could use the GREATER_THAN function

to screen for compounds with higher activity than the lead. In this situation, the activity distribution could be determined by a small-scale assay of a handful of compounds.

4.5 Conclusions

Pooling is a means to increase the efficiency of screening experiments by increasing the throughput (and by consequence allowing for sampling of more diversity), or by extending a limited screening capacity to analyze a larger library. The evolution of β -galactosidase to β -fucosidase by Zhang et al required seven rounds of evolution with screening of 10,000 colonies per round.¹⁰ With a conservative estimate of three days per round of diversity generation followed by two days per round of screening, the total time estimate for the study was 35 days. In contrast, Parikh and Matsumura were able to achieve higher activity in one round of evolution with screening 10,000 colonies, for an approximate total time of five days.⁹ In the same amount of time, and with an equivalent amount of effort, we were able to screen five times as many colonies using pooling in 384-well plates. The gain in screening capacity could have been further increased if we had allowed an unequal distribution of capacity across the two stages (please see Chapter VIII for further discussion).

REFERENCES

1. Venton, D. L.; Woodbury, C. P., Screening Combinatorial Libraries. *Chemometrics and Intelligent Laboratory Systems* **1999**, 48, 131-150.
2. Bajpai, M.; Adkison, K. K., High-throughput screening for lead optimization: a rational approach. *Current Opinion in Drug Discovery & Development* **2000**, 3, (1), 63-71.
3. Wehrli, D.; Chen, P., Thales Technologies AG: Research services for the chemical industries. How to find the needle in the haystack and how to find it FAST. *Chimia* **2003**, 57, (6), 354-357.
4. Bruno, W.; Knill, E.; Balding, D.; Bruce, D.; Doggett, N.; Sawhill, W.; Stallings, R.; Whittaker, C.; Torney, D., Efficient Pooling Designs for Library Screening. *Genomics* **1995**, 26, 21-30.
5. Cai, W.-W.; Chen, R.; Gibbs, R. A.; Bradley, A., A Clone-Array Pooled Shotgun Strategy for Sequencing Large Genomes. *Genome Research* **2001**, 11, (10), 1619-23.
6. Andersson, S.; Gessain, A.; Taylor, G. P., Pooling of Samples for Seroepidemiological Surveillance of Human T-cell Lymphotropic Virus Types I and II. *Virus Research* **2001**, 78, (1-2), 101-106.
7. Brookmeyer, R., Analysis of Multistage Pooling Studies of Biological Specimens for Estimating Disease Incidence and Prevalence. *Biometrics* **1999**, 55, (2), 608-612.
8. Polizzi, K. M.; Spencer, C. U.; Dubey, A.; Matsumura, I.; Lee, J. H.; Realff, M. J.; Bommarius, A. S., Simulation modeling of pooling for combinatorial protein engineering. *Journal of Biomolecular Screening* **2005**, (accepted).
9. Parikh, M. R.; Matsumura, I., Site-saturation mutagenesis is more efficient than DNA shuffling for the directed evolution of beta-fucosidase from beta-galactosidase. *Journal of Molecular Biology* **2005**, (accepted).
10. Zhang, J.-H.; Dawes, G.; Stemmer, W. P., Directed evolution of a fucosidase from a galactosidase by DNA shuffling and screening. *Proceedings of the National Academy of Sciences* **1997**, 94, 4504-4509.
11. Matsumura, I.; Olsen, M.; Ellington, A., Optimization of Heterologous Gene Expression for In Vitro Evolution. *BioTechniques* **2001**, 30, (3), 474-476.

12. Chodorge, M.; Fourage, L.; Ullmann, C.; Duvivier, V.; Masson, J.-M.; Fefevre, F., Rational strategies for directed evolution of biocatalysts- Application to *Candida antarctica* lipase B (CALB). *Advanced Synthesis and Catalysis* **2005**, 347, 1022-1026.
13. Patrick, W. M.; Firth, A. E.; Blackburn, J. M., User-friendly algorithms for estimating completeness and diversity in randomized protein-encoding libraries. *Protein Engineering* **2003**, 16, (6), 451-457.

CHAPTER 5

DEVELOPMENT OF CLONING VECTORS TO AID IN HIGH THROUGHPUT SCREENING

5.1 Introduction

In many cases, the objective of expressing a protein is to purify and characterize it to learn its function or to use it as a catalyst for a desired reaction. When doing so, the goal is to obtain a very high level of expression so that the highest possible yield is obtained from each experiment. The protein may also be affinity tagged for easy purification. The culture volume in such an experiment are in general very large, on the order of liters, and the organisms are grown to very high cell densities often using baffled flasks for high levels of aeration. Most commercially available expression plasmids are developed with this purpose in mind. In contrast, protein expression for high throughput screening takes place in well-plates which contain small culture volumes, on the order of microliters, and may be poorly aerated. The goal here is also different—to make measurements of activity as quickly as possible and with the highest amount of information garnered in each measurement. Thus, the toolbox for protein expression differs for high-throughput applications and requires the development of expression vectors geared towards this purpose. The goal of this work is to assemble a collection of vectors formulated for the high throughput screening of directed evolution libraries.

5.2 Discussion of Vector Features

5.2.1 Basic elements necessary for protein expression

5.2.1.1 Origin of Replication

All expression plasmids contain certain necessary elements. First and foremost, the plasmid must have an origin of replication which directs its own copying and propagation to future generations. The origin also controls the frequency of replication, and therefore, the average number of plasmid copies that are maintained within the cell. A high-copy plasmid is one that is present in 100 copies or more within the cell, while a low-copy plasmid is usually maintained at less than 5 copies per cell. Intermediate copy numbers are called medium-copy. While higher copy number results in higher protein production, high-copy plasmids represent a large metabolic burden on the cell, and as a result can be highly unstable.¹

5.2.1.2 Promoter and Operator

All expression vectors also require a mechanism to initiate transcription, the precursor to protein production. The promoter is a particular nucleotide sequence positioned 10 to 100 base pairs upstream of the ribosome binding site. In *E. coli*, most promoters contain two conserved hexanucleotide sequences, one at -10 bp and the other at -35 bp from the start of transcription. The regulation of the promoter (on/off switch) is controlled by the operator region, which is recognized by repressors (turn off transcription) and inducers (turn on transcription).²

Most traditionally used expression vectors contain a lac promoter/operator derived from the *E. coli* metabolic machinery for lactose metabolism. The main disadvantage to this system is that it is part of the innate cellular metabolism. This results

in a high level of background, or “leaky” transcription resulting from incomplete repression because some of the naturally produced repressor is used to control the naturally occurring operon. In addition, full scale induction of this promoter to produce protein also results in induction of the naturally occurring genes, and results in unwanted metabolic changes to the cell. An alternative, orthogonal, promoter/operator system is derived from the (Tn10) tet operon. The tet system is characterized by extremely tight repression, followed by linear induction in response to inducer over a 5000-fold range.³ In addition, the best inducer for this promoter/operator is anhydrotetracycline, which can be economically produced by autoclaving tetracycline at low pH.⁴

5.2.1.3 Constitutive vs. Inducible Expression

Constitutive protein expression is very important in high throughput screening systems because an induction step requires additional handling which decreases the efficiency of the screen. The ability to vary the level of expression as evolution progresses is also of interest. This allows for high levels of expression in early rounds of directed evolution when the level of activity is low (thereby increasing the overall signal) tapering to lower levels of expression in later rounds so that small increases in an activity can still be detected (increases the dynamic range). The level of expression can also be tuned to modulate toxicity of the protein being expressed and to decrease overall metabolic burden.⁵

One method to create a library of vectors with varying levels of expression is to randomize portions of the promoter sequence. One such approach was undertaken by Matsumura et. al., who randomized six conserved nucleotides in the – 10 region of the lac

promoter in pET20, which resulted in a spectrum of expression levels ranging over three orders of magnitude.⁵

5.2.1.4 Ribosome Binding Site

Finally, in order to have translation of the gene of interest into a protein, each plasmid must contain a ribosome binding site for initiation of protein production. Though there is some evidence that the level of expression changes with the particular sequence of the ribosome binding site and the spacing of the initiation codon of the protein from it, most commercial vectors contain a standardized ribosome binding site with a 5 to 13 nucleotide spacer. This is sufficiently optimized for protein production in most cases.²

5.2.1.5 Base Vector to Provide These Elements

We chose to use the commercially available pPROTet vector as a base for constructing the high throughput screening vectors. The vector has a ColE1 origin, which maintains a high copy number, a tet promoter/operator, and a ribosome binding site. In this case, the high copy number should not cause toxicity problems due to the tunability of expression level in response to the inducer anhydrotetracycline. The multiple cloning site allows for cloning of a gene with an N-terminal 6XHN tag (alternating histidine and asparagine), however, this was removed during vector construction. We also explored the idea of trading the tet promoter/operator, for a lac system that was randomized in the operator region to produce variable levels of constitutive expression.

5.2.2 Selection for Plasmid Retention

For efficient overexpression of the protein of interest, it is necessary to have some mechanism which causes the plasmid to be retained by the cell. The same mechanism

can also be used to select for colonies containing the plasmid in the initial cloning step. Plasmid loss is extremely detrimental to high density fermentation systems and is more likely to occur with high-copy plasmids or when the expression of the gene is toxic or interferes with cellular growth.⁶

5.2.2.1 Antibiotic Resistance Genes

The most widespread method to ensure plasmid uptake and retention is the use of a plasmid-borne antibiotic resistance marker, which confers resistance to an antibiotic of choice when the plasmid is present in the cell. The main problem with such a selection method is that antibiotics are degraded and/or inactivated by these enzymes, which leads to reduced pressure to maintain the plasmid as the fermentation progresses.⁷ In addition, the promoter systems controlling the antibiotic resistance genes are constitutively active, usually at a very high level, causing a high metabolic burden on the cell.⁸ Thus, when the selective pressure is diminished, it is advantageous for the cell to release the plasmid.

5.2.2.2 Growth Inhibitors

To construct our vectors, we focused on the use of a cellular growth inhibitor to drive plasmid maintenance. The main advantage here is that the bacteria survive via an alternate metabolic gene provided on the plasmid which is not sensitive to the inhibitor, thus the inhibitor is not degraded or inactivated by the mechanism of selection. This means a constant concentration of inhibitor will be present throughout the fermentation, maintaining the selection pressure, and also the advantage for the host cell to maintain the plasmid.

5.2.2.3 Triclosan

Triclosan, the active ingredient in antibiotic soaps and in toothpaste, is an inhibitor of the lipid biosynthetic enzyme enoyl-acyl carrier protein reductase (ACP reductase) of *E. coli*. Triclosan is effective at a concentration of less than 10 ppm, thus only a low level of inhibitor is necessary to maintain selective pressure. Variants of ACP reductase from some other organisms are not inhibited by triclosan and a cloning vector containing the *Bacillus subtilis* ACP reductase and its natural promoter has been demonstrated to confer resistance to triclosan up to 10,000 ppm when expressed in *E. coli*.⁹ We focused on using this enzyme to maintain plasmid selection.

5.2.3 Mechanism of DNA Insertion

All vectors must have a mechanism to insert the DNA of interest. Most often, this is done by using restriction enzymes to digest both the vector and the DNA fragment to produce complementary ends that are ligated together. Most modern vectors have a region, called the multiple cloning site, which is rich in sequences recognized by common restriction enzymes. One of the limits of traditional cloning is that the restriction enzymes chosen must only cut the DNA of interest in the intended place and since there are a limited number of reliable restriction enzymes some genes can be very problematic to clone. Additionally, the efficiency of insertion of DNA by traditional methods is affected by the efficiency of each of the enzymatic steps (digestion and ligation), which can result in a high background level of plasmid that does not contain the DNA of interest when improper digestion or religation of empty vector occurs.

5.2.3.1 Ligation-Independent Cloning

Ligation-independent cloning (LIC) is a method of gene insertion that does not rely on restriction enzyme digestion and ligation. Rather, complementary 12-18 bp overhangs are generated on both the insert and the vector. These are allowed to anneal and the vector is then transformed into cells where repair of the nicks occurs to create a fully circularized plasmid. The overhangs can be generated in several ways. Commercial LIC vectors rely on the 3' to 5' exonuclease activity of T4 DNA polymerase to generate the overhangs. With this method, a sequence rich in three of the four nucleotides is required and incubation of the DNA with the enzyme and the fourth nucleotide results in base excision until the nucleotide is encountered, followed by an equilibrium of insertion and deletion which controls the length of the overhang.¹⁰ Another approach is to use a short digestion with exonuclease to create the overhangs, though this can result in a spectrum of overhang sizes that reduces the efficiency of cloning.¹¹ Finally overhangs can be generated directly during PCR by use of hetero-staggered primers¹², incorporating RNA bases into a primer which are not copied by the polymerase¹³, or truncation of the PCR using an atypical nucleotide.¹⁴

5.2.4 Expression Reporters

Visualization of the ability of an individual clone to express the protein of interest, as well as a crude measurement of the level of that expression can be obtained using a fluorescent protein as an expression reporter. Several such proteins exist. The most well known is the green fluorescent protein (GFP) a non-enzymatic protein found in jellyfish such as the species *Aequorea*. Depending on the variant in question, GFP has a range of excitation values, from the long-range ultraviolet through around 490 nm, and an

emission wavelength in the green area, usually between 504-511 nm. Various mutants of GFP have also been created, these include cyan, blue, and yellow fluorescent versions with emissions wavelengths in the areas of their respective colors.¹⁵ The use of GFP as an expression reporter in an operonic system (i.e. transcriptionally fused, but translationally distinct)¹⁶ and as a fusion protein with a protein of interest¹⁷ has been reported.

Another class of fluorescent proteins is derived from the *Discosoma* species of coral, and fluoresces in the red region, around 600 nm. The original protein, DsRed, is a tetrameric enzyme¹⁸, but monomeric variants of this protein have been evolved via directed evolution.¹⁹ In addition, the monomeric red fluorescent protein has been evolved to a rainbow of colors, ranging from red to green.²⁰

5.2.5 Folding Reporters

Selection for proper folding can be achieved by the C-terminal fusion to a protein that can report the state of the protein of interest. The premise is that in order for the reporter protein to fold, the protein of interest must be properly folded.

5.2.5.1 Fluorescent Proteins

The use of GFP as a folding reporter was studied extensively by Waldo et al, who showed via studies with 20 different proteins that the level of GFP fluorescence could be correlated with the likelihood that the protein of interest would aggregate.²¹

5.2.5.2 Antibiotic Resistance Markers

An additional class of proteins that can be used as folding reporters are genes conferring resistance to an antibiotic. These can be used to select for properly folded mutants by plating on medium containing the antibiotic. Those transformants containing

properly folded proteins will be resistant, and as a consequence, will grow, whereas those with misfolded proteins will not be resistant and will die. One example used the chloramphenicol acetyl-transferase gene to select for mutants of HIV integrase that were more soluble.²²

5.2.5.3 Intein-Mediated Fusions

Finally, one strategy that allows for fusion tag monitoring of folding, followed by purification of the protein of interest without purification of the reporter protein is to use an intein-mediated fusion. Inteins are self-excising protein elements which will cleave themselves out of proteins when some chemical or physical cue is received. Some inteins splice the protein halves together as they are cleaved, but modified inteins have been created which excise without splicing, thereby leaving the protein of interest free in solution. This strategy has been used with GFP and can include a purification tag such as a hexahistidine or chitin-binding domain which can be used to trap the protein of interest for purification following cleavage.⁷

5.3 Materials and Methods

5.3.1 Materials

Molecular biology reagents (restriction enzymes, ligase, polymerases) were obtained from Fermentas (Hanover, MD) through VWR International (Suwanee, GA). Primers were synthesized by MWG Biotech (High Point, NC). The pPROTet 6XHN and GFPuv vectors were purchased from BD Biosciences Clontech (Mountain View, CA). The SuperGlo™ vector was obtained from Qbiogene (Irvine, CA). The pET32 Ek/LIC cloning kit was purchased from Novagen (San Diego, CA). The pBBRT vector was a kind gift from Dr. Anthony Hay (Cornell University, Ithaca, NY)⁹. The Qiaprep Spin

Miniprep Kit and Qiaquick Gel Extraction Kit were purchased from Qiagen (Valencia, CA). Fluorimetric measurements were made using a Fluorostar plate reader (BMG Technologies, Offenburg, Germany). The rapidly maturing DsRed variant was a kind gift from Dr. Benjamin Glick (University of Chicago, Chicago, IL)¹⁸. The mRFP gene was synthesized and cloned by fellow lab member Bernard Loo.

5.3.2 Test of Ligation-Independent Cloning Efficiency

The basic experimental procedure for LIC using T4 DNA polymerase to generate the overhanging ends was tested using the pET32 Ek/LIC cloning kit from Novagen according to the manufacturer's instructions using DsRed as a test insert. The experimental annealing reaction consisted of 1 μ l of linearized plasmid provided with the kit, and 2 μ l of the prepared insert. A control experiment was set up using the linearized vector provided with the kit and 2 μ l of water. Following annealing and transformation, 40 μ l of both annealing reactions were plated on LB media containing ampicillin to select for the plasmid. The resulting colonies were minipreped and analyzed by restriction digest to determine the presence of insert.

5.3.3 Insertion of Triclosan into pPROTet

The chloramphenicol resistance gene on pPROTet was replaced with the ACP reductase gene from the pBBRT vector using overlap extension PCR. Two fragments were generated by PCR, the ACP reductase gene including its promoter, and the pPROTet vector minus the chloramphenicol acetyl transferase gene and its promoter. The fragments were gel purified and mixed in equimolar ratios for the primerless extension PCR. The product of the overlap extension reaction was transformed directly

into DH5 α (PRO) cells and plated onto LB containing 2 μ g/ml triclosan to select for successful transformants. Insertion of the triclosan gene was confirmed by sequencing.

5.3.4 Design of LIC Inserts

Novel LIC sites were designed using VectorNTI (Invitrogen, Carlsbad, CA) and synthesized from complementary primers by melting the DNA at 95 °C and reannealing by slowly bringing the temperature down to 25°C. The DNA fragments were stored at -20°C.

5.3.5 Randomization of the lac operator

Whole plasmid amplification by PCR using randomized primers was performed using the Roche High Fidelity Long-Range PCR kit on DsRed-pET32 according to the manufacturer's protocol. A good concentration of a single product of the correct size was obtained. The amplified plasmids were transformed into electrocompetent *E. coli* DH5 α . Colonies from the transformation were picked into single wells of a 96-well plate containing 200 μ l of LB plus ampicillin and supplemented with 30% glycerol using sterile toothpicks. The plates were grown at 30 °C for 48 to 72 hours and expression was tested by fluorimetry.

5.3.6 Fluorescent Reporter Tests

The DsRed and GFPuv genes were amplified via PCR and cloned into the pET32 Ek/LIC vector. The plasmids pET32-DsRed, pET32-GFPuv, and pQBP1-T7-SupergloTM GFP were transformed into *E. coli* DH5 α and their fluorescence tested by induction with IPTG. pPROTet-mRFP was transformed into DH5 α (PRO) and induced with autoclaved tetracycline. GFPuv was excited with a handheld UV light at 365 nm, the other proteins were excited with a full spectrum of light.

5.3.7 Intein-Mediated Fusion Proteins

The intein fusion proteins were designed according to Zhang et al, but substituting mRFP for the green fluorescent protein and including a LIC site for gene cloning (see Figure 5.1).⁷ In addition, the option for a 6XHN-tag as an alternative to the chitin-binding domain was designed.

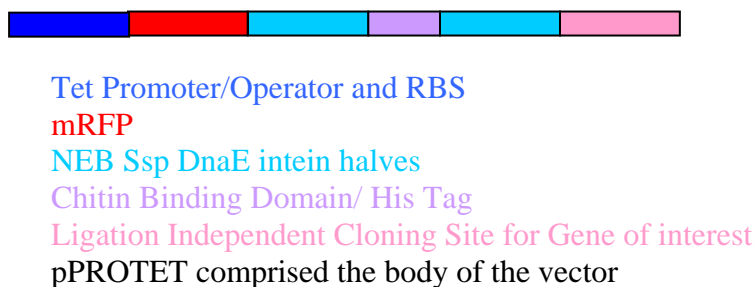


Figure 5.1: Intein-Mediated Fusion Reporter Construct

The individual genes were amplified by PCR and the full construct assembled by overlap extension PCR. The presence of each piece was confirmed by a PCR using primers specific for that portion and the full construct was sequenced to determine whether point mutations were present. The LIC site was linearized by digestion with SspI for 16 hours at 37°C and the overhangs generated by treatment with T4 DNA polymerase and dCTP at 25°C for 1 hour. As a test insert, β -lactamase was amplified by PCR with primers coding for complementary overhangs to the LIC site. The PCR product was gel purified and treated with T4 DNA polymerase and dGTP for 1 hour at 25°C. Two μ l of the treated insert and 1 μ l the treated vector were annealed at 22°C for 30 minutes and the annealing reaction was terminated by addition of 1 μ l of 25 mM EDTA. The annealed vector was transformed into DH5 α (PRO) competent cells. The expression of the mRFP gene was

induced with autoclaved tetracycline both before and after insertion of the β -lactamase gene. The expression level of the fusion construct was evaluated by the fluorescence level of the mRFP and by SDS-PAGE gel.

5.4 Results

5.4.1 Insertion of the Triclosan Gene into pPROTet

The chloramphenicol acetyl transferase gene was successfully replaced in pPROTet by the ACP reductase gene as evidenced by the ability of cells containing the plasmid to grow on medium containing triclosan, but not on medium containing chloramphenicol. Sequencing results also confirmed that the correct gene was inserted.

5.4.2 Test of LIC Efficiency

The efficiency of the LIC procedure was tested using the commercially available pET32 Ek/LIC kit. A negative control reaction using water in place of treated insert was also run. From 40 μ l of transformation reaction plated onto LB medium plus ampicillin, six colonies were obtained. When these were analyzed further, no inserts were obtained, and in the case of 4 of these, the plasmid obtained was not consistent with the pET32 plasmid on the basis of size. From 40 μ l of transformation reaction plated from the annealing reaction containing insert, thirty-nine colonies were obtained. Upon further analyses, 36 of these contained the correct DsRed fragment, for an efficiency of 92%.

5.4.3 Design of LIC Inserts

The first LIC site was designed to allow for the use of a C-terminal 6X Histidine tag with the protein of interest (Figure 5.2). It is linearized by the restriction enzyme *Ssp*I (blue) to begin T4 DNA polymerase treatment and is flanked by *Eco*RI sites (red).

GAATTCATTAAGAGGAGAAAGAGAAATATTCACCATCATCATCATCATGGAATTC
CTTAAGTAATTCTCCTCTTTCTCTTATAAGTGGTAGTAGTAGTAGTACCTTAAG

Figure 5.2: Designed LIC site allowing for C-terminal His-tag option

The second LIC site does not provide for a C-terminal His-tag and has shorter overhangs (Figure 5.3). It is linearized using *SrfI* or *SmaI* (blue) and is flanked by *KpnI* sites (red).

GGTACC GGCCCCGGAGCCCGGGCGGCGGCTTAGGTACC
CCATGGCCGGGGCCTCGGGCCCGCCCGAATCCATGG

Figure 5.3: The second designed LIC site without His-tag

Both LIC sites were assembled and inserted into the pPROTet vector. Linearization was tested using the appropriate restriction enzyme. The *SmaI* enzyme did not digest as efficiently as the *SspI* enzyme, as evidenced by multiple bands being present on the gel following digestion. Additionally, the sequence of the second LIC site is very G/C rich, so further experiments used the first LIC site.

5.4.4 Constitutive Expression by Randomization of the lac Operator

Following the randomization procedure, single colonies were picked to the wells of five sets of 96-well plates and grown at 30 °C. Each plate also contained a medium only control as well as a native plasmid (one which was not randomized by PCR). Fluorometric measurements were taken at 48 hours (Figure 5.4) and then again at 72 hours (data not shown).

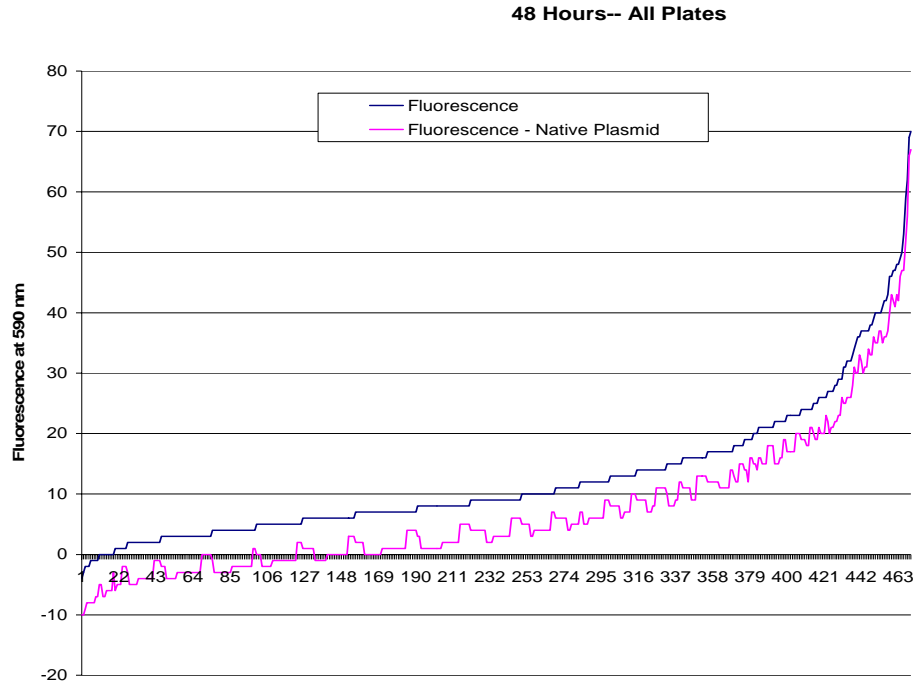


Figure 5.4: Results of variable expression by randomization of the lac operator region

The range of fluorescence values after 48 hours was rather limited (-10 to 70 units), and spanned 1.5 orders of magnitude. This is inferior to the method of randomizing the promoter region.⁵ In addition, the fluorescence obtained was extremely low, indicating a high metabolic burden on the cells to produce protein.

5.4.5 Fluorescent Reporter Tests

Each of the potential fluorescent reporters was successfully expressed in both liquid and solid medium. Figures 5.5a through 5.5d show the visual fluorescence test of DsRed, GFPuv, SuperGloTM, and mRFP induced on solid media, respectively.

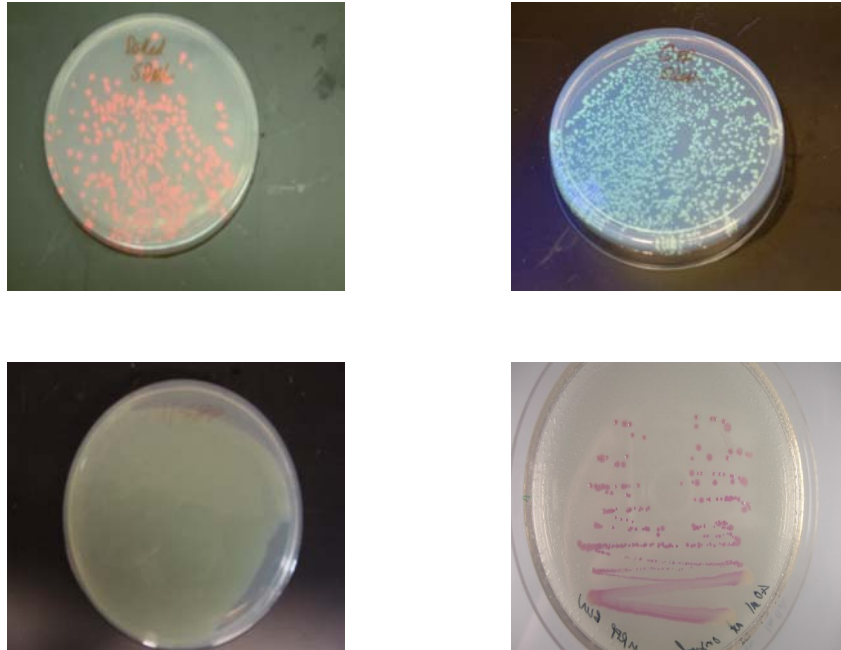


Figure 5.5: Fluorescent reporter tests. Clockwise from top left: a) expression of the DsRed T3 Mutant (Glick) in *E. coli* DH5 α , b) expression of GFPuv (Clontech) in *E. coli* DH5 α , c) Expression of SuperGloTM in *E. coli* DH5 α , d) Expression of mRFP in *E. coli* DH5a(PRO) cells.

5.4.6 Intein-Mediated Fusion Proteins

A diagnostic PCR showed that each piece of the fusion construct was present in more than one of the plasmids picked for analysis (Figure 5.6).

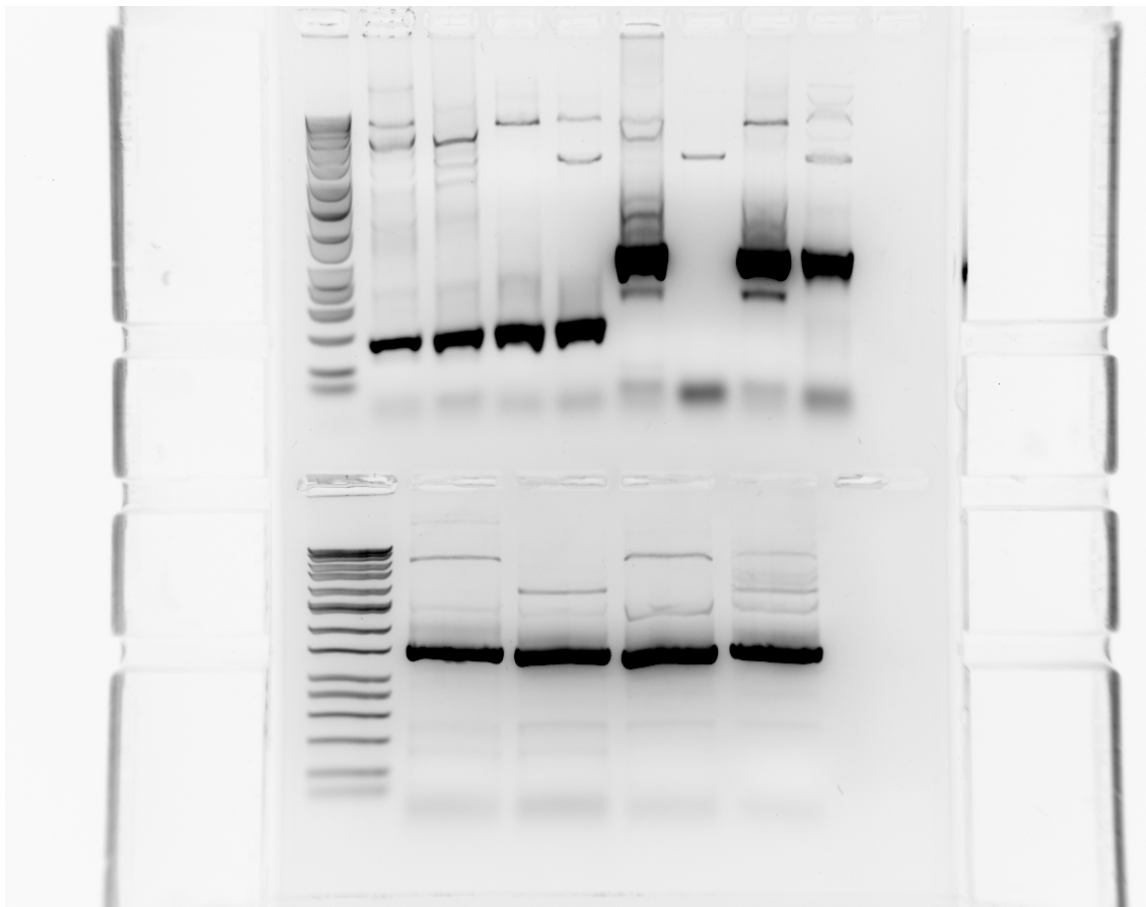


Figure 5.6: Diagnostic PCR of intein fusion constructs. Top from left: Ladder, pKMPa 1-4 PCR to check for chitin-binding domain, pKMPa 1-4 PCR to check for mRFP. Bottom from left: Ladder, pKMPa PCR to check for pPROTet plus triclosan including LIC site.

Unfortunately, none of the colonies were able to visibly express mRFP. Since it was possible that this was due to a protein production issue, as there was no gene of interest cloned in with a stop codon to terminate translation, β -lactamase was cloned in as a test

protein. None of these colonies produced visible mRFP either. Two colonies were overexpressed to check for the presence of protein expression. There is some evidence of insoluble expression, meaning that the fusion protein does not fold correctly.

5.5 Summary and Conclusions

The aim of this work was to develop expression systems that are amenable to high-throughput screening situations found in directed evolution. When proteins are being expressed in a high-throughput manner, the goals and problems differ from expression for purification and characterization. Instead of large culture volumes and high levels of overexpression, high throughput systems require small volumes, stable expression systems, and information density in measurements. It is of interest to tell whether the protein being expressed is folding correctly and the amount of expression is useful for estimating specific activities. Thus, we set out to combine these features into an expression system that would support our directed evolution activities.

The commercially available pPROTet vector was chosen as a base vector because it is small and already contains the tet operator/promoter region, as well as providing the necessary origin of replication. Although high copy plasmids can be toxic to cells, in this case the ability to induce expression at lower levels using anhydrotetracycline mitigates the problem. The antibiotic selection mechanism (chloramphenicol acetyl transferase) was successfully replaced with the triclosan resistance gene using overlap extension PCR. Four fluorescent reporter genes were successfully expressed and one, mRFP, was inserted into pPROTet as an expression reporter. Use of an intein-mediated fusion of mRFP as a folding reporter yielded solely insoluble expression. Recent work has shown that the original mRFP is not very tolerant in fusion proteins, but changing the termini to be more

similar to GFP can increase the tolerance to fusion with another protein.²⁰ This is one possibility for future work.

REFERENCES

1. Keasling, J. D., Gene-expression tools for the metabolic engineering of bacteria. *Trends in Biotechnology* 1999, 17, 452-460.
2. Makrides, S. C., Strategies for achieving high-level expression of genes in *Escherichia coli*. *Microbiological Reviews* 1996, 60, (3), 512-538.
3. Lutz, R.; Bujard, H., Independent and tight regulation of transcriptional units in *Escherichia coli* via the LacR/O, the TetR/O, and AraC/I1-I2 regulatory elements. *Nucleic Acids Research* 1997, 25, (6), 1203-1210.
4. Barton, D.; Magnus, P., Experiments on the synthesis of tetracycline. I. Introduction to the series. *Journal of the American Chemical Society* 1971, 84, 2164-2166.
5. Matsumura, I.; Olsen, M. J.; Ellington, A. D., Optimization of heterologous gene expression for in vitro evolution. *BioTechniques* 2001, 30, 474-476.
6. Baneyx, F., Recombinant protein expression in *Escherichia coli*. *Current Opinion in Biotechnology* 1999, 10, 411-421.
7. Zhang, A.; Gonzalez, S. M.; Cantor, E. J.; Chong, S., Construction of a mini-intein fusion system to allow both direct monitoring of soluble protein expression and rapid purification of target proteins. *Gene* 2001, 275, 241-252.
8. Cranenburgh, R. M.; Hanak, J. A. J.; Williams, S. G.; Sherratt, D. J., *Escherichia coli* strains that allow antibiotic-free plasmid selection and maintenance by repressor titration. *Nucleic Acids Research* 2001, 29, (5), e26-e32.
9. Kagle, J.; Hay, A. G., Construction of a broad host range cloning vector conferring triclosan resistance. *BioTechniques* 2002, 33, (3), 490-492.
10. Novy, R. E.; Yaeger, K. W.; Kolb, K. M., Ligation independent cloning: efficient directional cloning of PCR products. *inNovations* 1996, 5, 1-3.
11. Li, C.; Evans, R. M., Ligation independent cloning irrespective of restriction site compatibility. *Nucleic Acids Research* 1997, 25, (20), 4165-4166.
12. Tillett, D.; Neilan, B. A., Enzyme-free cloning: a rapid method to clone PCR products independent of vector restriction enzyme sites. *Nucleic Acids Research* 1999, 27, (19), e26-e26.

13. Coljee, V. W.; Murray, H. L.; Donahue, W. F.; Jarrell, K. A., Seamless gene engineering using RNA- and DNA- overhang cloning. *Nature Biotechnology* 2000, 18, 789-791.
14. Lutz, S.; Ostermeier, M.; Benkovic, S. J., Rapid generation of incremental truncation libraries for protein engineering using α -phosphotioate nucleotides. *Nucleic Acids Research* 2001, 29, (4), e16-e22.
15. Tsien, R. Y., The green fluorescent protein. *Annual Reviews of Biochemistry* 1998, 67, 509-544.
16. Albano, C. R.; Randers-Eichhorn, L.; Bentley, W. E.; Rao, G., Green fluorescent protein as a real time quantitative reporter of heterologous protein production. *Biotechnology Progress* 1998, 14, 351-354.
17. Cha, H. J.; Wu, C.-F.; Valdes, J. J.; Rao, G.; Bentley, W. E., Observations of green fluorescent protein as a fusion partner in genetically engineered *Escherichia coli*: Monitoring protein expression and solubility. *Biotechnology and Bioengineering* 2000, 67, (5), 565-574.
18. Bevis, B. J.; Glick, B. S., Rapidly maturing variants of the *Discosoma* red fluorescent protein (DsRed). *Nature Biotechnology* 2002, 20, 83-87.
19. Campbell, R. E.; Tour, O.; Palmer, A. E.; Steinbach, P. A.; Baird, G. S.; Zacharias, D. A.; Tsien, R. Y., A monomeric red fluorescent protein. *Proceedings of the National Academy of Sciences* 2002, 99, (12), 7877-7882.
20. Shaner, N. C.; Campbell, R. E.; Steinbach, P. A.; Giepmans, B. N. G.; Palmer, A. E.; Tsien, R. Y., Improved monomeric red, orange and yellow fluorescent proteins derived from *Discosoma* sp. red fluorescent protein. *Nature Biotechnology* 2004, 22, (12), 1567-1572.
21. Waldo, G. S.; Standish, B. M.; Berendzen, J.; Terwilliger, T. C., Rapid protein-folding assay using green fluorescent protein. *Nature Biotechnology* 1999, 17, 691-695.
22. Maxwell, K. L.; Mittermaier, A. K.; Forman-Kay, J. D.; Davidson, A. R., A simple in vivo assay for increased protein solubility. *Protein Science* 1999, 8, 1908-1911.

CHAPTER VI

CLONING AND OVEREXPRESSION OF ENOLASE TEMPLATE ENZYMES

6.1 Introduction

6.1.1 α/β -barrel enzymes

α/β -barrel enzymes are among the most ubiquitous protein folds in nature. Of the currently solved crystal structures of proteins, approximately 10% adopt this structure; the majority of these are enzymes. The range of reactions catalyzed by α/β -barrel enzymes is quite diverse; members of this fold can be found in five of the six enzyme classes (only class six, the ligases, are absent to date). Additionally, a few non-enzymatic α/β -barrel proteins, such as narbonin, a storage globulin, and concanavalin B, a chitin-binding protein, have been identified.²

The classical α/β -barrel enzyme consists of an inner core of 8 β -strands surrounded by an outer core of 8 α -helices (Figure 6.1).¹⁻⁵ In addition, there are quite a few related structures (deemed ‘cousins’⁵) that have added or subtracted helices or strands, but still form a very similar general shape.^{2, 5}

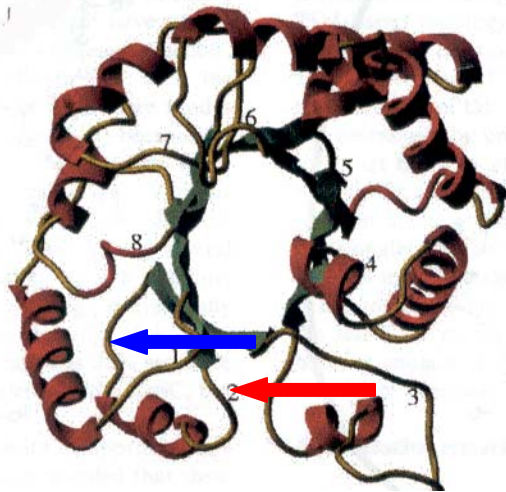


Figure 6.1: The structure of an α/β -barrel enzyme.¹ The blue arrow denotes a loop connecting a beta-strand to an alpha-helix (function). The red arrow denotes a loop connecting an alpha-helix to a beta-strand (stability).

Several common motifs occur in a large proportion of the α/β -barrel enzymes, despite their chemical diversity. The active site varies as to which amino acid residues are involved, but is always located at the C-terminal end of the barrel in the loop regions.¹⁻⁵ Architecturally, the loops responsible for function and those responsible for the stability of the enzyme are separated, with the former being those that connect the end of a β -strand to the beginning of the next α -helix, and the latter being the loops connecting the end of an α -helix to the beginning of the next β -strand (red and blue arrows of Figure 6.1).¹ In addition, approximately two-thirds of the α/β -barrel enzyme families that have been characterized to date have a phosphate binding group. These are used to bind substrates and/or cofactors that contain a phosphate group and are located primarily towards the C-terminus of the protein in β -strand- α -helix pair 7 or 8. Finally, about one-half of the known α/β -barrel enzymes use a metal ion in their catalytic scheme. The location of the residues responsible for coordinating the metal ion do vary from

protein to protein, but may be conserved among closely related members of the same superfamily.²

α/β -barrel enzymes make a good template for protein engineering for several reasons. First, the fact that natural examples exist with very diverse chemistries indicates that almost any function can be supported on this template. Second, since a large portion of the residues are involved in the core structure (i.e. form α -helices and β -strands), these residues can remain untouched in diversity generation steps, limiting the library size to more manageable numbers. Further, since the loop residues responsible for stability and function are physically separated, it should be possible to engineer a different function without decreasing the stability of the fold.

6.1.2 Superfamilies

A superfamily is a group of structurally-related, sometimes functionally diverse, enzymes that catalyze a common reaction or partial reaction step using conserved active site residues.⁶ Typically, members of the same superfamily will have a sequence identity of less than 50%⁷ and very often the identity level is less than 20%.⁶ What unifies members of the superfamily is their conserved chemistry and a similar tertiary structure. In fact, the structures of enzymes of the same superfamily are often so superimposable that a template can be developed based on a handful of structural similarities among a portion of the superfamily that can later be used to identify other members, including putative proteins.⁸ Recent work by Livesay et al seems to suggest that the conserved chemistry is, at least in part, due to conserved electrostatic properties across the protein structure, despite the actual identities of the amino acids involved.⁹

6.1.3 Enolase Superfamily

The enolase superfamily is a group of α/β -barrel enzymes that share a common partial reaction step, the abstraction of an α -proton from a substrate, followed by the formation and stabilization of an enolate anion intermediate. The chemistry that follows the generation of the enolate anion is very diverse, including racemization, elimination of water or ammonia (with different stereoselectivity regarding syn versus anti, depending on case), and cycloisomerization.¹⁰ Members of the enolase superfamily can currently be divided into three subgroups based on the identity of the metal ligand groups and the residues responsible for catalysis: the enolase subgroup, which consists of a large number of enolases, the mandelate racemase subgroup, which includes mandelate racemase and several sugar acid dehydratases, and the muconate lactonizing subgroup, which contains muconate lactonizing enzyme, *ortho*-succinyl benzoate synthase, and N-acetyl amino acid racemase.¹⁰

Each member of the enolase superfamily has a coordinated divalent metal ion (usually Mg^{++}), held in place by three acidic residues. This metal ion assists in forming the enolate anion intermediate of the substrate. Note that the metal ion is absolutely required as the pK_a s of the abstracted proton of the substrate range from 29 to 32, making formation of the enolic intermediate thermodynamically unfavored in the absence of the metal ion.⁶ In general, all reactions in the enolase superfamily use an acid/base chemical mechanism, but the identity and location of the general acid catalysis residues differs depending on the particular enzyme, including dual lysine residues on opposite faces of the substrate (example, muconate lactonizing enzyme), a single lysine residue coupled with an additional coordinated magnesium ion (example, enolase), and a

histidine/aspartic acid dyad also on opposite faces of the substrate (example, mandelate racemase).³ The electrostatic properties of the enzymes differ among subgroup as well and can be used to distinguish members of different subgroups.⁹

The enolase superfamily is a good choice for an α/β -barrel enzyme template because of the chemical diversity of the reactions catalyzed by its members, which allows for several diverse evolution targets. In addition, there is such a high degree of structural similarity among the superfamily that the crystal structures from each of the subgroups are completely superimposable, with their active site residues and metal ligands aligning on top of each other,¹¹ this suggests that changing residues in the substrate binding region should allow for catalysis of the new substrate with the existing active site, allowing for a semi-rational design strategy. Additional data from superimposition shows a large amount of structural conservation in other regions of the protein, with most of the variation in two outer loop regions and in the upper right quadrant of the structures, suggesting that these areas should be targeted for changing the substrate binding.⁷ Finally, previous work by Schmidt et. al. shows that a single amino acid mutation which relaxes the substrate specificity of the enzyme is sufficient to change the reaction catalyzed by two different members of the enolase superfamily to a reaction catalyzed by a third,¹² lending credence to the strategy of changing substrate binding to evolve new function.

6.1.4 Enolase

Enolase catalyzes the penultimate step in the glycolytic pathway of central metabolism, converting 2-phosphoglyceric acid to phosphoenolpyruvate (Figure 6.2).

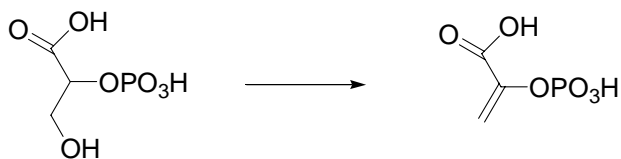


Figure 6.2: The reaction catalyzed by enolase. 2-phosphoglyceric acid (left) is converted to phosphoenolpyruvate (right).

The enzyme is ubiquitous; found in all organisms from single-celled microbes to higher eukaryotes.¹³ The enolases thus far identified from eukaryotes have all been dimeric in tertiary structure, as is the enolase *Eschericia coli*. Recently, however, several prokaryotic enzymes have been identified with an octameric structure, including those from *Bacillus subtilis*, *Streptococcus mutans*, *Zymomonas mobilis*, and *Thermus aquaticus*.¹⁴

The exact mechanism of catalysis for enolase is still under study. The formation of the enolate anion intermediate is thought to begin with abstraction of the α -proton by the ϵ -amino group of a lysine at the end of the 6th beta-strand (K345)¹¹ which in crystal structures is shown to be positioned near the proton of 2-phosphoglyceric acid¹⁵. Enolase also requires the presence of two divalent metal ions per subunit (usually Mg^{++}) for catalysis to occur.^{11, 15} One of these magnesium ions is a permanent part of the structure and is coordinated by the side chains of three aspartic and glutamic acid residues (D246, E295, and D320).¹¹ The second magnesium ion adds after the substrate has been bound and probably aids in decreasing the pKa of the proton to aid in its abstraction.¹⁵

The second step in catalysis is the *anti* β -elimination of a hydroxyl ion, most likely due to an acid/base catalytic mechanism. Here, the most likely participant is a glutamic acid residue (E211) which activates the water molecule for attack. Electron

density of the hydroxyl group of the substrate is visibly directed towards this residue in the crystal structure of yeast enolase. Other important residues, identified by loss of activity in site-directed mutants are E168, K396 and S39.¹⁵ The latter is thought to regulate loop movement upon substrate binding to yield the closed form of the enzyme, promoting catalysis.¹⁶

This work focuses on the enolase enzyme from *E. coli*, *Saccharomyces cerevisiae*, *Lactococcus lactis*, and *Lactobacillus plantarum*. The enolases from both *E. coli* and *S. cerevisiae* have been extensively studied and are well characterized.^{17, 18} In *L. lactis*, the presence of two isoforms of enolase has been noted, one of which is thought to be plasmid-mediated and to have been acquired by lateral gene transfer from *Streptococcus thermophilus*.¹⁹ In addition, enolase activity has been demonstrated in whole cell lysate grown in the presence of glucose and galactose.²⁰ No characterization of the *Lactobacillus plantarum* enolase has been reported, however, a patent reporting its transport to the cellular surface does exist.²¹

The purpose of this work is to clone and overexpress the enolase genes from *E. coli*, *Saccharomyces cerevisiae*, *Lactococcus lactis*, and *Lactobacillus plantarum* in preparation for their use as templates for directed evolution.

6.2 Materials and Methods

6.2.1 Materials

Lactococcus lactis (ATCC #19435), *Lactobacillus plantarum* (ATCC #10012), and *S. cerevisiae* prepared genomic DNA were purchased from the American Type Culture Collection (Manassas, VA). Phosphoenolpyruvate was obtained from Alfa Aesar (Ward

Hill, MA). Bacterial culture medium, antibiotics, IPTG, and Fermentas brand restriction enzymes and T4 DNA ligase were purchased from VWR International (Suwanee, GA). All other chemical reagents were purchased from Sigma Aldrich (St. Louis, MO). The Failsafe PCR enzyme and Premix Buffer A were obtained from Epicentre Biotechnologies (Madison, WI). Primers were synthesized by MWG Biotech (High Point, NC). The *E. coli* BL21(DE3-RIL) expression strain was purchased from Stratagene (La Jolla, CA). The pET 30 Ek/LIC kit was obtained from Novagen (San Diego, CA). The gel extraction kit and Talon Metal Affinity resin were purchased from Qiagen (Valencia, CA). The pET 30 Ek/LIC kit, the Talon Metal Affinity Resin, and the gel extraction kit were used according to the manufacturer's instructions. Polarimetric measurements were taken with a Jasco, Inc. P-1010 polarimeter (Eastman, MD). Spectrophotometric measurements were taken with a DU-800 spectrophotometer from Beckman-Coulter (Fullerton, CA).

6.2.2 Isolation of the Enolase Genes

Genomic DNA from *E.coli*, *L. lactis*, and *L. plantarum* was prepared using the method developed by Murphy.²² Blunt-end PCR primers were designed for the initial cloning out of genomic DNA. Two genes were annotated as enolases in the *L. plantarum* genome; both were cloned. Despite the existence of two enolase genes in *L. lactis*,¹⁹ only the sequence corresponding to the ancestral gene was cloned (enoA). The primer sequences are as follows:

E. coli: Forward ATG TCC AAA ATC GTA AAA ATC ATC G

Reverse TTA TGC CTG GCC TTT GAT CTC

S. cerevisiae: Forward ATG GCT GTC TCT AAA GTT TAC GCT AG

Reverse TTA TAA TTT GTC ACC GTG GTG G

L. lactis: Forward ATG TCA ATT ATT ACT GAT ATT TAT GCT CG

Reverse TTA TTT TTT AAG GTT GTA GAA TGC TTT AAG

L. plantarum: Forward ATG TCT ATT ATT ACA GAT ATT TAT GCT CGC

Reverse TTA CTT GCT AGT AAT GGT GTT CCG T

Forward ATG GAA AAA CAA GTT ATT GAA ACG G

Reverse TTA GTC AAA CAT CAC GTT ATT TGG G

For the *E. coli*, *L. lactis*, and both *L. plantarum* genes, a hot-start PCR was performed using a 5 minute denaturation at 98°C, followed by a pause to add polymerase, then thirty cycles with a 30 second denaturation at 98°C, 30 second annealing step at 55°C, and a 2 minute extension step at 72°C. Finally a 10 minute polishing step at 72°C was performed. For the *S. cerevisiae* gene, the PCR cycling was similar, except that the annealing step was done at 53°C.

Following PCR, the fragments corresponding to the enolase genes were gel purified to remove excess primers, dNTPs, and PCR enzyme. Fifty nanograms of purified DNA were used in a secondary PCR reaction to add the extensions necessary for ligation-independent cloning. The primers were exactly as above, except with the addition of the following regions complementary to the pET30 LIC cloning region:

5' GAC GAC GAC AAG

3' GAG GAG AAG CCC GGT

The PCR products with extensions were generated using the same cycling parameters as above, gel purified, and cloned into the pET30 vector following the procedures outlined in the product manual. The vectors were transformed into the

BL21(DE3-RIL) expression strain. Restriction digest was used to determine the presence of a correctly sized insert, and selected colonies were sequenced at the FAME sequencing center.

6.2.3 Overexpression and Purification of the Enolase Enzymes

Single colonies were inoculated into at 10 mL overnight seed culture and grown at 37°C. The next day, 100 µl of this culture was diluted into 500 mL of fresh medium with 30 µg/mL kanamycin. The cells were grown until the optical density at 600 nm reached 0.5, and then induced with IPTG to a final concentration of 0.2 mM. The culture was incubated for an additional 4 hours at 37 °C to allow for protein production and then harvested by centrifugation and frozen overnight at -20°C.

The following day, cells were lysed via sonication, and the enolase protein was purified using the Talon Metal Affinity resin. The resin was used primarily according to manufacturer's instructions, except that 50 mM HEPES was used in place of 50 mM phosphate due to the potential metal chelating effects of phosphate and the knowledge that enolase requires the presence of magnesium ions. The protein lysate, unbound fraction, and eluate were analyzed via SDS-PAGE gel.

6.2.4 Synthesis of N-acetyl-(R)-phenylglycine

N-acetyl-(R)-phenylglycine (Figure 6.3) was synthesized by acetylating the amino group of (R)-phenylglycine. 1.51 grams of (R)-phenylglycine was added to 2 mL of 50:50 (w/v) sodium hydroxide: water. To this, 1 mL of acetic anhydride was added and the pH was adjusted to 6.5 to prevent chemical racemization of the product. The solution was stirred until it became clear, approximately 3 hours. Subsequently, 10 N HCl was added to adjust the pH to 1.5 and promote crystallization of the product. The solution was

stirred an additional 1.5 hours to allow crystals to form. The crystals were filtered, washed with water, and vacuum dried. The optical rotation of a 0.5 g/ 100 mL solution in distilled water was measured and compared to the literature value.

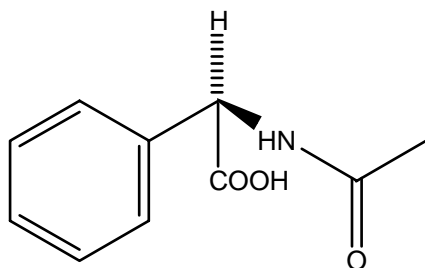


Figure 6.3: N-acetyl-(R)-phenylglycine

6.2.5 Activity Assays

The eluted protein was assayed for enolase activity using a solution of 2 mM phosphoenolpyruvate in a buffer consisting of 50 mM imidazole, 10 mM magnesium sulfate, and 400 mM KCl, pH 8.0. Various amounts of enzyme were added and the decrease in absorbance at 240 nm was monitored for the conversion of phosphoenolpyruvate.

The activity of the enolases as a mandelate racemase was tested using a 13.1 mM solution of (S)-mandelic acid in 50 mM Tris-HCl pH 7.5. Enzyme was added and the change in optical rotation at 589 nm was monitored. The N-acetyl amino acid racemase activity was tested using a 50 mM solution of N-acetyl-(R)-phenylglycine in 50 mM Tris-HCl pH 7.5. The enzyme was added and the change in optical rotation at 589 nm was monitored.

6.3 Results

6.3.1 Isolation of Enolase Genes

The initial amplification of the enolase genes from genomic DNA is shown in Figure 6.4. In all five cases, there is a single product band, of approximately correct size. The cloning into pET30 was successful and the sequencing results are listed in Table 6.1. In each case, there was at least one mutation; however, none of these were in residues identified in the literature as critical for catalysis.

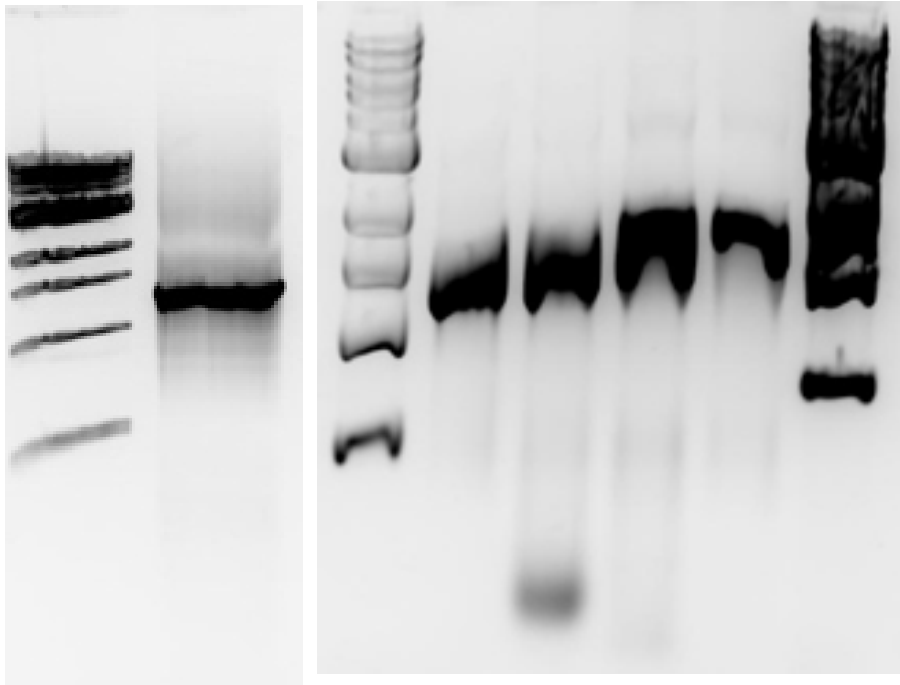


Figure 6.4: Initial amplification of the enolase genes
LEFT: Lane 1- Ladder, Lane 2- *S. cerevisiae* enolase
RIGHT Lane 1- Ladder, Lane 2- *E. coli* enolase, Lane 3- *L. lactis* enolase, Lane 4- *L. plantarum* 1 enolase, Lane 5- *L. plantarum* 2 enolase

Table 6.1: Summary of mutations in the various enolase enzymes:

<i>S. cerevisiae</i>	S188F
<i>E. coli</i>	D255G
<i>L. lactis</i>	I74T
<i>L. plantarum</i> 1	A306T
<i>L. plantarum</i> 2	S200T, M221I

6.3.2 Enolase Overexpression and Purification

The SDS-PAGE results for the overexpression of the *E. coli* and *S. cerevisiae* enolases are shown in Figure 6.5. Both of these expressed well and were purified to near homogeneity using the Metal Affinity resin.

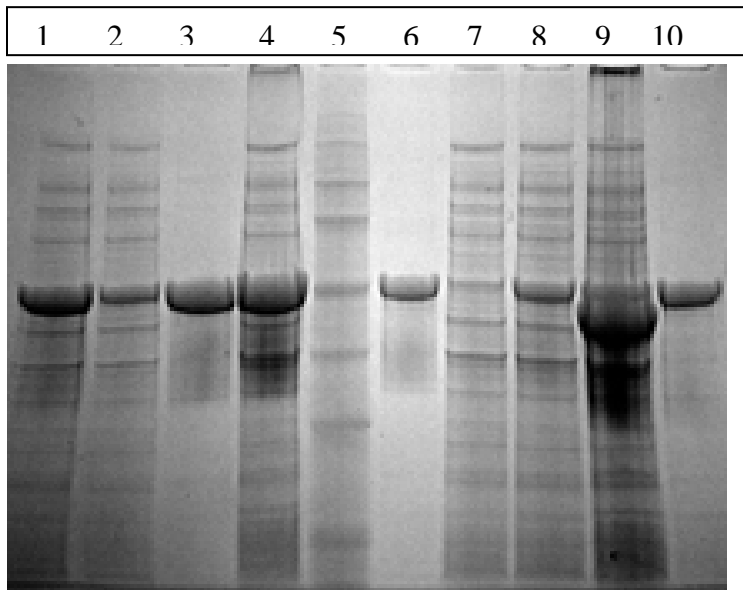


Figure 6.5: Expression of *S. cerevisiae* and *E. coli* enolases
Lane 1- *E. coli* lysate, Lane 2- *E. coli* unbound fraction, Lane 3- purified *E. coli* enolase, Lane 4- *E. coli* insoluble fraction, Lane 5- Ladder, Lane 6- purified *S. cerevisiae* enolase, Lane 7- purified *S. cerevisiae* unbound fraction, Lane 8- *S. cerevisiae* lysate, Lane 9- *S. cerevisiae* insoluble fraction, Lane 10- purified *S. cerevisiae* enolase

The SDS-PAGE results for the overexpression of *enoA* from *L. lactis* are shown in Figure 6.6. Here, the expression level in the soluble fraction is very low and the amount of purified protein obtained was also much lower than with *E. coli* or *S. cerevisiae*. The purity also seems to be much lower as indicated by the presence of additional bands with similar level of intensity to the enolase band.

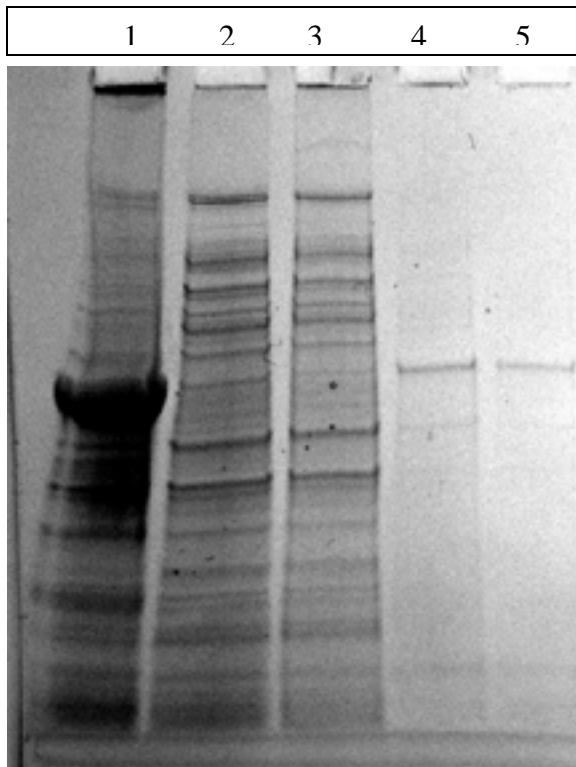


Figure 6.6: Expression of *L. lactis* enolase
Lane 1- Insoluble Fraction, Lane 2- Lysate, Lane 3- Unbound, Lanes 4 and 5 - purified *L. lactis* enolase

Finally, the SDS-PAGE results for the overexpression of both enolases from *L. plantarum* are shown in Figure 6.7. In neither case is a visible amount of purified protein obtained, despite measurable amounts of protein detected in the Bradford assay (165

$\mu\text{g/mL}$ for LP1 and $200 \mu\text{g/mL}$ for LP2). This indicates non-specific binding of various proteins to the resin.

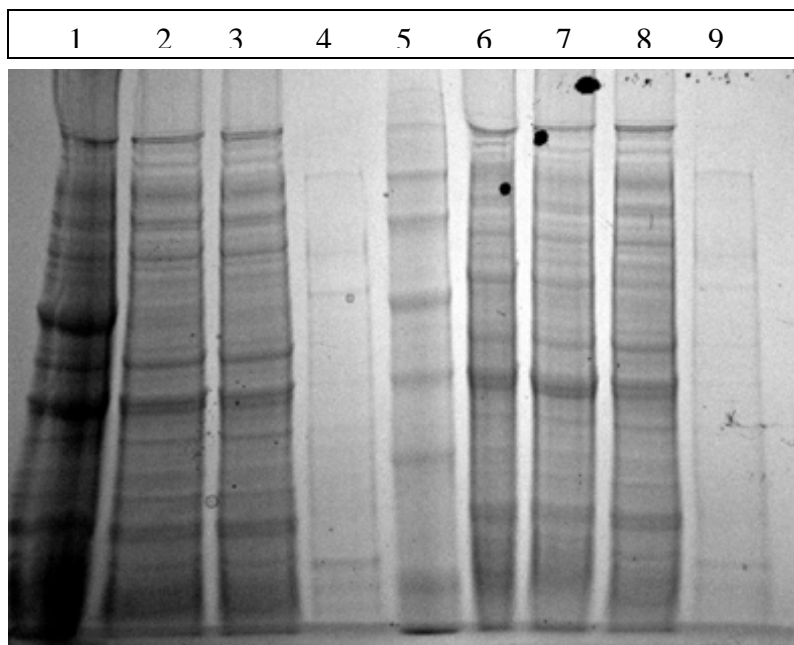


Figure 6.7: Expression of *Lactobacillus plantarum* enolases
Lane 1- *L. plantarum* 1 insoluble fraction, Lane 2- *L. plantarum* 1 lysate, Lane 3- *L. plantarum* 1 unbound fraction, Lane 4- purified *L. plantarum* 1, Lane 5- Ladder, Lane 6- *L. plantarum* 2 insoluble fraction, Lane 7- *L. plantarum* 2 lysate, Lane 8- *L. plantarum* 2 unbound fraction, Lane 9- purified *L. plantarum* 2.

6.3.3 N-acetyl-D-phenylglycine Synthesis

H^1 and C^{13} NMR synthesized N-acetyl-(R)-phenylglycine were consistent with successful synthesis of the desired product. The optical rotation of a 0.5 g/100 mL solution in room temperature distilled water at 589 nm was -0.6429 , which corresponds to a specific molar rotation of -128.64° . The published value for the molar rotation under the same conditions is -197° ,²³ indicating that the sample is about 34% racemized.

6.3.4 Activity Assays

The equilibrium of the enolase reaction lies toward the conversion of 2-phosphoglyceric acid to phosphoenolpyruvate. Therefore, reaction of phosphoenolpyruvate to 2-phosphoglyceric acid does not go to completion. This complicates the calculation of specific activity and characterization of the enzyme under different conditions because a minimal decrease in absorbance at 240 nm is seen before equilibrium is achieved. Therefore, measurements of enolase activity on the proteins (Table 6.2) are only classified as 'active' (i.e. decrease in absorbance) or 'inactive' (i.e. no decrease in absorbance). The enolases from *E. coli*, *S. cerevisiae*, and *L. lactis* were all active, whereas the two enzymes from *L. plantarum* did not show any activity. None of the enolases in this study showed the ability to racemize mandelic acid or N-acetyl-D-phenylglycine under the conditions of measurement.

Table 6.2: Activity tests of purified enolases

	enolase activity	mandelate racemase activity	N-acetyl amino acid racemase activity
<i>E. coli</i>	active	not active	not active
<i>S. cerevisiae</i>	active	not active	not active
<i>L. lactis</i>	active	not active	not active
<i>L. plantarum</i> 1	not active	not active	not active
<i>L. plantarum</i> 2	not active	not active	not active

6.4 Discussion

The enolases from *E. coli*, *S. cerevisiae*, and *L. lactis* were successfully cloned and overexpressed in *E. coli* BL21(DE3-RIL). All three were active as enolases, but not active towards mandelic acid or N-acetyl-D-phenylglycine. Neither of the putative enolase genes from *L. plantarum* overexpressed adequately in the soluble, nor did the

purified fraction show activity towards phosphoenolpyruvate, mandelic acid, or N-acetyl-D-phenylglycine. The only report on the *L. plantarum* enzymes in the literature is a patent application which reports evidence that they may be partially transported to the cellular membrane and exposed extracellularly.²¹ In order to be exposed to the extracellular environment, the enolase would have to be at least partially inserted into the cellular membrane. To accomplish this, special hydrophobic domains would be necessary and these may cause aggregation when the protein is being overexpressed, causing a lack of soluble expression. Use of the freeware program SignalP to evaluate the likelihood of a signal sequence that could be used for such translocation and insertion, however, detected no evidence of a signal sequence.²⁴

The substrate N-acetyl-D-phenylglycine was successfully synthesized. Although the product is predicted to be 34 % racemized based on comparison to the literature value for the molar optical rotation, the rotation is sufficient for the compound to be used in assays for N-acetyl amino acid racemase activity without further purification, since it has sufficient optical rotation.

In summary, three potential α/β -barrel scaffolds, the enolases from *E. coli*, *S. cerevisiae*, and *L. lactis* were successfully cloned, overexpressed, and their enolase activity verified by assay with phosphoenolpyruvate. These scaffolds can now be used for directed evolution of various enolase superfamily activities.

REFERENCES

1. Wierenga, R., The TIM-barrel fold: A versatile framework for efficient enzymes. *FEBS Letters* **2001**, 492, 193-198.
2. Vega, M. C.; Lorentzen, E.; Linden, A.; Wilmanns, M., Evolutionary markers in the (B/A)8-barrel fold. *Current Opinion in Chemical Biology* **2003**, 7, 694-701.
3. Gerlt, J. A.; Raushel, F. M., Evolution of function in (B/A)8-barrel enzymes. *Current Opinion in Chemical Biology* **2003**, 7, 1-13.
4. Nagano, N.; Orengo, C. A.; Thornton, J. M., One fold with many functions: The evolutionary relationships between TIM barrel families based on their sequences structures and functions. *Journal of Molecular Biology* **2002**, 321, 741-765.
5. Reardon, D.; Farber, G. K., The structure and evolution of a/b barrel proteins. *FASEB Journal* **1995**, 9, 497-503.
6. Gerlt, J. A.; Babbitt, P. C., Divergent evolution of enzymatic function: Mechanistically diverse superfamilies and functionally distinct suprafamilies. *Annual Reviews of Biochemistry* **2001**, 70, 209-246.
7. Babbitt, P. C.; Gerlt, J. A., New functions from old scaffolds: How nature reengineers enzymes for new functions. *Advances in Protein Chemistry* **2001**, 55, 1-28.
8. Meng, E. C.; Polacco, B. J.; Babbitt, P. C., Superfamily active site templates. *Proteins: Structure, Function, and Bioinformatics* **2004**, 55, 962-976.
9. Livesay, D. R.; Jambeck, P.; Rojnuckarin, A.; Subramaniam, S., Conservation of electrostatic properties within enzyme families and superfamilies. *Biochemistry* **2003**, 42, 3464-3473.
10. Babbitt, P. C.; Hasson, M. S.; Wedekind, J. E.; Palmer, D. R.; Barrett, W. C.; Reed, G. H.; Rayment, I.; Ringe, D.; Kenyon, G. L.; Gerlt, J. A., The enolase superfamily: A general strategor for enzyme-catalyzed abstraction of the α -protons of carboxylic acids. *Biochemistry* **1996**, 35, 16489-16501.
11. Hasson, M. S.; Schlicting, I.; Moulai, J.; Taylor, K.; Barrett, W. C.; Kenyon, G. L.; Babbitt, P. C.; Gerlt, J. A.; Petsko, G. A.; Ringe, D., Evolution of an enzyme active site: The structure of a new crystal form of muconate lactonizing enzyme compared with mandelate racemase and enolase. *Proceedings of the National Academy of Sciences* **1998**, 95, 10396-10401.

12. Schmidt, D. M.; Mundorff, E. C.; Dojka, M.; Bermudez, E.; Ness, J. E.; Govindarajan, S.; Babbitt, P. C.; Minshull, J.; Gerlt, J. A., Evolutionary potential of (b/a)8-barrels: Functional promiscuity produced by single substitutions in the enolase superfamily. *Biochemistry* **2003**, 42, (28), 8387-8393.
13. Tracy, M. R.; Blair, H. S., Evolutionary history of the enolase gene family. *Gene* **2000**, 259, 129-138.
14. Brown, C. K.; Kuhlman, P. L.; Mattingly, S.; Slaters, K.; Calie, P. J.; Farrar, W. W., A model of the quaternary structure of enolases, based on structural and evolutionary analysis of the octameric enolase from *Bacillus subtilis*. *Journal of Protein Chemistry* **1998**, 17, (8), 855-866.
15. Reed, G. H.; Poyner, R. R.; Larsen, T. M.; Wedekind, J. E.; Rayment, I., Structural and mechanistic studies of enolase. *Current Opinion in Structural Biology* **1998**, 6, 736-743.
16. Gunasekaran, K.; Ma, B.; Nussinov, R., Triggering loops and enzyme function: Identification of loops that trigger and modulate movements. *Journal of Molecular Biology* **2003**, 332, 143-159.
17. Westhead, E.; McLain, G., A purification of Brewers' and Bakers' yeast enolase yielding a single active component. *The Journal of Biological Chemistry* **1964**, 239, (8), 2464-2468.
18. Spring, T. G.; Wold, F., The purification and characterization of *Escherichia coli* enolase. *The Journal of Biological Chemistry* **1971**, 248, (22), 6797-6802.
19. Jamet, E.; Ehrlich, S. D.; Duperray, F.; Renault, P., Study of the duplicated glycolytic genes in *Lactococcus lactis* IL1403. *Lait* **2001**, 81, 115-129.
20. Even, S.; Lindley, N. D.; Cocaign-Bousquet, M., Molecular physiology of sugar catabolism in *Lactococcus lactis* IL 1403. *Journal of Bacteriology* **2001**, 183, (13), 3817-3824.
21. Israelsen, H.; Madsen, S. M.; Glenting, J.; Vrang, A.; Noerrelykke, M. R.; Hansen, A. M.; Ahrne, S. E. I.; Molin, G.; Ravn, P.; Beck, H. C. Cell surface polypeptides from *Lactobacillus* or *Bifidobacterium* and their use as immunomodulating probiotic compounds. 2004.
22. Murphy, T. M. Directed evolution of a novel biocatalyst: N-acetyl amino acid racemase. Master's, Georgia Institute of Technology, Atlanta, 2002.
23. Shiraiwa, T.; Sakata, S.; Kurokawa, H., Asymmetric transformation of N-acetyl-(RS)-2-phenylglycine. *Chemistry Express* **1988**, 3, (7), 415-417.

24. Bendtsen, J. D.; Nielsen, H.; Heijne, G. v.; Brunak, S., Improved prediction of signal peptides: SignalP 3.0. *Journal of Molecular Biology* **2004**, 340, 783-795.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

The widespread adoption of biocatalytic processes is currently hampered by long development times for biocatalysts with novel and sufficient activity and adequate stability under operating conditions. Protein engineering, while extremely useful for modifying the properties of protein catalysts in select cases, still cannot be performed rapidly enough for many applications. The work presented herein seeks to increase the efficiency of protein engineering experiments and decrease the time necessary to develop novel biocatalysts.

Protein engineering involves two steps: the generation of diversity and the screening or selection of variants with the desired properties. Chapter VI outlines preliminary work on the diversity generation step, attempting to create a smaller, more manageable number of mutants, but with a larger likelihood of improved activity for any given mutant. Chapters III through V are concerned with decreasing the time investment in the screening process.

This work represents a foundation for a number of subsequent experiments. In Chapters III and IV, we have shown that the use of pooling increases the sample size that can be screened, and as a consequence, increases the number of improved mutants that will be identified. The Monte-Carlo simulation model of pooling can be used to test several pooling strategies *in silico* to determine which experimental protocol is best for a given system. Given that screening experiments represent a significant time investment,

while computational results can be generated for a number of scenarios rather quickly, the use of the model is expected to save a significant amount of time and effort in the laboratory. Pooling is most suited to experimental systems where the average level of background is low, such as in the evolution of a completely new activity, or with a diversity generation protocol that create a high percentage of inactive mutants.

One particular type of experiment for which pooling might be useful is the evolution of mutants with very highly increased thermostability. Much of the literature in this area suggests that gains in thermostability come largely from the additive contributions of several interactions that have small impact on their own. One can envision a strategy whereby mutants containing single, beneficial mutations identified by error-prone PCR or similar methods are recombined and screened using pooling at a very high temperature cut-off. The background level of activity in this situation would be negligible, and only the very rarely occurring recombinants that received most of the beneficial mutations would survive. This strategy could rapidly identify very highly thermostable mutants.

Additionally, the concept of pooling could be expanded to include the creation of pools at the transformation step using electroporation. Theoretically, each cell in the transformation mixture can take up more than one plasmid, and although these plasmids might segregate into different daughter cells, each colony grown on an agar plate would then represent a heterogeneous population containing variants with different activity levels, essentially an artificially constructed pool. To utilize our Monte-Carlo model we would need to estimate the pooling level of each colony (e.g. how many different variants are represented in each colony). There are several models of DNA uptake by

electroporation, both empirical¹ and computational² which could be integrated into the simulation to determine how many plasmids will enter into each cell on average. To determine the number of different variants that would be included, the activity distribution would be applied to determine the probability of a given number of unique plasmids entering the cell. The external DNA concentration, the pulse time of the electric field, and the voltage applied could be used to tune the number of variants per pool. In this scheme, each colony on an agar plate would actually represent a heterogeneous population, or a pool that could be assayed individually. These colonies could be directly picked to a multi-well plate, creating one fewer step in the experimental protocol. Alternatively, such a system would allow the extension of the pooling model to filter lift assays of the colonies directly after transformation.

Chapter V discusses the creation of a series of cloning vectors to express the variant proteins. The current vectors were constructed in additive fashion, so the intermediates are available for use as well as the final vector which contains a tetracycline-based promoter/operator, triclosan resistance, a ligation-independent cloning site, and a monomeric red fluorescent protein reporter (mRFP). We attempted to construct an intein-mediated fusion of the mRFP expression reporter to our gene of interest so that it would be present in stoichiometric amounts, however this did not fold correctly. A recent paper by Tsien suggests that the problem may be due to a low tolerance to fusion proteins by mRFP, which could potentially be improved by changing the termini of the protein to be more similar to GFP.³ This should be the first future experiment and could be done combinatorially with fast and efficient selection for fusion-tolerant mRFP genes by fluorescence activated cell sorting.⁴

Chapter VI outlines the cloning, overexpression, and basic activity assay of the α/β -barrel enzyme enolase for use as a scaffold for the evolution of new function. α/β -barrel enzymes represent an efficient scaffold since a large portion of the residues are responsible for maintaining the secondary structure elements and do not participate in binding of the substrate or catalysis. In addition, the loop residues responsible for structure are separate from the residues, which are responsible for stability. Thus, with only mutations targeted to one portion of the enzyme, new functions can be evolved.

α/β -barrel enzymes participate in reactions in nearly all of the enzyme classes, so any chemistry is compatible with the fold. To test the feasibility of the α/β -barrel as a scaffold, the first evolution targets should be chosen from the same superfamily as enolase. Closely related functions include the *ortho*-succinyl benzoate synthase, N-acetyl amino acid racemase, muconate lactonizing enzyme, glucarate dehydratase, and mandelate racemase.⁵ The *ortho*-succinyl benzoate synthase activity is thermodynamically much easier than the other reactions since the pKa of the hydrogen that is abstracted is only approximately 8, while the other reactions involve abstraction of a hydrogen atom that has a much higher pKa (15 and higher). However, this substrate must be synthesized enzymatically, which could prove very difficult. In addition, the pKa of the alpha hydrogen of 2-phosphoglyceric acid (the substrate of enolase) is greater than 30,⁶ and the enzyme is still able to abstract it, so this bears less consideration than the ease of assay and/or a selection method for identification of active mutants. Thus, mandelate racemase should be the first target as its activity can be selected for (using the ability of *Pseudomonas aeruginosa* to grow on S-mandelate as a sole carbon source as in the identification of the gene for the enzyme originally,⁷ and the substrate has a high

optical rotation, making it ideal for polarimetric assays. Another potential set of targets should include other reactions that are thermodynamically easy to accomplish, such as hydrolyses. Since it is known which residues are responsible for binding the current substrate in enolase, and the structures of the *E. coli* and *S. cerevisiae* structures have been solved, structure guided determination areas to select for mutation should be explored for further refinement of residues to target. Another potential tool in choosing which areas of the protein to mutate would be to dock the targeted substrate into the crystal structure. Docking may also help determine which residues to target to change the active site chemistry as well.

The work presented here is a foundation for many future protein engineering projects. These tools should help to shorten the process of improving existing functions, and evolving new ones, increasing the applicability of biocatalysts to reactions which are currently inaccessible.

REFERENCES

1. Canatella, P.; Prausnitz, M., Prediction and optimization of gene transfection and drug delivery by electroporation. *Gene Therapy* **2001**, 8, 1464-1469.
2. Smith, K. C.; Neu, J. C.; Krassowska, W., Model of creation and evolution of stable electropores for DNA delivery. *Biophysical Journal* **2004**, 86, 2813-2826.
3. Shaner, N. C.; Campbell, R. E.; Steinbach, P. A.; Giepmans, B. N. G.; Palmer, A. E.; Tsien, R. Y., Improved monomeric red, orange and yellow fluorescent proteins derived from *Discosoma* sp. red fluorescent protein. *Nature Biotechnology* **2004**, 22, (12), 1567-1572.
4. Daugherty, P. S.; Iverson, B. L.; Georgiou, G., Flow cytometric screening of cell-based libraries. *Journal of Immunological Methods* **2000**, 243, 211-227.
5. Babbitt, P. C.; Hasson, M. S.; Wedekind, J. E.; Palmer, D. R.; Barrett, W. C.; Reed, G. H.; Rayment, I.; Ringe, D.; Kenyon, G. L.; Gerlt, J. A., The enolase superfamily: A general strategor for enzyme-catalyzed abstraction of the α -protons of carboxylic acids. *Biochemistry* **1996**, 35, 16489-16501.
6. Reed, G. H.; Poyner, R. R.; Larsen, T. M.; Wedekind, J. E.; Rayment, I., Structural and mechanistic studies of enolase. *Current Opinion in Structural Biology* **1998**, 6, 736-743.
7. Ransom, S.; Gerlt, J.; Powers, V.; Kenyon, G., Cloning, DNA sequence analysis, and expression in *Escherichia coli* of the gene for mandelate racemase from *Pseudomonas putida*. *Biochemistry* **1988**, 27, (2), 540-545.

APPENDIX A: POOLING CODE

The executable command to run a pooling experiment is:

```
javac -classpath . *.java
java -classpath . -Xms128m -Xmx512m Experiment insert_data_file_name_here.txt
```

ActivityDistribution:

```
/**
 * Interface ActivityDistribution
 *
 * This interface forces any implementing class to
 * provide a function called getActivity() that simulates
 * the sampling of a random activity from a population.
 *
 * @author <a href="mailto:cspencer@cc.gatech.edu">Cody Spencer</a>
 */
public interface ActivityDistribution {

    /**
     * Simulates the sampling of a random activity
     * from a population.
     *
     * @return A random activity value
     */
    public double getActivity();

} //Interface ActivityDistribution
```

ActivityFunction:

```
import cern.jet.random.Uniform;
/**
 * Class ActivityFunction
 *
 * Represents an activity distribution function with
 * seven defining parameters.
 *
 * The activity value is generated by the following function:
 *  $f(x) = y + a * e^{(-b * x)} + c * e^{(-d * x)} + g * e^{(-h * x)}$  where
```

```

* y, a, b, c, d, g, and h are the set function parameters,
* e is the exponential number (~2.71), and
* x is a uniformly distributed random variable.
*
* @author <a href="mailto:cspencer@cc.gatech.edu">Cody Spencer</a>
*/
public class ActivityFunction implements ActivityDistribution{

    /* Function parameters */
    private double y;
    private double a;
    private double b;
    private double c;
    private double d;
    private double g;
    private double h;

    /**
     * Class constructor for ActivityFunction.
     *
     * @param params The initial activity function parameters.
     */
    public ActivityFunction(double[] params){
        setParams(params);
    }//ActivityFunction(double[])

    /**
     * Modifier for the parameters to the
     * activity distribution function.
     *
     * @param params The array of parameters that
     *             define this activity function.
     */
    public void setParams(double[] params){
        y = params[0];
        a = params[1];
        b = -1.0 * params[2];
        c = params[3];
        d = -1.0 * params[4];
        g = params[5];
        h = -1.0 * params[6];
    }//setParams(double[])

    /**
     * Simulates the sampling of an activity from this activity

```

```

    * distribution function with set parameters.
    *
    * @return A non-negative value representing a
    *         sampled activity level.
    */
    public double getActivity(){
        double x = Uniform.staticNextDouble();
        double toReturn = y+ a*Math.exp(b*x)+ c*Math.exp(d*x)+ g*Math.exp(h*x);

        if(toReturn < 0.0)
            return 0.0;

        return toReturn;
    }//getActivity()

} //class ActivityFunction

```

ActivityHistogram:

```

import cern.jet.random.Uniform;

/**
 * Class ActivityHistogram
 *
 * An ActivityHistogram represents the distribution of possible values
 * that the activity level could be in a population.
 *
 * @author <a href="mailto:cspencer@cc.gatech.edu">Cody Spencer</a>
 */
public class ActivityHistogram implements ActivityDistribution {

    /**
     * An array representing this values in this histogram.
     */
    private double[] histo;

    /**
     * Class constructor for ActivityHistogram.
     *
     * @param h An array representing this values in this histogram.
     */
    public ActivityHistogram(double[] h){
        histo = h;
    } //ActivityHistogram(double[])
}

```

```

/**
 * Simulates the sampling of a random activity from this
 * activity distribution histogram.
 *
 * @return A non-negative value representing a
 *         sampled activity level.
 */
public double getActivity(){
    int index = (int)(Uniform.staticNextDouble() * histo.length);
    return histo[index];
} //getActivity()

} //Class ActivityHistogram

```

BinDistribution:

```

import cern.jet.random.Uniform;

public class BinDistribution implements ActivityDistribution{

    private double[] percents;
    private double[] activities;

    public BinDistribution(double[] percents, double[] activities){
        this.percents = percents;
        this.activities = activities;
        normalize();
    } //BinDistribution(double[],double[])

    private void normalize(){
        double sum = 0.0;
        for(int i=0; i < percents.length; i++){
            sum += percents[i];
        }
        for(int j=0; j < percents.length; j++){
            percents[j] = percents[j] / sum;
        }
    } //normalize()

    public double getActivity(){
        int bin = 0;
        double temp = percents[0];
        double rand = Uniform.staticNextDouble();
        int max_size = percents.length - 1;

```

```

        while((rand > temp) && (bin < max_size)){
            bin++;
            temp += percents[bin];
        }//while

        return activities[bin];

    }//getActivity()

    public String toString(){
        String strToReturn = "BinDistribution ";
        strToReturn += "[ACTIVITY,%]\n";
        for(int i=0; i < percents.length; i++){
            strToReturn += "[" + activities[i] + "," + percents[i] + "]\n";
        }
        return strToReturn;
    }//toString()

} //class BinDistribution

```

Cell:

```

import java.io.*;

/**
 * Cell.java
 *
 * For use in the Directed Evolution Project.
 *
 * This class simulates sequences of bases that are composed into the DNA
 * of Cells and defines operations on such sequences, like mutatgenesis.
 *
 * Created: Mon Sep 02 03:16:50 2002
 *
 * @author <a href="mailto:rjojr@cc.gatech.edu">Jim Ogilvie</a>
 * @version 1.0
 */
public class Cell {

    /**=====
    ==*/
    /*      INSTANCE VARIABLES      */
    /**=====
    ==*/

```

```

/**
 * Instance variable denoting the base sequence of this Cell's DNA
 */
private char[] bases;

/**
 * A value indicating the activity of this Cell
 */
private double activity;

/**
 * A value indicating the stability of this Cell
 */
private double stability;

/**
 * Is this cell a supermutant or dud?
 */
private boolean supermutant;

/*=====
==*/
/*          S T A T I C   V A R I A B L E S          */

/*=====
==*/

/**
 * Constant <code>BASE_A_INDEX</code> represents the
 * int used to index the base A in arrays.
 */
public static final int BASE_A_INDEX = 0;

/**
 * Constant <code>BASE_C_INDEX</code> represents the
 * int used to index the base C in arrays.
 */
public static final int BASE_C_INDEX = 1;

/**
 * Constant <code>BASE_G_INDEX</code> represents the
 * int used to index the base G in arrays.
 */

```

```

public static final int BASE_G_INDEX = 2;

/**
 * Constant <code>BASE_T_INDEX</code> represents the
 * int used to index the base T in arrays.
 */
public static final int BASE_T_INDEX = 3;

/**
 * Constant <code>BASE_A_CHAR</code> is the
 * char representation of the base A.
 */
public static final char BASE_A_CHAR = 'a';

/**
 * Constant <code>BASE_C_CHAR</code> is the
 * char representation of the base C.
 */
public static final char BASE_C_CHAR = 'c';

/**
 * Constant <code>BASE_G_CHAR</code> is the
 * char representation of the base G.
 */
public static final char BASE_G_CHAR = 'g';

/**
 * Constant <code>BASE_T_CHAR</code> is the
 * char representation of the base T.
 */
public static final char BASE_T_CHAR = 't';

/**
 * <code>BASE_CHARS</code> is a constant array containing BASE_X_CHAR
 * at index BASE_X_INDEX for all four bases, A,C,G,and T.
 */
public static final char[] BASE_CHARS = { BASE_A_CHAR, BASE_C_CHAR,
                                         BASE_G_CHAR, BASE_T_CHAR };

/**
 * Standard, default constant distribution
 * used to generate random gene sequences.
 */
public static final double[] DEFAULT_FREQ = {0.25, 0.25, 0.25, 0.25};

```

```

/**
 * Another constant used in generating random gene sequences -
 * this value specifies the length of the random sequence.
 */
public static final int DEFAULT_GENE_LENGTH = 60;

/**
 * Default file to look in for mutation rates among bases
 */
public static final String DEFAULT_MUTAGENESIS_FILE = "mutagenesis.txt";

/*=====
==*/
/*          C O N S T R U C T O R S          */
/*=====
==*/

/**
 * Creates a new <code>Cell</code> instance given the bases.
 *
 * @param bases a <code>char[]</code> value specifying the base
 *           sequence for this Cell - this array should only
 *           contain the ASCII characters 'a', 'c', 'g', and 't'.
 */
public Cell(char[] bases){
    this.bases = bases;
}

/**
 * Creates a new <code>Cell</code> instance from the default values.
 */
public Cell(){
    this(DEFAULT_GENE_LENGTH);
}

/**
 * Creates a new <code>Cell</code> instance of given length,
 * using the default distribution on bases.
 *
 * @param length a positive length(in number of bases) this Cell should be.
 */

```

```

public Cell(int length){
    this(length, DEFAULT_FREQ);
}

/**
 * Creates a new <code>Cell</code> instance of given activity.
 *
 * @param activity This Cell's activity level.
 */
public Cell(double activity){
    this.activity = activity;
}

public Cell(double activity, boolean sm){
    this.activity = activity;
    supermutant = sm;
}

/**
 * Creates a new <code>Cell</code> instance.
 *
 * @param length an <code>int</code> value
 * @param frequency a <code>double[]</code> value
 */
public Cell(int length, double[] frequency){
    // making sure length is a multiple of 3
    length = (length/3)*3;

    if(length > 0) {
        bases = new char[length];

        double rand = 0.0;
        for(int i=0; i<length; i++) {
            rand = Math.random();

            rand -= frequency[BASE_A_INDEX];
            if(rand < 0) {
                bases[i] = BASE_A_CHAR;
            }
            else {
                rand -= frequency[BASE_C_INDEX];
                if(rand < 0) {
                    bases[i] = BASE_C_CHAR;
                }
            }
            else {
                rand -= frequency[BASE_G_INDEX];

```

```

        if(rand < 0) {
            bases[i] = BASE_G_CHAR;
        }
        else {
            rand -= frequency[BASE_T_INDEX];
            if(rand < 0) {
                bases[i] = BASE_T_CHAR;
            }
            else {
                // error!
            }
        }
    }
} // for(i)
} // if(length > 0)
else {
    // error
}

} // Cell(int, double[])

```

```

/*=====
==*/
/*          S T A T I C   M E T H O D S          */
/*=====
==*/

```

```

/**
 * Describe <code>doMutagenesis</code> method here.
 *
 * @param g a <code>Cell</code> value
 * @return a <code>Cell</code> value
 */
public static Cell doMutagenesis(Cell g){
    return doMutagenesis(g, DEFAULT_MUTAGENESIS_FILE);
} // doMutagenesis(Cell)

```

```

/**
 * Describe <code>doMutagenesis</code> method here.
 *

```

```

* @param g a <code>Cell</code> value
* @param filename a <code>String</code> value
* @return a <code>Cell</code> value
*/
public static Cell doMutagenesis(Cell g, String filename){
    BufferedReader br = null;
    try {
        br = new BufferedReader(new FileReader(filename));
    }
    catch(IOException ioe) {
        System.out.println("Error opening file: " + filename);
    }

    double[][] transitions = new double[4][4];

    try {
        String line = br.readLine();
        int num = 0;
        while(line != null) {
            int index = line.indexOf(':');
            if(index >= 0) {
                transitions[num/4][num%4] =
                    Double.parseDouble(line.substring(index+1));
                num++;
            }
            else {
                throw new Exception("Invalid file format");
            }
            line = br.readLine();
        }
    }
    catch(IOException ioe) {
        System.out.println("Error reading file: " + filename);
    }
    catch(Exception e) {
        System.out.println("Error parsing file: " + filename);
        e.printStackTrace();
    }

    return doMutagenesis(g, transitions);
} //doMutagenesis(Cell,String)

/**
 * Describe <code>doMutagenesis</code> method here.
 */

```

```

* @param g a <code>Cell</code> value
* @param transitions a <code>double[][]</code> value
* @return a <code>Cell</code> value
*/
public static Cell doMutagenesis(Cell g, double[][] transitions){
    char[] bases = new char[g.getBases().length];
    for(int i=0; i<bases.length; i++) {
        bases[i] = g.getBases()[i];
    }

    for(int i=0; i<bases.length; i++) {
        double rand = Math.random();

        int index = getIndexOf(bases[i]);

        boolean done = false;
        for(int j=0; j<transitions[index].length && !done; j++) {
            rand -= transitions[index][j];
            if(rand < 0) {
                bases[i] = BASE_CHARS[j];
                done = true;
            }
        }
    }

    return new Cell(bases);
}

public static final int numSpecialPositions = 8;
public static final double probPosGood = .25;

public static boolean evaluateLandscape(Cell g){
    // determine which positions are crucial to acticity,
    // basically an easy attemp at simulating small worlds
    int[] specPos = new int[numSpecialPositions];
    int index = 0;
    while(index < numSpecialPositions) {
        boolean done = true;
        int newPos = (int) (Math.random()*DEFAULT_GENE_LENGTH/3);
        for(int i=0; i<index; i++)
            if(specPos[i] == newPos)
                done = false;
        if(done)
            specPos[index++] = newPos;
    }
}

```

```

    }

    // setting up a matrix of good/bad values for each position
    // as chosen above, using the specified probability
    boolean[][] goodBad = new boolean[numSpecialPositions][20];
    for(int i=0; i<goodBad.length; i++)
        for(int j=0; j<goodBad[i].length; j++)
            if(Math.random() > probPosGood)
                goodBad[i][j] = true;

    int[] posEffects = new int[numSpecialPositions];
    for(int i=0; i<goodBad.length; i++) {
        char[] codon = new char[3];
        codon[0] = g.getBases()[3*specPos[i]];
        codon[1] = g.getBases()[3*specPos[i]+1];
        codon[2] = g.getBases()[3*specPos[i]+2];
        int aa = AminoAcid.getAminoAcidFromBases(codon);
        posEffects[i] += (goodBad[i][aa] ? 1 : -1);
    }

    int sumEffect = 0;
    for(int i=0; i<posEffects.length; i++) {
        sumEffect += posEffects[i];
    }

    g.setActivity(sumEffect);

    if(sumEffect > 0)
        return true;
    else
        return false;
} //evaluateLandscape(Cell)

/**
 * Describe <code>getIndexOf</code> method here.
 *
 * @param c a <code>char</code> value
 * @return an <code>int</code> value
 */
public static int getIndexOf(char c){
    int index = -1;
    switch(c) {
        case BASE_A_CHAR:
            index = BASE_A_INDEX;
            break;

```

```

        case BASE_C_CHAR:
            index = BASE_C_INDEX;
            break;

        case BASE_G_CHAR:
            index = BASE_G_INDEX;
            break;

        case BASE_T_CHAR:
            index = BASE_T_INDEX;
            break;
    }
    return index;
} //getIndexOf(char)

/**
 * Describe <code>compare</code> method here.
 *
 * @param g a <code>Cell</code> value
 * @param h a <code>Cell</code> value
 */
public static void compare(Cell g, Cell h){
    int[][] changes = new int[4][4];
    int num = Math.min(g.getBases().length, h.getBases().length);
    for(int i=0; i<num; i++){
        changes[getIndexOf(g.getBases()[i])][getIndexOf(h.getBases()[i])]++;
    }

    double[] sums = new double[4];

    for(int i=0; i<changes.length; i++) {
        for(int j=0; j<changes[i].length; j++) {
            sums[i] += changes[i][j];
        }
    }

    for(int i=0; i<changes.length; i++) {
        for(int j=0; j<changes[i].length; j++) {
            System.out.println(BASE_CHARS[i] + " to " +
                BASE_CHARS[j] + ": " + changes[i][j] + "\t " +
                ((double)changes[i][j])/(sums[i]));
        }
    }
} //compare(Cell,Cell)

```

```

/*=====
==*/
/*      INSTANCE METHODS      */

/*=====
==*/

/**
 * Describe <code>getBases</code> method here.
 *
 * @return a <code>char[]</code> value
 */
public char[] getBases(){
    return this.bases;
} //getBases()

/**
 * Modifier for the activity field.
 *
 * @param d This Cell's activity level.
 */
public void setActivity(double d){
    this.activity = d;
} //setActivity(double)

/**
 * Accessor for the activity field.
 *
 * @return This Cell's activity level.
 */
public double getActivity(){
    return this.activity;
} //getActivity()

/**
 * Is this cell a supermutant?
 *
 * @return true or false
 */
public boolean isSupermutant(){
    return this.supermutant;
} //isSupermutant()

```

```

/**
 * Describe <code>toString</code> method here.
 *
 * @return a <code>String</code> value
 */
public String toString(){
    String toRet = "";
    if(bases != null)
        for(int i=0; i<bases.length; i++)
            toRet += bases[i];

    return toRet;
} //toString()

```

```

/**
 * Produces an exact copy of this Cell.
 *
 */
public Cell copy(){
    Cell c = new Cell(this.getBases());
    c.setActivity(this.getActivity());
    return c;
} //copy()

```

```

/*=====
==*/
/*          M A I N   M E T H O D          */
/*=====
==*/

```

```

/**
 * Debug main method
 *
 * @param argv a <code>String[]</code> value
 */
public static void main(String[] argv){
    Cell g = new Cell();
    System.out.println(g);
    evaluateLandscape(g);
    System.out.println(g.getActivity());

    Cell h = Cell.doMutagenesis(g);

```

```

        System.out.println(h);
        evaluateLandscape(h);
        System.out.println(h.getActivity());

        compare(g, h);
    }//main(String[])

} //Cell Class

CellDistribution:

/**
 * Interface CellDistribution
 *
 * A CellDistribution is a way of representing how many cells
 * are picked from a pool of many cells.
 *
 * @author <a href="mailto:cspencer@cc.gatech.edu">Cody Spencer</a>
 */
public interface CellDistribution{

    /**
     * Get the number of cells to use.
     *
     * @return number of cells
     */
    public int sample();

} //interface CellDistribution

```

DBHelper:

```

import java.sql.*;
import java.util.StringTokenizer;

public class DBHelper{

    /*specifies how to connect to the database*/
    private static final String dbURL =
        "jdbc:mysql://newcastle/epool?user=cody";

    /**
     * See if the username and password match in the database.
     * @param usr username
     * @param pass password
     */

```

```

public static boolean validateUser(String usr, String pass){

    //resources
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    String dbpass = null; // or use String dbpass = execSQL(query);
    String sql_query = "select password from users where name like '" +
        usr + "'";

    try {

        //register the driver
        Class.forName("org.gjt.mm.mysql.Driver").newInstance();

        //get a connection to the database
        conn = DriverManager.getConnection(dbURL);
        stmt = conn.createStatement();
        rs = stmt.executeQuery(sql_query);
        dbpass = rs.getString(1);

    }
    catch(SQLException sqle){
        sqle.printStackTrace();
    }
    catch(Exception e){
        e.printStackTrace();
    }
    finally{ //CLOSE ALL RESOURCES
        try{
            if(stmt != null)
                stmt.close();
        }catch(Exception e2){
            e2.printStackTrace();
        }
        finally {
            try{
                if(conn != null)
                    conn.close();
            }catch(Exception e3){
                e3.printStackTrace();
            }
        }
    }
}

```

```

//is correct password?
if(pass.equalsIgnoreCase(dbpass))
    return true;

return false;
} //validateUser(String,String)

/**
 * Executes a single SQL statement and returns relevant information.
 */
public static String execSQL(String sql_query){

    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    String toReturn = null;

    try {

        //register the driver
        Class.forName("org.gjt.mm.mysql.Driver").newInstance();

        //get a connection to the database
        conn = DriverManager.getConnection(dbURL);
        stmt = conn.createStatement();

        if(sql_query == null){
            toReturn = null;
        }
        else{
            //parse query to determine action (query or update)
            StringTokenizer st = new StringTokenizer(sql_query);
            String temp = st.nextToken();
            if(temp.equalsIgnoreCase("select")){
                rs = stmt.executeQuery(sql_query);
                toReturn = rs.getString(1);
            }
            else if(temp.equalsIgnoreCase("insert") ||
                    temp.equalsIgnoreCase("update") ||
                    temp.equalsIgnoreCase("delete")){
                int updatedRows = stmt.executeUpdate(sql_query);
                toReturn = Integer.toString(updatedRows) +
                    " row(s) changed.";
            }
        }
    }
}

```

```

        else
            toReturn = null;
    }//else

    }
    catch(SQLException sqle){
        sqle.printStackTrace();
    }
    catch(Exception e){
        e.printStackTrace();
    }
    finally{ //CLOSE ALL RESOURCES
        try{
            if(stmt != null)
                stmt.close();
        }catch(Exception e2){
            e2.printStackTrace();
        }
        finally {
            try{
                if(conn != null)
                    conn.close();
            }catch(Exception e3){
                e3.printStackTrace();
            }
        }//finally
    }//finally

    return toReturn;

} //end execSQL(String, int)

/**
 * Debugging main method.
 */
public static void main(String argv[]){

    String name = "test";
    String pass = name;

    System.out.println(validateUser(name,pass));

} //end main

```

```
}//class DBHelper
```

DiscreteDistribution:

```
public class DiscreteDistribution implements ActivityDistribution{

    private static double val1 = 0.3125;
    private static double val2 = 0.9375;
    private double val3;
    private double pMutant;

    public DiscreteDistribution(double pMutant){
        if(pMutant > 1)
            this.pMutant = 1;
        else if (pMutant < 0)
            this.pMutant = 0;
        else
            this.pMutant = pMutant;

        val3 = (96.0 - pMutant) / 96.0;
    }

    public double getActivity(){
        double r = Math.random();
        if(r < val1)
            return 0.0;
        if(r < val2)
            return 1.0;
        if(r < val3){
            return 2.0;
        }
        else
            return 15.0;
    }

}

}//class DiscreteDistribution
```

Exp Data:

```
/**
 * Class ExpData
 *
 * A simple record class used to store the data from
 * pooling experiments.
```

```

*
*@see Experiment
*@author <a href="mailto:cspencer@cc.gatech.edu">Cody Spencer</a>
*/
public class ExpData {

    /**
     * Whether or not the target was found.
     */
    private boolean found;

    /**
     * The arithmetic mean value.
     */
    private double mean;

    /**
     * The standard deviation.
     */
    private double stdDev;

    /**
     * The minimum value.
     */
    private double min;

    /**
     * The maximum value.
     */
    private double max;

    private String printout;

    /**
     * Number of supermutants found.
     */
    private int sm_found;

    private int sm_false_pos;
    private int sm_false_neg;

    /**
     * Class constructor specifying all four values.
     *
     * @param mean The arithmetic mean of the experimental data.

```

```

* @param stdDev The standard deviation of the experimental data.
* @param min The minimum value of the experimental data.
* @param max The maximum value of the experimental data.
*/
public ExpData(double mean, double stdDev, double min, double max,
                String printout){
    this.mean = mean;
    this.stdDev = stdDev;
    this.min = min;
    this.max = max;
    this.printout = printout;
} //ExpData(double, double, double, double, String)

/**
 * Class constructor for the case when we only care if the target
 * was found or not.
 *
 * @param found If the target was found or not.
 */
public ExpData(boolean found){
    this.found = found;
} //ExpData(boolean)

/**
 * Class constructor for the case when we only care
 * how many supermutants were found.
 *
 * @param sm_found How many supermutants were found.
 */
public ExpData(int sm_found, String printout){
    this.sm_found = sm_found;
    this.printout = printout;
} //ExpData(int)

/**
 * Class constructor for the case when we only care
 * how many supermutants were found and any false pos/neg
 *
 * @param sm_found How many supermutants were found.
 * @param sm_false_pos Number of false positives
 * @param sm_false_neg Number of false negatives
 */
public ExpData(int sm_found, int sm_false_pos, int sm_false_neg,
                String printout){
    this.sm_found = sm_found;
    this.sm_false_pos = sm_false_pos;

```

```

        this.sm_false_neg = sm_false_neg;
        this.printout = printout;
    }//ExpData(int,int,int,String)

/**
 * Converts all four data values into a textual
 * representation(a String).
 *
 * Overrides the default toString() method found in
 * the Object class (java.lang.Object).
 *
 * @return A textual representation of this object.
 */
public String toString(){
    String toReturn = "Mean = " + Double.toString(mean) +
        "\nStandard Deviation = " + Double.toString(stdDev) +
        "\nMinimum = " + Double.toString(min) +
        "\nMaximum = " + Double.toString(max);
    return toReturn;
}//toString()

/**
 * Accessor for the found variable.
 *
 * @return <code>true</code> if target was found,
 * <code>false</code> if not.
 */
public boolean getFound(){
    return found;
}//getFound()

/**
 * Accessor for the sm_false_pos value.
 *
 * @return The number of false positives found in this experiment.
 */
public int getSMFalsePos(){
    return sm_false_pos;
}//getSMFalsePos()

/**
 * Accessor for the sm_false_neg value.
 *
 * @return The number of false negatives found in this experiment.
 */
public int getSMFalseNeg(){

```

```

        return sm_false_neg;
    }//getSMFalseNeg()

/**
 * Accessor for the sm_found value.
 *
 * @return The number of supermutants found in this experiment.
 */
public int getSMFound(){
    return sm_found;
} //getSMFound()

/**
 * Accessor for the minimum value.
 *
 * @return This experiment's minimum value.
 */
public double getMin(){
    return min;
} //getMin()

/**
 * Accessor for the maximum value.
 *
 * @return This experiment's maximum value.
 */
public double getMax(){
    return max;
} //getMax()

/**
 * Accessor for the mean value.
 *
 * @return This experiment's mean value.
 */
public double getMean(){
    return mean;
} //getMean()

/**
 * Accessor for the standard deviation value.
 *
 * @return This experiment's standard deviation.
 */
public double getStdDev(){

```

```

        return stdDev;
    }//getStdDev()

    public String getPrintout(){
        return printout;
    }//getPrintout()

} //class ExpData

```

Experiment:

```

import cern.jet.random.*;
import java.lang.Math;
import java.util.Arrays;
import java.util.Date;
import java.util.StringTokenizer;
import java.lang.StringBuffer;
import java.io.*;

/**
 * Class Experiment
 *
 * The Experiment class represents a simulated lab experiment,
 * generally used to simulate pooling experiments for Directed Evolution.
 *
 * @author <a href="mailto:cspencer@cc.gatech.edu">Cody Spencer</a>
 */
public class Experiment {

    /**
     * *****
     * ***** INSTANCE VARIABLES *****
     * *****
     *
     * Activity Ratio being used.
     *
     * NOTE - This needs to be implemented better.
     */
    private int AR;

    private double MIN_ACT;

    /**
     * Oversampling ratio being used.
     *

```

```

* 1.0 means no oversampling.
*/
private double OR = 1.0;

/**
 * Average number of cells expected to be in each well.
 */
private int mean_cells;

/**
 * Percent of wells to evaluate using std data analysis.
 */
private double top_pct;

/**
 * CV of assay being used.
 */
private double CV;

/**
 * Represents which activity distribution the cells are
 * being sampled from.
 *
 * Ex: SuperMutant distribution of cell activity.
 */
private ActivityDistribution ad;

/**
 * Represents which cell distribution the cells are
 * being sampled from.
 *
 * Ex: Poisson distribution of cells.
 */
private CellDistribution cell_dist;

/**
 * Uncertainty 1: Poisson of cells/well (stage 1 & 2)
 */
private boolean u_type1 = true;

/**
 * Uncertainty 2: Gaussian of assay error.
 */
private boolean u_type2 = true;

/**

```

```

* Uncertainty 3: Multinomial distribution of cell sampling.
*/
private boolean u_type3 = true;

/**
 * SuperMutant evaluation or not.
 */
private boolean SMeval;

private boolean GTeval;

private boolean print_all;

/*****
***** CONSTRUCTORS *****/
*****/

/* No need for specific constructors yet */
/* Perhaps sub-class for supermutant vs normal evaluation? */

/*****
***** INSTANCE METHODS *****/
*****/

/**
 * Simulates the running of a Directed Evolution
 * Experiment without using pooling (1 cell/well).
 * Since there is only one stage (no need for a resolution stage),
 * all of the wells are used in the first stage.
 *
 * @param total_wells Number of wells to use for this Experiment.
 * @return The desired data reaped from this Experiment.
 */
public ExpData run_exp_no_pool(int total_wells){

    /*cells per drop, random over poisson distribution*/
    int cells_per_well = 1;

    Well[] wells = new Well[total_wells];

    /*Sample the cells from the broth into the wells*/
    for(int i = 0; i < total_wells; i++){

        if(u_type1){

```

```

        cells_per_well = cell_dist.sample();
    }
    wells[i] = new Well(cells_per_well);

    //sample cells into wells
    for(int j=0; j < cells_per_well; j++){
        double ddd = ad.getActivity();
        if(SMeval){
            if(ddd >= AR)
                wells[i].setValue(j, new Cell(ddd,true));
            else
                wells[i].setValue(j, new Cell(ddd,false));
        }
        else
            wells[i].setValue(j, new Cell(ddd));

    }//for(j)
    if(u_type2){
        wells[i].assay(CV);
    }
    else{
        wells[i].assay(0.0);
    }
} //for(i)

/*sort the wells by assayed activity level*/
/*Uses merge-sort O(n*log(n))*/
Arrays.sort(wells);

/* SUPERMUTANT ANALYSIS OF DATA */
if(SMeval)
    return sm_data_analysis(wells, 1);
else if(GTeval)
    return greater_than_analysis(wells);
else
    return std_data_analysis(wells);

} //run_exp_no_pool(int)

/**
 * Simulates the running of a Directed Evolution
 * Experiment using pooling (more than 1 cell/well).
 *
 * @param stg1_wells Number of wells to use for the first stage.

```

```

* @param stg2_wells Number of wells to use for the second stage.
* @return The desired data reaped from this Experiment.
*/
public ExpData run_exp(int stg1_wells, int stg2_wells){
    return exp_stage1(stg1_wells,stg2_wells);
} //run_exp(int,int)

/**
* Pooling stage.
*
* @param stg1_wells Number of wells to use for the first stage.
* @param stg2_wells Number of wells to use for the second stage.
* @return The desired data reaped from this Experiment.
*/
private ExpData exp_stage1(int stg1_wells, int stg2_wells){

    /*desired cells per drop*/
    int cells_per_well = mean_cells;

    Well[] stage1 = new Well[stg1_wells];
    Well[] stage2 = new Well[stg2_wells];

    /*SAMPLING STAGE (stage1)*/
    for(int i = 0; i < stg1_wells; i++){

        /*Uncertainty of cell distribution*/
        if(u_type1){
            cells_per_well = cell_dist.sample();
        }

        /*create Well that holds specified # of cells*/
        stage1[i] = new Well(cells_per_well);

        //sample cells into wells
        for(int j=0; j < cells_per_well; j++){
            double ddd = ad.getActivity();
            if(SMeval){
                if(ddd >= AR)
                    stage1[i].setValue(j, new Cell(ddd,true));
                else
                    stage1[i].setValue(j, new Cell(ddd,false));
            }
            else
                stage1[i].setValue(j, new Cell(ddd));
        } //for(j)
    }
}

```

```

    /*How to assay*/
    if(u_type2){
        stage1[i].assay(CV);
    }
    else{
        stage1[i].assay(0.0);
    }

} //for(i)

/*If using the two-stage approach, uncomment line below*/
return exp_stage2(stage1, stage2, mean_cells);

/*If using the two-stage approach, comment-out line below*/
//return sm_data_analysis(stage1, mean_cells);
} //exp_stage1(int,int)

/**
 * Resolution stage of DE.
 *
 * @param stage1 Wells from stage 1.
 * @param stage2 Wells used in resolution stage (already allocated).
 * @param mean_cells Average cells per well in stage 1.
 * @return The desired data reaped from this Experiment.
 */
private ExpData exp_stage2(Well[] stage1, Well[] stage2, int mean_cells){

    /*sort the wells in stage1 by assayed activity level*/
    /*Uses merge-sort O(n*log(n))*/
    Arrays.sort(stage1);

    /*variables to keep track of which wells have been sampled*/
    int cell_num = 0;
    int well_num = stage1.length - 1;
    int oversamples = (int)(mean_cells * OR); //how much to oversample

    /*RESOLUTION STAGE (stage2)*/

    for(int k = 0; k < stage2.length; k++){

        if(cell_num == oversamples){
            well_num --;
            cell_num = 0;
        } //if
    }
}

```

```

    if(well_num < 0)
        well_num = stage1.length - 1;

    //sample
    int samples = 1;
    if(u_type1){
        samples = Poisson.staticNextInt(1.0);
    }//if

    stage2[k] = new Well(samples);

    for(int r=0; r < samples; r++){
        if(u_type3)
            stage2[k].setValue(r, stage1[well_num].sampleCell());
        else
            stage2[k].setValue(r, stage1[well_num].getCell(cell_num));
    }//for(r)

    if(u_type2){
        stage2[k].assay(CV);
    }
    else{
        stage2[k].assay(0.0);
    }

    cell_num++;
} //for(k)

/* ANALYZE DATA */
if(SMeval)
    return sm_data_analysis(stage2, mean_cells);
else if(GTeval)
    return greater_than_analysis(stage2);
else
    return std_data_analysis(stage2);
} //exp_stage2(Well[], Well[], int)

/**
 * Standard Analysis of top 3% of wells.
 * Mean, Std Dev, High and Low.
 *
 * @param wells The wells to analyze.
 * @return Mean, Std. Dev, High and Low of best 3% of wells.
 */
private ExpData std_data_analysis(Well[] wells){

```

```

String well_print = null;
if(print_all)
    well_print = "WELL ACTIVITIES\n-----\n";

/*sort the wells in stage2 by assayed activity level*/
Arrays.sort(wells);

/*get the top 3% of cells*/
int top_cells = (int)(0.01*top_pct*wells.length);
double top[] = new double[top_cells];
for(int q = 0; q < top.length; q++){
    top[q] = wells[(wells.length - q) - 1].getActivity();
    if(print_all){
        well_print += MathHelper.formatDouble(top[q],3) + ", ";
        if((q%10)==9)
            well_print += "\n";
    }
}
}

/* Analyze data */
double dmean = MathHelper.mean(top);
double stdev = MathHelper.standardDeviation(top, dmean);
return new ExpData(dmean, stdev, top[top.length-1], top[0],
    well_print);
}

private ExpData greater_than_analysis(Well[] wells){
    int num_found = 0;
    int false_pos = 0;
    int false_neg = 0;
    Well w = null;
    String well_print = null;

    if(print_all)
        well_print = "WELL ACTIVITIES\n-----\n";

    for(int i=0; i < wells.length; i++){
        w = wells[i];
        if(w.getActivity() >= MIN_ACT){
            num_found++;
            if(w.getRealActivity() < MIN_ACT)
                false_pos++;
        }
    }
}

```

```

else if(w.getRealActivity() >= MIN_ACT){
    false_neg++;
}

if(print_all){
    well_print += MathHelper.formatDouble(w.getActivity(),3) +
        ", ";
    if((i%10)==9)
        well_print += "\n";
}
} //for(i)

return new ExpData(num_found, false_pos, false_neg, well_print);
} //greater_than_analysis(Well[])

/**
 * Supermutant Analysis
 *
 * @param wells The wells to analyze.
 * @param mean_cells Average cells per well in stage 1.
 * @return Number of SuperMutants found in wells.
 */
private ExpData sm_data_analysis(Well[] wells, int mean_cells){
    int num_found = 0;
    int false_pos = 0;
    int false_neg = 0;
    String well_print = null;
    if(print_all)
        well_print = "WELL ACTIVITIES\n-----\n";

    //May want to look at when AR < mean_cells
    //double min_act = ((AR - 1)/mean_cells) + 1;
    double min_act = AR * (1.0 - 3*CV);

    Well w;

    for(int i=0; i < wells.length; i++){
        w = wells[i];

        if(w.getActivity() >= min_act){
            num_found++;
            if(!w.containsSupermutant()){
                false_pos++;
                //System.out.println("Act: "+w.getRealActivity());
                //System.out.println("MAct:"+w.getActivity());
            }
        }
    }
}

```

```

    }
    else{
        if(w.containsSupermutant()){
            false_neg++;
        }
    }
    if(print_all){
        well_print += MathHelper.formatDouble(w.getActivity(),3) +
            ", ";
        if((i%10)==9)
            well_print += "\n";
    }
} //for(i)

//System.out.println("MEM_MID=" + getMemoryKB());

return new ExpData(num_found, false_pos, false_neg, well_print);
} //sm_data_analysis(Well[])

/**
 * Supermutant Analysis - find how many AR=2 at end
 *
 * @param wells The wells to analyze.
 * @param mean_cells Average cells per well in stage 1.
 * @return Number of SuperMutants found in wells.
 */
private ExpData sm2_data_analysis(Well[] wells, int mean_cells){

    int num_found = 0;
    Well w;
    Cell[] c;

    for(int i=0; i < wells.length; i++){
        w = wells[i];
        c = w.getAllCells();
        if(c != null){
            for(int j=0; j < c.length; j++){
                if(c[j].getActivity() == 2.0)
                    num_found++;
            } //for(j)
        }
    } //for(i)
    return new ExpData(num_found, null);
} //sm2_data_analysis(Well[])

```

```

/**
 * Modifier method for the u_type1 field.
 *
 * @param on Whether or not to test this uncertainty type.
 */
public void setUTYPE1(boolean on){
    this.u_type1 = on;
}

/**
 * Modifier method for the u_type2 field.
 *
 * @param on Whether or not to test this uncertainty type.
 */
public void setUTYPE2(boolean on){
    this.u_type2 = on;
}

/**
 * Modifier method for the u_type3 field.
 *
 * @param on Whether or not to test this uncertainty type.
 */
public void setUTYPE3(boolean on){
    this.u_type3 = on;
}

/**
 * Accessor method for the mean_cells field.
 *
 * @return The average number of cells per well.
 */
public int getMeanCells(){
    return mean_cells;
} //getMeanCells()

/**
 * Modifier method for the mean_cells field.
 *
 * @param cells The average number of cells per well.
 */
public void setMeanCells(int cells){
    this.mean_cells = cells;
} //setMeanCells(int)

```

```

/**
 * Modifier method for the ad field.
 *
 * @param ad The new activity distribution to use for this Experiment.
 */
public void setActivityDistribution(ActivityDistribution ad){
    this.ad = ad;
} //setActivityDistribution(ActivityDistribution)

/**
 * Modifier method for the cell_dist field.
 *
 * @param cd The new cell distribution to use for this Experiment.
 */
public void setCellDistribution(CellDistribution cd){
    cell_dist = cd;
} //setCellDistribution(CellDistribution)

/**
 * Modifier method for the AR field.
 *
 * @param ratio The new activity ratio to use for this Experiment.
 */
public void setActivityRatio(int ratio){
    AR = ratio;
} //setActivityRatio(int)

public void setCV(double CV){
    this.CV = CV;
} //setCV(double)

public void setSMEval(boolean eval){
    this.SMEval = eval;
}

public void setTopPct(double top_pct){
    this.top_pct = top_pct;
}

public void setPrintAll(boolean printall){
    this.print_all = printall;
}

public void setGTeval(boolean eval){

```

```

        this.GTeval = eval;
    }

    public void setMIN_ACT(double min_act){
        this.MIN_ACT = min_act;
    }

    /**
     * *****
     * ***** CLASS METHODS *****
     * *****
     *
     * Runs multiple supermutant experiments with set parameters.
     * Modify hard-coded numbers to set up your experiment.
     *
     * NOTE - Need to modulize this... interface?
     */
    public static void run_main(ActivityDistribution ad, int act_ratio,
                                double assay_acc, double sm_ratio,
                                int num_plates, int plate_size, int cpw,
                                String output_file, int num_exps,
                                boolean sm_eval, double top_pct,
                                boolean print_all, boolean gt_eval,
                                double min_act){
        FileWriter fw = null;
        PrintWriter pw = null;

        int stg1_wells = num_plates*plate_size;
        int stg2_wells = stg1_wells;

        ExpData ed1 = null;
        ExpData ed2 = null;

        /*SUPERMUTANT DATA ANALYSIS*/
        double[] count_e1 = new double[num_exps];
        double[] count_e2 = new double[num_exps];
        double[] count_fp1 = new double[num_exps];
        double[] count_fp2 = new double[num_exps];
        double[] count_fn1 = new double[num_exps];
        double[] count_fn2 = new double[num_exps];

        /*STANDARD DATA ANALYSIS*/
        double[] mean_e1 = new double[num_exps];
        double[] sd_e1 = new double[num_exps];
        double[] max_e1 = new double[num_exps];
        double[] min_e1 = new double[num_exps];

```

```

double[] mean_e2 = new double[num_exps];
double[] sd_e2 = new double[num_exps];
double[] max_e2 = new double[num_exps];
double[] min_e2 = new double[num_exps];
String printout1 = null;
String printout2 = null;

```

```

ActivityDistribution ad1;
CellDistribution cd1 = new PoissonDistribution(cpw);
Experiment e1 = new Experiment();
e1.setUTYPE1(true);
e1.setUTYPE2(true);
e1.setUTYPE3(true);
e1.setSMEval(sm_eval);
e1.setPrintAll(print_all);
e1.setTopPct(top_pct);
e1.setMeanCells(cpw);
e1.setCellDistribution(cd1);
e1.setActivityDistribution(ad);
e1.setActivityRatio(act_ratio);
e1.setCV(assay_acc);
e1.setGTEval(gt_eval);
e1.setMIN_ACT(min_act);

```

```

try{
    /*Open file and write header info*/
    fw = new FileWriter(output_file);
    pw = new PrintWriter(fw);
    pw.println(new Date().toString());
    pw.println("");
    pw.println("");
    pw.println("-----");
    pw.println("Assay CV = " + assay_acc);
    pw.println("Mean Cells/Well = " + cpw);
    pw.println("# plates per round = " + num_plates);
    pw.println("# wells per plate = " + plate_size);
    pw.println(ad.toString());

    for(int d=0; d < num_exps; d++){
        if(cpw==1){
            ed1 = e1.run_exp_no_pool(stg1_wells+stg2_wells);
            ed2 = e1.run_exp_no_pool(stg1_wells+stg2_wells);
        }
        else{

```

```

        ed1 = e1.run_exp(stg1_wells,stg2_wells);
        ed2 = e1.run_exp(stg1_wells,stg2_wells);
    }

    if(sm_eval){
        count_e1[d] = ed1.getSMFound();
        count_e2[d] = ed2.getSMFound();
        count_fp1[d] = ed1.getSMFalsePos();
        count_fp2[d] = ed2.getSMFalsePos();
        count_fn1[d] = ed1.getSMFalseNeg();
        count_fn2[d] = ed2.getSMFalseNeg();
        printout1 = ed1.getPrintout();
        printout2 = ed2.getPrintout();
    }
    else if(gt_eval){
        count_e1[d] = ed1.getSMFound();
        count_e2[d] = ed2.getSMFound();
        count_fp1[d] = ed1.getSMFalsePos();
        count_fp2[d] = ed2.getSMFalsePos();
        count_fn1[d] = ed1.getSMFalseNeg();
        count_fn2[d] = ed2.getSMFalseNeg();
        printout1 = ed1.getPrintout();
        printout2 = ed2.getPrintout();
    }
    else{
        mean_e1[d] = ed1.getMean();
        sd_e1[d] = ed1.getStdDev();
        min_e1[d] = ed1.getMin();
        max_e1[d] = ed1.getMax();
        mean_e2[d] = ed2.getMean();
        sd_e2[d] = ed2.getStdDev();
        min_e2[d] = ed2.getMin();
        max_e2[d] = ed2.getMax();
        printout1 = ed1.getPrintout();
        printout2 = ed2.getPrintout();
    }
}

}

if(sm_eval){
    double mean1 = MathHelper.mean(count_e1);
    double sd1 = MathHelper.standardDeviation(count_e1, mean1);
    double fp1 = MathHelper.mean(count_fp1);
    double fn1 = MathHelper.mean(count_fn1);

```

```

double mean2 = MathHelper.mean(count_e2);
double sd2 = MathHelper.standardDeviation(count_e2, mean2);
double fp2 = MathHelper.mean(count_fp2);
double fn2 = MathHelper.mean(count_fn2);

pw.println("Counting number of supermutants found");
pw.println("First " + num_exps + " simulations");
pw.println(" Mean: " + mean1);
pw.println(" S.Dev: " + sd1);
pw.println(" F.Pos: " + fp1);
pw.println(" F.Neg: " + fn1);
pw.println("");
pw.println("Second " + num_exps + " simulations");
pw.println(" Mean: " + mean2);
pw.println(" S.Dev: " + sd2);
pw.println(" F.Pos: " + fp2);
pw.println(" F.Neg: " + fn2);
pw.println("");
pw.println("-----");
pw.println("");
pw.println("");
if(print_all){
    pw.println("");
    pw.println(printout1);
    pw.println("");
    pw.println(printout2);
}
}
else if(gt_eval){
double mean1 = MathHelper.mean(count_e1);
double sd1 = MathHelper.standardDeviation(count_e1, mean1);
double fp1 = MathHelper.mean(count_fp1);
double fn1 = MathHelper.mean(count_fn1);

double mean2 = MathHelper.mean(count_e2);
double sd2 = MathHelper.standardDeviation(count_e2, mean2);
double fp2 = MathHelper.mean(count_fp2);
double fn2 = MathHelper.mean(count_fn2);

pw.println("Counting number of bugs with activity greater "
    + "than " + min_act);
pw.println("First " + num_exps + " simulations");
pw.println(" Mean: " + mean1);
pw.println(" S.Dev: " + sd1);
pw.println(" F.Pos: " + fp1);
pw.println(" F.Neg: " + fn1);

```



```

        pw.println(printout2);
    }
    pw.println("-----");
    pw.println("");
    pw.println("");
}
pw.close();
fw.close();
}catch(Exception e){
    e.printStackTrace();
}finally{
    try{
        pw.close();
    }catch(Exception e2){/*do nothing*/ }
}finally

} //run_main()

/**
 * Used to optimize plates used in each stage of DE.
 *
 * NOTE - Need to update this method for structural changes made.
 *
 * @param total_plates Total # of plates available.
 * @param plate_size Number of wells per plate.
 * @param cd CellDistribution to use for experiments.
 * @param ad ActivityDistribution to use for experiments.
 * @param output_file Name of file to write to.
 */
public static void optimizePlates(int total_plates,
                                  int plate_size,
                                  CellDistribution cd,
                                  ActivityDistribution ad,
                                  String output_file){

    FileWriter fw = null;
    PrintWriter pw = null;

    ExpData ed = null;
    Experiment e1 = new Experiment();
    //e1.setMeanCells(cpw);
    e1.setActivityDistribution(ad);
    e1.setCellDistribution(cd);
    e1.setUTYPE1(true);
    e1.setUTYPE2(true);
    e1.setUTYPE3(true);

```

```

int num_exps = 50;
int num_rounds = 10;
int upper_plates;
int lower_plates;
double upper_mean;
double lower_mean;

double[] sd = new double[num_exps];
double[] mean = new double[num_exps];

int[] plates = new int[num_rounds];
double[] sd_results = new double[num_rounds];
double[] mean_results = new double[num_rounds];

//start at top
int stg1_plates = total_plates - 1;
int stg2_plates = 1;
upper_plates = stg1_plates;

for(int d=0; d < num_exps; d++){
    ed = e1.run_exp(stg1_plates*plate_size,stg2_plates*plate_size);
    sd[d] = ed.getStdDev();
    mean[d] = ed.getMean();
}

//for(d)

plates[0] = stg1_plates;
sd_results[0] = MathHelper.mean(sd);
mean_results[0] = MathHelper.mean(mean);
upper_mean = mean_results[0];

//check bottom
stg1_plates = (int)(.5 * total_plates);
stg2_plates = total_plates - stg1_plates;
lower_plates = stg1_plates;

for(int d=0; d < num_exps; d++){
    ed = e1.run_exp(stg1_plates*plate_size,stg2_plates*plate_size);
    sd[d] = ed.getStdDev();
    mean[d] = ed.getMean();
}

//for(d)

plates[1] = stg1_plates;

```

```

sd_results[1] = MathHelper.mean(sd);
mean_results[1] = MathHelper.mean(mean);
lower_mean = mean_results[1];

for(int i=2; i < num_rounds; i++){

    stg1_plates = (upper_plates + lower_plates)/2;
    stg2_plates = total_plates - stg1_plates;

    for(int d=0; d < num_exps; d++){
        ed = e1.run_exp(stg1_plates*plate_size,stg2_plates*plate_size);
        sd[d] = ed.getStdDev();
        mean[d] = ed.getMean();

    }//for(d)

    plates[i] = stg1_plates;
    sd_results[i] = MathHelper.mean(sd);
    mean_results[i] = MathHelper.mean(mean);

    if(upper_mean > lower_mean){
        lower_mean = mean_results[i];
        lower_plates = stg1_plates;
    }
    else {
        upper_mean = mean_results[i];
        upper_plates = stg1_plates;
    }
} //for(i)

try{
    fw = new FileWriter(output_file, true);
    pw = new PrintWriter(fw);

    pw.println("Plate Optimizer");
    pw.println("Bisectional search");
    pw.println("");
    //pw.println("CPW = " + cpw);
    pw.println("Total Plates = " + total_plates);
    pw.println("");
    pw.println("Stage1 StdDev Mean");
    for(int i = 0; i < num_rounds; i++) {
        pw.println(" " + plates[i] + " " +
            MathHelper.formatDouble(sd_results[i], 3) + " " +
            MathHelper.formatDouble(mean_results[i], 3));
    } //for(i)
}

```

```

        pw.println("");
        pw.println("");
        pw.println("*****");
        pw.println("");
        pw.println("");
        pw.close();
    }//try
    catch(Exception e){
        e.printStackTrace();
    }
    finally{
        try{
            pw.close();
        }catch(Exception e2){
            e2.printStackTrace();
        }
    }
}

} //optimizePlates(int,int,CellDistribution,ActivityDistribution,String)

public static long getTime(){
    return System.currentTimeMillis();
}

public static long getMemoryKB(){
    Runtime.getRuntime().gc();
    long max = Runtime.getRuntime().totalMemory();
    long r = Runtime.getRuntime().freeMemory();
    return (max - r)/1024;
}

public static void run_main_infile(String filename){
    ActivityDistribution act_dist = null;
    int act_ratio = 0;
    double sm_ratio = 0.0;
    double sm_percent = 0.0;
    double assay_acc = 0.0;
    double top_percent = 0.0;
    double min_act = 0.0;
    boolean supermutant = false;
    boolean print_all = false;
    boolean greater_than = false;
    int num_plates = 0;
    int plate_size = 0;
}

```

```

int cells_per_well = 0;
int num_exps = 0;
double[] act_bins;
double[] pct_bins;
String output_file = null;

File infile;
FileReader fr;
BufferedReader br;
StringTokenizer st;

try{
    infile = new File(filename);
    fr = new FileReader(infile);
    br = new BufferedReader(fr);

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    output_file = st.nextToken();

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    num_exps = Integer.parseInt(st.nextToken());

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    num_plates = Integer.parseInt(st.nextToken());

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    plate_size = Integer.parseInt(st.nextToken());

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    assay_acc = Double.parseDouble(st.nextToken());

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    cells_per_well = Integer.parseInt(st.nextToken());

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    String temp = st.nextToken();
    if(temp.equals("SUPERMUTANT")){
        act_ratio = Integer.parseInt(st.nextToken());
        supermutant = true;
    }
}

```

```

        greater_than = false;
        if(st.hasMoreTokens())
            if(st.nextToken().equals("PRINT_ALL"))
                print_all = true;
    }

else if(temp.equals("TOP_PCT")){
    top_percent = Double.parseDouble(st.nextToken());
    supermutant = false;
    greater_than = false;
    if(st.hasMoreTokens())
        if(st.nextToken().equals("PRINT_ALL"))
            print_all = true;
}

else if(temp.equals("GREATER_THAN")){
    min_act = Double.parseDouble(st.nextToken());
    supermutant = false;
    greater_than = true;
    if(st.hasMoreTokens())
        if(st.nextToken().equals("PRINT_ALL"))
            print_all = true;
}

/*GET ACTIVITY CURVE*/
st = new StringTokenizer(br.readLine());
st.nextToken();
temp = st.nextToken();
if(temp.equals("SUPERMUTANT")){

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    act_ratio = Integer.parseInt(st.nextToken());

    st = new StringTokenizer(br.readLine());
    st.nextToken();
    sm_ratio = Double.parseDouble(st.nextToken());
    sm_percent = 1.0/sm_ratio;
    act_dist = new SuperMutantDistribution(act_ratio,sm_percent);
}

else if(temp.equals("BINS")){
    int num_bins = Integer.parseInt(st.nextToken());
    act_bins = new double[num_bins];
    pct_bins = new double[num_bins];

    for(int i=0; i < num_bins; i++){

```

