Structure Prediction and Visualization in Molecular Biology

by

Christopher S. Poultney

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May, 2010

_____

Dennis E. Shasha

Dedication

For my parents, Dr. Sherman K. Poultney and Dr. Joan M. Poultney, who nurtured my joy of learning, demonstrated the value of life-long education, and never faltered in their support. For my wife, who believed in me even when I was unsure, and without whose love and support I could not possibly have come so far.

Acknowledgements

This work owes a great deal to Dennis Shasha, who has been a source of inspiration and guidance since we first met during my undergraduate studies, and without whose constant support and encouragement I would never have been able to bring this endeavor to a successful conclusion. I also owe a debt of gratitude to Rich Bonneau, Glenn Butterfoss, and Kevin Drew, who supplied excellent guidance, conversation, and ideas over the last few years; to my thesis auditors, Kris Gunsalus and Yann LeCun; and to the many additional collaborators who contributed to the temperature-sensitive mutant project, especially Kris Gunsalus, David Gresham, Michelle Gutwein, and David Jukam. I am also very grateful to my Sungear co-authors Dennis Shasha, Rodrigo A. Gutiérrez, Manpreet S. Katari, Miriam L. Gifford, W. Bradford Paley, and Gloria M. Coruzzi; and to Delin Yang and Eric Leung, who implemented GeneLights.

This work has been partly supported by the U.S. National Science Foundation and National Institutes of Health under grants IIS-0414763, IIS-9988234, DBI-0445666, N2010 IBN-0115586, N2010 IOB-0519985, N2010 DBI-0519984, DBI-0421604, and MCB-0209754. This support is greatly appreciated.

Table of Contents

List of Figures

List of Tables

Introduction

The tools of computer science can be a tremendous help to the working biologist. Two broad areas where this is particularly true are visualization and prediction. In visualization, the size of the data involved often makes meaningful exploration of the data and discovery of salient features difficult and time-consuming. Similarly, intelligent prediction algorithms can greatly reduce the lab time required to achieve significant results, or can reduce an intractable space of potential experiments to a tractable size.

We describe two specific projects designed with these precepts in mind. The first, a software program called Sungear (Poultney et al., 2007; Poultney and Shasha, 2009), is a visualization tool that shows the outcomes of N experiments on M entities. Sungear displays sets of entities – the intersections of these outcomes – within a generalization of the traditional Venn diagram, and provides an interactive, visual system to answer queries about the data. The second project addresses the issue of rational design of temperature-sensitive mutants. Temperature-sensitive mutations are a tremendously valuable research tool, but to date, most methods for generating such mutants involve large-scale random mutations followed by an intensive screening and characterization process. Surprisingly little work has been done in the area of predicting these mutations given the importance they play in many genetic and functional studies of gene function. Furthermore, the existing methods are based

entirely on secondary sequence, and do not use 3-dimensional structure. We describe a system that, given the structure of a protein of interest, uses a combination of protein structure prediction and machine learning to provide a ranked list of likely candidates for temperature-sensitive mutations.

Sungear

Introduction

Systems biology deals with integrating large-scale measurements of genes, proteins, ions and other molecular entities following perturbations and genetic dissection. A typical systems biology study involves the integrated analysis of these molecular entities from many experiments conducted under different conditions, where each condition represents the system-wide response to perturbations (genetic and environmental) relevant to the biology of the system. This common feature – many treatment conditions and entities numbering in the thousands or more – poses a challenge to finding the answers to many natural questions such as: Which subsets of conditions yield similar outcomes? Which entities respond similarly in many conditions? Which functional groupings of entities are most affected by which conditions? We describe a visualization tool to help answer these questions and complementary informatic tools to support alternate visualizations, and we explore the lessons learned in constructing that tool both from the software designer's and biologist's perspective.

Motivation

Systems biology presents particular challenges for data integration and visualization. For starters, -omic data (genomic, proteomic) has no natural visualization because there is no natural physical layout for much of the data. On the other hand, an -ome tends to have lots of instances (tens of thousands of genes or proteins), and lists of responses are often unwieldy without visual support. Unfortunately, presenting a holistic picture, no matter how attractive, does not guide biologists to detailed study of individual molecules, which remains an important step in the biological process. In our visualization work developed in collaboration with the plant biology community, we have adopted the following goals in an attempt to unify the goals of the two disciplines:

1) The software should be easy enough to use that even biology lab directors can use it (biology-inspired goal).
2) The software should respond to a concrete present need as well as anticipated future needs (biology-inspired goal).
3) The software should give not only a holistic view but also the means to locate individual targets – individual genes, functional categories, and experiments.
4) The software should be "stupid", that is agnostic to its current application, so it can be extended to others (computer science-inspired goal).

Goal 1 is not meant to be impudent. To us, a litmus test of good visualization software is that our busy biological colleagues should be able to use the software

easily even after having been away from it for some time or upon first being introduced to it. A second litmus test is that biologists, even those just starting to use the tool, should quickly come to discuss the data rather than the interface. Goal 2 follows from our early observation that biologists, like any other scientists, have their own goals and inevitably think of people in other disciplines as purveyors of tools. We computer scientists have the same view of DNA chip makers. The utility of those tools is what counts. Goal 3 addresses a frustration we have felt with visualizations such as heat maps, in which a false color picture appears to give substantial information, but fails to provide any type of "cognitive zoom" to a more informative display of individual genes or proteins (we use the phrase "cognitive zoom" to denote a "zoom in" to a greater level of informational detail, which may or may not be accompanied by a visual "zoom in" that enlarges a part of the visualization). Goal 4 derives from the computer science aesthetic: the best software is software that can adapt to many situations. For biologists, this amortizes the cognitive cost of learning a new tool over its many uses.

We present a visualization tool called Sungear (Poultney et al., 2007; Poultney and Shasha, 2009) through increasingly functional examples and then derive lessons for future visualization efforts.

Design Principles

The basic representation of much important system biological data is a list of molecules, commonly genes. Sometimes the genes have associated data, e.g. expression values, rankings, or connections to other genes. As useful as these lists are, it's difficult for people to gain insight from them, especially when the length of a list can grow into the thousands and beyond. But sifting through vast quantities of data to gain insight is central to the biological enterprise, even if what is "interesting" about the data can vary widely and depend on many things: organism, experimental design, data set, experimenter, etc. It is this relationship between M (thousands) of entities under N (several) conditions that we want to help visualize. The start of the visualization should be a picture that gives a holistic representation of what is going on, but the visualization should also support a cognitive and/or visual zoom to groups and then to individuals.

The Basics: Venn Diagrams

For three or fewer conditions, conventional Venn diagrams provide a good start. For example, consider a fictitious microarray experiment in which three input nutrients A, B, and C are either at a level 0 or at a level 1. We want to determine which genes are differentially expressed in which situation. One might represent the outcome as a Venn diagram (Figure 1).

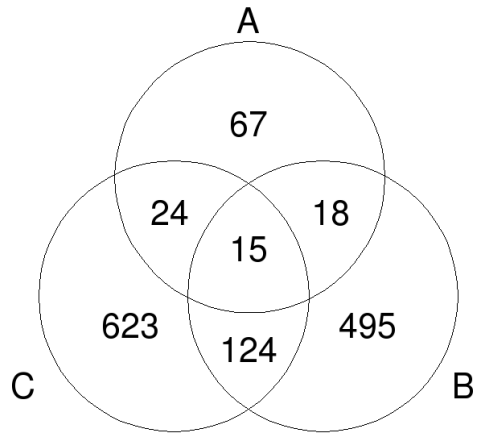

Figure 1 Venn diagram

Venn diagrams are a primary source of inspiration for our software. They give much information at a glance. For example, the region in which A and B overlap corresponds to genes that are differentially expressed when A and B are at level 1 but C is at level 0. A value drawn outside of all circles (not shown) represents the case where A, B, and C are at level 0.

Given this Venn diagram, one can label each of the eight possible combinations resulting from the binary levels of the three inputs. There are however some problems with this visualization:

1) The Venn diagram gives no visual cue of which combinations of inputs gives the largest or smallest result – it is necessary to read the numbers.
2) The Venn diagram cannot be extended to more than three inputs, without using far more complicated shapes that make the image hard to read.
3) The Venn diagram does not by itself tell us which genes are in which intersection.

The Sungear Plot: Higher-Dimensional Venn Diagrams

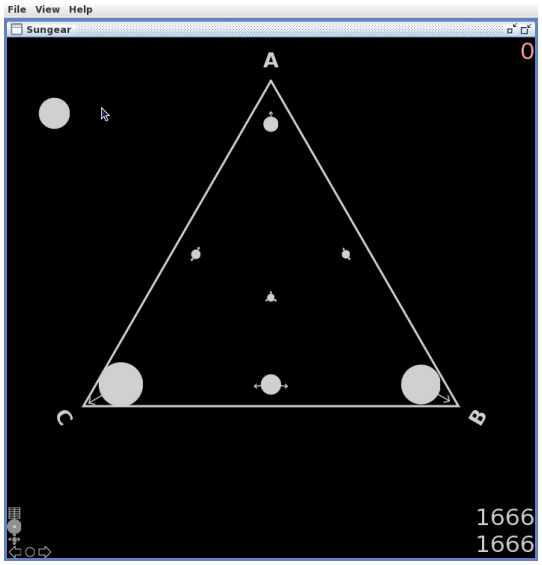

Figure 2 An equivalent Sungear representation to the earlier Venn diagram

Each circle is called a "vessel" and is pulled towards the "anchors" from which it comes. For example, the vessel between anchors A and B corresponds to genes that are differentially expressed when input A is at level 1, input B is at level 1, but input C is at level 0. The case where A, B, and C are all at level 0 is represented by the vessel located outside of the triangle.

Our software Sungear resolves the first objection in its basic window by representing A, B, and C as the vertices of a polygon (a triangle when there are only three inputs) and representing the intersections as circles of different sizes depending on the number of genes (Figure 2).

In contrast to Venn diagrams, the Sungear visualization extends easily to more inputs. Imagine for example that we have seven inputs, and we are interested in all possible binary combinations of those inputs (Figure 3).

The data used here (and in all Sungear plots except for Figure 2 and Figure 14) are from an analysis of hormone treatments of young Arabidopsis seedlings (Nemhauser et al., 2006) (see the Case Study below for more details). Figure 3 also shows the associated information that the full Sungear display provides: TAIR (Rhee et al., 2003) and GO (Gene Ontology Consortium, 2001) annotations, and the control panel.

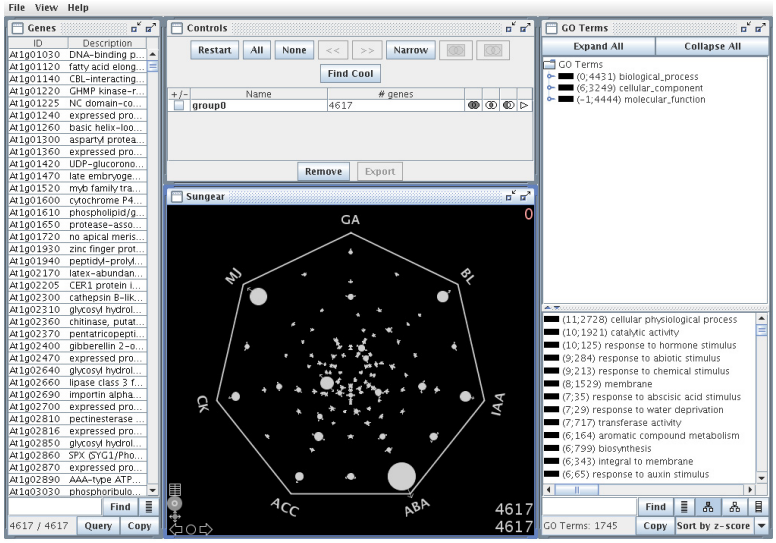

Figure 3 Sungear plot with seven inputs

The 105 vessels in the figure reflect the relative numbers of genes corresponding to each combination of conditions (128 combinations are possible; the 23 null combinations are not drawn). GA, BL, and IAA play the roles of A, B, and C, respectively, from Figure 2.

Exploring Associated Information

Figure 3 gives a holistic view of the data, but it is crowded enough to make it difficult to understand which vessel corresponds to which collection of anchors. For that reason, mousing over a vessel paints the associated anchors pink and specifies the number of genes corresponding to that vessel (Figure 4; arrows indicate highlighted anchors and vessels).

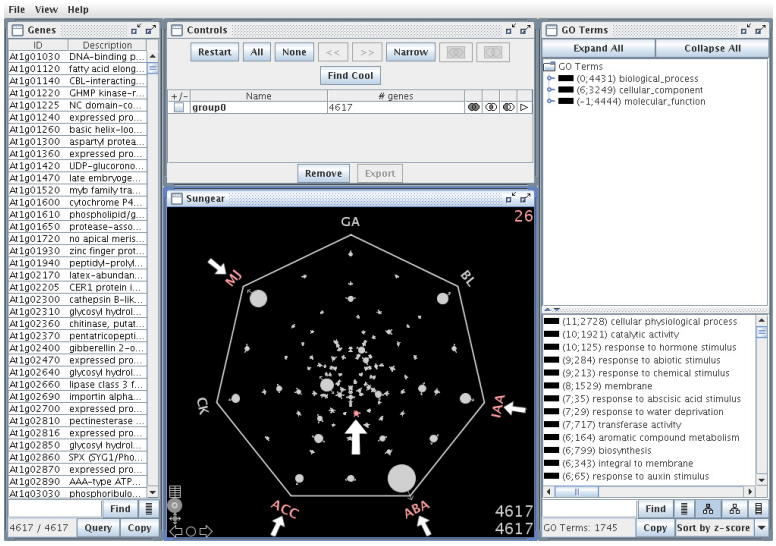

Figure 4 Mouse rollover highlighting

Moving the mouse over a vessel will highlight that vessel and its anchors in pink, and display the number of genes in the vessel at the upper right of the window. For clarity, the highlighted vessel and anchors have also been marked with arrows.

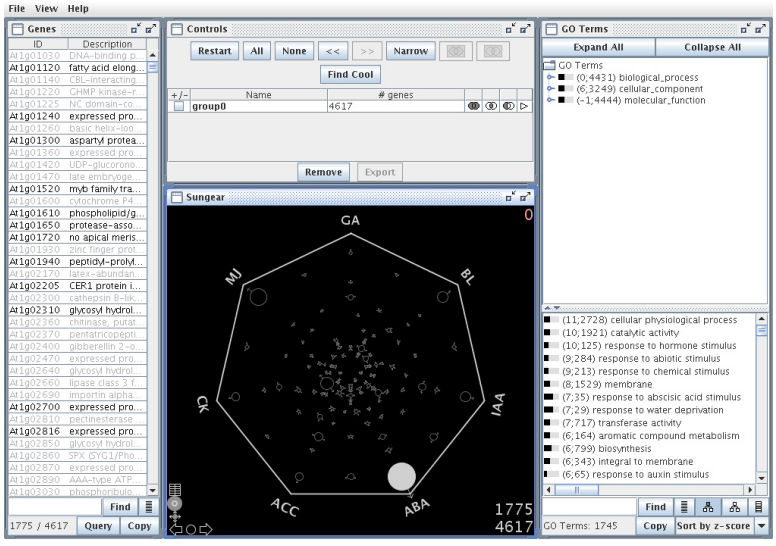

Figure 5 Single vessel selection

In the full Sungear display, clicking on a vessel yields the genes that correspond to that vessel as well as the GO terms corresponding to those genes.

Knowing how many genes are associated with each vessel does not tell us which genes they are, nor does it suggest what is special about those genes. To support that functionality, Sungear allows a click on a vessel to yield the genes and GO annotations that correspond to the set of conditions associated with the genes in that vessel (Figure 5).

Narrowing and Z-scores

Finding the genes and functional terms associated with a vessel may require the perusal of a long list. As an alternative, Sungear presents those functional terms that are significantly overrepresented, and displays a z-score for each GO term at the

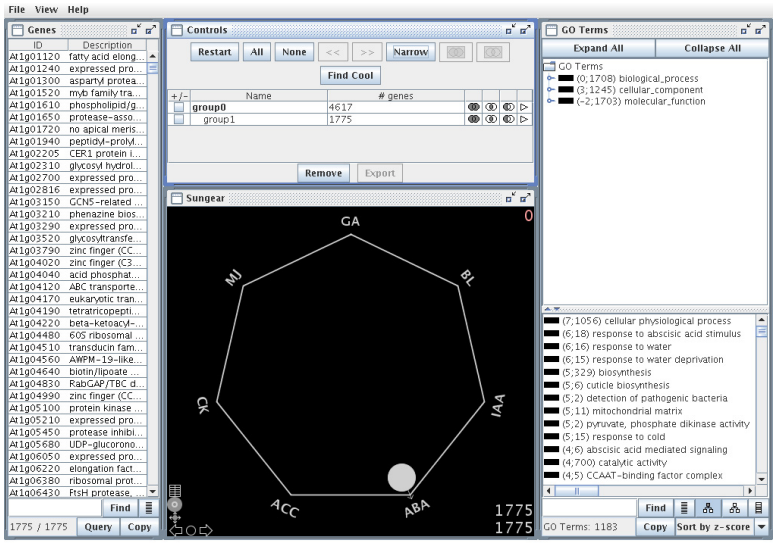

Figure 6 Narrowing and z-score updates

GO terms with recalculated z-scores after narrowing to the vessel selected in Figure 5. The three highest z-scores are for “cellular physiological process) (z-score 7; 1056 genes), “response to abscisic acid stimulus (6; 18), and “response to water” (6; 16).

top of the lower right hand window. Changing the selection will affect the z-scores, which are recalculated when Sungear "narrows" its working set to a current selection. By narrowing to the vessel selected in Figure 5, the z-scores are recalculated, and several of the top entries among the overrepresented terms involve abscisic acid (Figure 6). This is unsurprising in this case, since the ABA condition involves treatment with abscisic acid, and the vessel contains genes that responded only to that condition.

Besides clicking on a single vessel, one can also click on a single anchor (Figure 7). This corresponds to the query "Find all genes that are differentially expressed in this experiment, as well as the overrepresented functional terms."

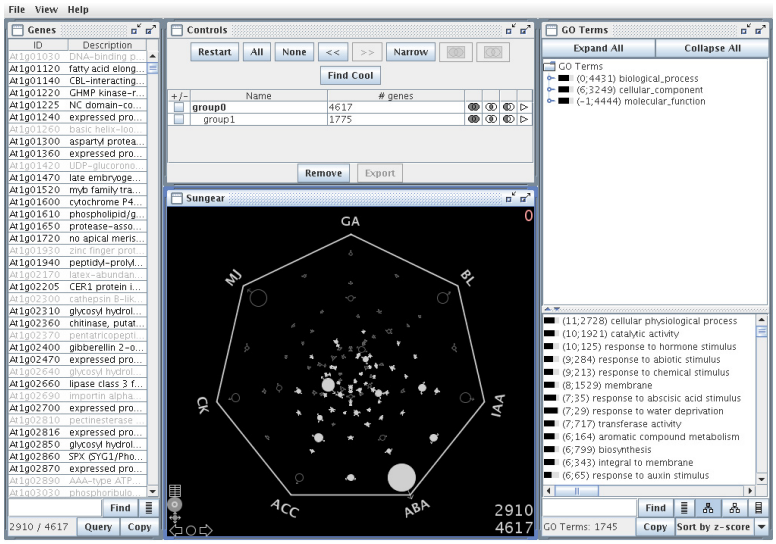

Figure 7 Single anchor selection

Clicking on anchor ABA leaves only those vessels that point to that anchor, indicating those vessels that require ABA to be at level 1 in order for any genes in those vessels to be differentially expressed.

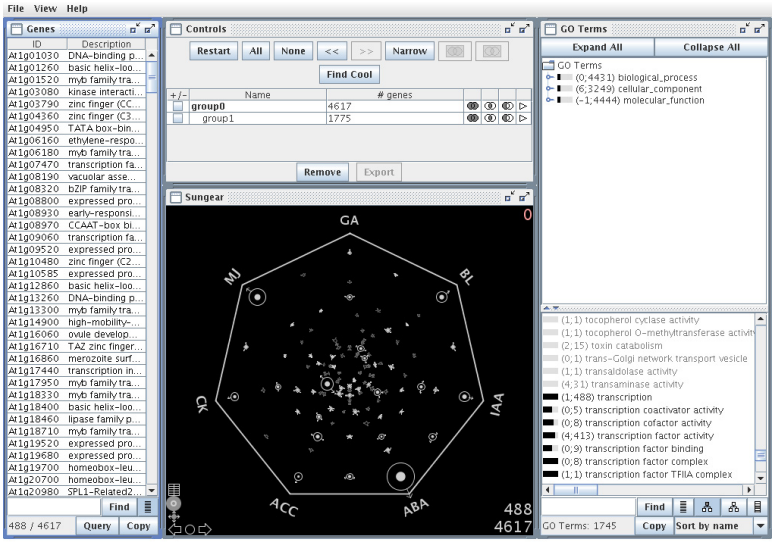

Figure 8 GO term selection

Clicking on the GO term “transcription” makes some vessels hollow and leaves a hollow annulus in other vessels, corresponding to the genes that do not descend from the term “transcription” in the GO hierarchy.

Similarly one can choose a GO term and get a sense of the number of genes in each combination of conditions that correspond to that term (Figure 8).

Z-scores were calculated as follows:

1) Find the number of genes associated directly or indirectly with each GO term *T*. This is an “offline” process that only need to be carried out when the mapping of GO terms to genes is updated:

$$p\_t = \frac{number\ of\ genes\ associated\ with\ T}{total\ number\ of\ genes\ in\ genome}$$

$$std(T) = \sqrt{p\_t \times (1 - p\_t)}$$

2) In Sungear, a data set will involve a subset *S* of the total number of genes in the genome. Using *N* to represent the number of genes in *S*, we define:

$$f\_t = \frac{number\ of\ genes\ in\ S\ associated\ with\ T}{N}$$

3) We can now calculate the z-score:

$$z\_score(T) = \frac{(f\_t - p\_t) \times \sqrt{N}}{std(T)}$$

Multi-Window Operations

Having seen how the choice of a single vessel, anchor, or GO term can influence the display, one can now ask how to combine such choices to answer more complex queries. For example, one may be curious about the transcription genes differentially expressed when both ABA and BL are at level 1. Clicking on the GO

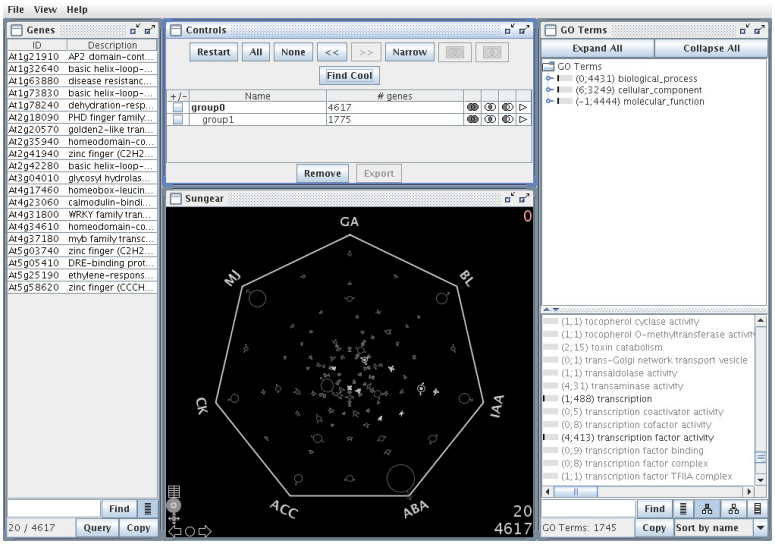

Figure 9 Set intersection

The result of using the set intersection operation to find genes associated to the GO term "transcription" and differentially expressed when both ABA and BL are at level 1.

term “transcription” and the anchors ABA and BL, and then clicking the “set intersect” icon yields the 20 genes in this 3-way intersection (Figure 9).

More complicated operations are possible. To find genes with a z-score above 10 that are differentially expressed when both ABA and BL are at level 1, first do a union operation over the 6 GO terms with z-scores over 10, and then perform an intersection between ABA and BL over the remaining genes (Figure 10). As a colleague once told us after a similar demonstration, “In 20 seconds, you have done what might take us two days to do.”

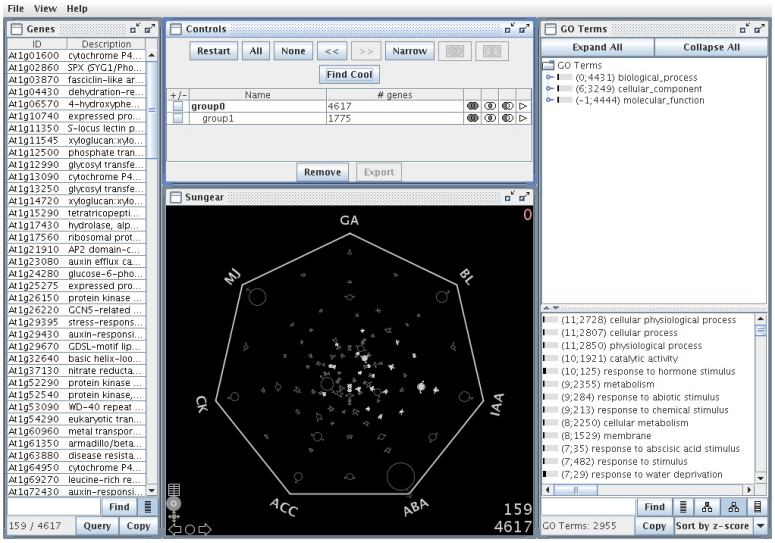

Figure 10 A two-stage operation

Shown is the result of taking the union of the top 6 GO terms, then the intersection with ABA and BL.

Extending Sungear

So far, we have imagined that each gene is either differentially expressed or not. Differential expression need not be the criterion for set membership – a researcher

can choose any binary criterion – but membership is a binary quantity. However, expression levels can vary significantly (as can most other measures) and it is worthwhile to extend Sungear to analyze such continuous data. One of the design principles of Sungear is that all windows communicate with one another through their effect on a single master list of genes. This means that to add a new window W, all that is required is that 1) window W can adjust itself when an action in another window W' changes the master list, and 2) window W can send changes to the master list when there is a change to W.

The module GeneLights, implemented by Delin Yang and Eric Leung, follows the rules above to allow exploration of continuous data. The idea is simple: for every anchor, there exists a range of expression values. Each anchor corresponds to a line segment whose endpoints represent expression values.

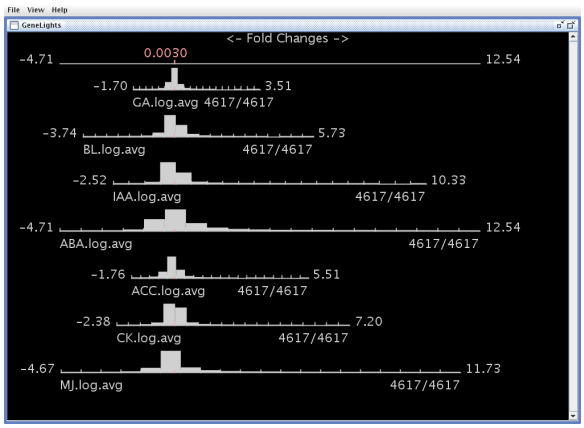

Figure 11 GeneLights

The basic GeneLights plot, showing log average fold change over the three time points that constitute each condition.

Superimposed on that line segment is a histogram composed of rectangles. The height of each rectangle corresponds to the number of genes in that range of expression values. Ranges of values can be chosen for one or more anchors and then linked with other windows in Sungear as illustrated below.

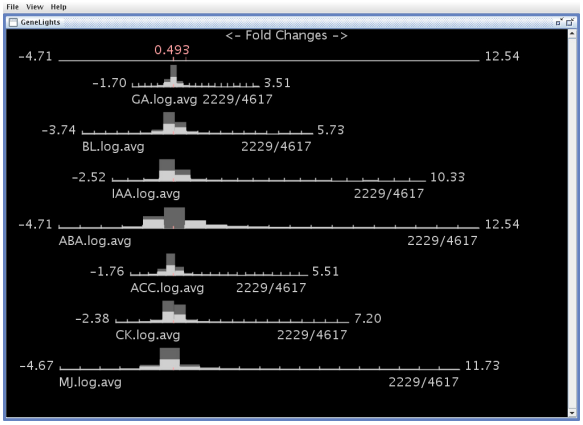

Figure 12 GeneLights selection

The GeneLights display after selecting all genes on the ABA line less than -0.5 or greater than 0.5 (log average differential expression greater than 0.5).

The basic GeneLights display (Figure 11) shows the line segments, histograms, and expression value ranges as described above. When a user selects two ranges on the ABA line, for example everything less than -0.5 and everything greater than 0.5, the result is a selection of all genes whose absolute log fold expression is greater than 0.5 for condition ABA (Figure 12). Since GeneLights is integrated into

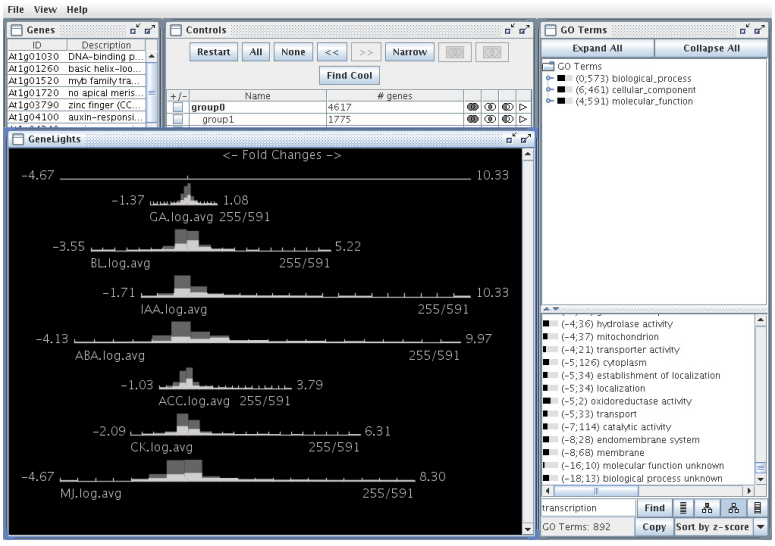

Figure 13 GeneLights and GO term intersection

The intersection of all genes differentially expressed by a factor of two in any experiment (log base 2 ranges less that -1.0 or greater than 1.0) with and GO term containing the word “transcription”. The active set was “narrowed” after selection the differentially expressed genes.

the Sungear framework, we can use it to perform more complex selection operations, such as finding all genes that have a GO annotation containing the word "transcription" and that are 2-fold differentially expressed in one or more conditions (Figure 13).

Sungear for Non-Genomic Data

So far, we have focused on a genomic application. Sungear however is agnostic to the application, making almost no assumptions about the type of data upon which it operates. The assumptions are minimal: Sungear operates on several sets of items and these items can also be classified into categories which may, like GO terms, be hierarchical. No assumptions are built into the software that this data pertains to a particular organism, or even that the data are of a biological nature. We have established a Sungear for baseball to demonstrate this point. Let us take a brief look at this application, just to underscore the point that any time a scientist has many entities and several conditions, regardless of the application, Sungear may be an appropriate tool.

Public repositories of baseball statistics such as The Baseball Archive (http://www.baseball1.com/statistics/) provide all the information necessary to adapt Sungear to display baseball information. In this baseball example, individual players are the Sungear entities, and the various teams are the categories, structured hierarchically by division and league. The first step is to prepare the Sungear files that

give player and team information, team hierarchy structure, and player/team correspondence (for Sungear's default Arabidopsis setup, these files would describe, respectively, TAIR and GO annotations, GO term parent-child relationships, and TAIR-to-GO correspondence). Next, a simple "experiment" data file is created based on the statistics from the 2004 season. Players are evaluated according to their performance in the four areas that constitute the Sungear plot (Figure 14), with membership in a set determined by a threshold. The categories are: batting average of .250 or above, 20 or more home runs, 50 or more runs batted in, and 10 or more stolen bases.

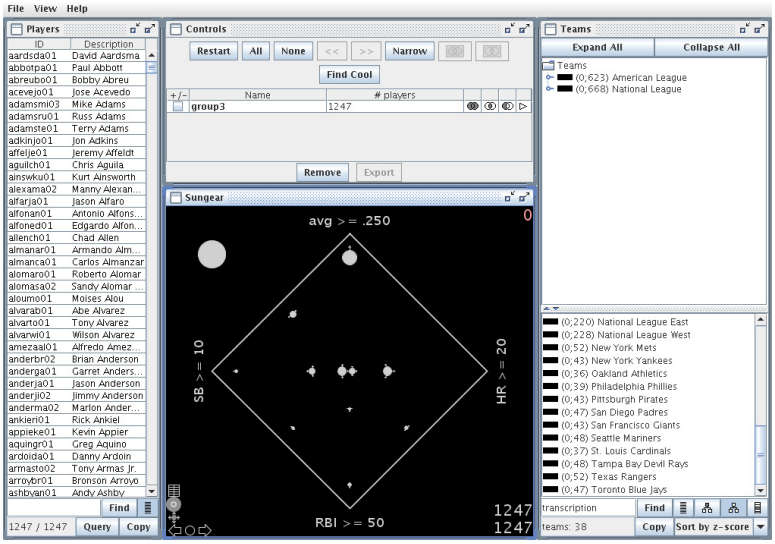

Figure 14 Sungear plot showing baseball data

The four anchors of the Sungear plot, clockwise from top, are: batting average of .250 or better (avg >= .250), 20 or more home runs (HR >= 20), 50 or more runs batted in (RBI >= 50), and 10 or more stolen bases (SB >= 10). The vessel outside the polygon represents the players who did not meet any of the above criteria in 2004 – the majority of players, in fact.

A quick perusal of the Sungear plot shows that 1) most players (835/1247) met none of the criteria during the 2004 season, and are represented by the vessel outside of the polygon, 2) only 23 players met all four criteria, with the American League claiming twice as many of those players as the National League (in fact, the American League champion Red Sox went on to win the World Series that season), and 3) 44 players played in both leagues during the course of the year, including one of the 23 four-criteria players, Carlos Beltran. This demonstrates that, after straightforward data preparation, Sungear is able to provide meaningful exploration of non-biological data.

A brief case study

We briefly show how Sungear can be used to help analyze experimental results. Nemhauser et al analyze AtGenExpress data collected by Yoshida et al under seven different hormone treatments at three fixed time points (descriptions available at http://Arabidopsis.org/info/expression/ATGenExpress.jsp). They then classify genes as belonging to one of four categories: upregulated at one or more time points, downregulated at one or more time points, upregulated at some time points and downregulated at others, or none of the above. This is a perfect starting point for Sungear analysis. We coalesced the three types of changes (down-regulated/up-regulated/complex) for each hormone; the genes we consider to be at level 1 for an anchor were upregulated or downregulated during at least one time point. This illustrates how our binary set membership criterion can vary depending on the

interests of the analyst. As mentioned earlier, all of the Sungear plots in this chapter (except for the Venn diagram example in Figure 2and the baseball example in Figure 14) use this data set.

Nemhauser et al investigate the hypothesis that there is a core growth-regulatory module that collects inputs from multiple hormones. A quick glance at the Sungear plot (Figure 15) shows that there are few target genes common to the hormones studied, and supports Nemhauser et al's conclusion that there is no core transcriptional growth-regulatory module. The largest vessels are around the periphery, where 1- and 2-anchor vessels are located, while the center, which corresponds to more multi-anchor vessels, contains only small vessels. The vessel statistics window shows that of the 4,617 genes in the experiment, 3,034 are

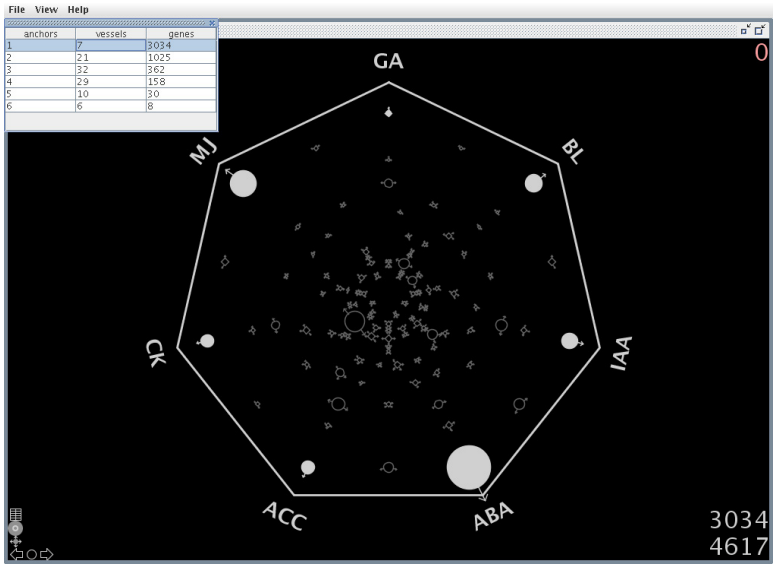

Figure 15 Sungear plot showing the vessel statistics window

Only vessels associated with exactly one anchor, and hence genes expressed in only one condition, are selected in the Sungear plot.

associated with only one vessel (Figure 15). In other words, the majority of genes in this experiment respond to only one hormone treatment.

Next we examine the GO annotations of the genes. Using the GO term search feature, we find that of the 98 genes whose annotations contain the phrase “cell wall”, 63 are expressed under only one condition, providing finer-grained support for the earlier statement (Nemhauser et al report 60 of 101 genes; we suspect that our annotation file version was slightly different). We can also use the z-scores associated with the GO terms to examine over- and under-representation of groups of genes. The GO term “metabolism” is strongly overrepresented, with a z-score of 9 (2,355 genes), while photosynthesis is slightly, if at all, overrepresented with a z-score of 1 (24 genes). This is similar to the Nemhauser et al survey of GO categories, which shows that genes annotated with “metabolism” are overrepresented in five of the experimental conditions, whereas the number of genes involved in photosynthesis was consistent with counts generated by a random distribution.

Sungear provides a shortcut to picking and examining GO terms that are expected to be interesting: the “Find Cool” feature. This feature searches for vessels that are highly enriched for any GO terms. Each vessel is scored relative to the entire genome (as if a “narrow” operation had been performed after selecting just that vessel), and any term associated with at least 2 genes in that vessel and with a z-score of 10 or more adds one to the “cool” score for that vessel. The “coolest” vessels are the ones with the highest scores. In this data set, the coolest vessel has a score of 11;

its 14 genes responded to GA, ABA, and MJ treatments. The highest-scoring GO terms contributing to the score are lactose catabolism and lactose metabolism (both with z-score 19). The biological significance of this vessel, if any, is as yet undetermined; but the system's capability of finding a combination of treatments that correspond to an overrepresented functionality can have many applications.

Combining Visualization Tools for Plant Systems Biology

In this section, we describe complementary tools that are in wide use among our biological colleagues. The tools we discuss include features such as the in-depth analysis of individual genes, visualization of expression profiles in different networks, and exploration of general networks. Table 1 lists the web sites for the visualization tools discussed in this chapter.

Software Tool	**URL**
Sungear	http://virtualplant.bio.nyu.edu/cgi-bin/sungear/index.cgi
MapMan	http://www.gabipd.org/projects/MapMan/
Genevestigator	http://www.genevestigator.com/
Cytoscape	http://www.cytoscape.org/
VirtualPlant	http://www.virtualplant.org/

Table 1 Websites for visualization tools discussed

As pointed out earlier, Systems Biology requires the integration of multiple experiments on multiple molecular entities. Sungear starts with a set of genes meeting

some measure of interest (e.g. expression greater than or equal to a certain level) for each experiment. It then compares the experiments by displaying vessels containing those genes and relating them to functional categories. As we have seen, a paradigmatic use of Sungear is to see which combinations of experiments affect genes having similar functional categories.

Other visualization tools choose different conceptual groupings and contextualization of data. MapMan (Thimm et al., 2004) shows gene expression values in the context of different pathways. Genevestigator (Zimmermann et al., 2004) shows "meta-profiles" of gene expression for several genes against a background chosen from a large database of experiments, and provides tools for working with these meta-profiles. Cytoscape (Shannon et al., 2003) supports network visualization, with many options for adding, analyzing, and visualizing node/edge annotations (including expression values); a plug-in framework allows for a wide variety of third-party extensions. VirtualPlant (Katari et al., 2009) acts as a central data repository and launching point for a collection of browsing and analysis tools. Each of these tools encourages a different type of data exploration.

MapMan

MapMan displays gene expression values in the context of a particular pathway (Thimm et al., 2004). That is, MapMan, at its core, associates expression values with a visual representation of a pathway or other process.

MapMan provides a list of pre-made pathways (currently around 45) for displaying expression data. Each pathway is arranged into a hierarchy of functional bins; there bins are assigned locations within the pathway image according to the pathway being represented. MapMan provides mappings from gene identifiers, such as Arabidopsis Genome Initiative (AGI) identifiers, to bins. When an expression data file is loaded, the value for each gene in the file is displayed in its assigned bin(s) according to the mapping file; a layout component then handles the organization of genes within a bin. Displaying the data is as simple as specifying a pathway file, an expression data file, and the appropriate mapping file. Only one expression data set can be shown on a pathway at a time, but multiple windows can mitigate this restriction.

Each gene in the data file is mapped to one or more bins (typically only one), and within each bin in the pathway, each gene's expression value is displayed using a continuous red-to-blue coloring (Figure 16). Bins can also be displayed as histograms of expression values of genes assigned to that bin. The MapMan display makes it easy to spot bins with a large number of highly over- or under-expressed genes; and the display as a whole gives a representation of where the genes in the data file are located in the overall pathway and how strongly each gene is expressed. Gene names, expression values, and other data are shown in a tooltip when the mouse is positioned over a gene, and displayed in the log window when the mouse is clicked. Further gene information is provided via links out to several web-based data sources (e.g. TAIR). A

separate window lists all the functional bins, with descriptions and associated information. The tool also computes the degree to which each bin's expression profile differs from that of the other bins; this provides the default sort order for all bins represented in the network, so that the bins most likely to be interesting are shown at the head of the list.

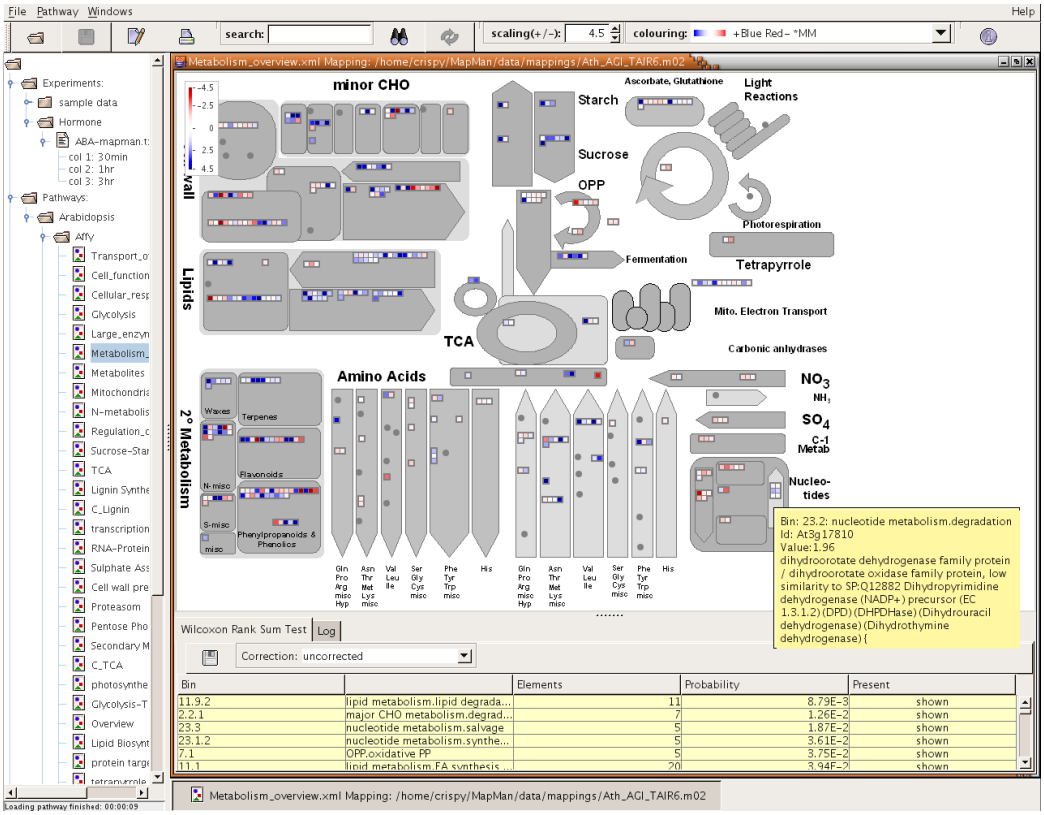

Figure 16 A MapMan pathway visualization

The main part of the screen shows gene expression data (log fold-change) for one experiment over the Metabolism_overview pathway. Under- and over-expression are represented by the degree of red or blue shading, respectively. The lower right of this window shows roll-over information about one gene and its enclosing bin. The window below the visualization gives information on the hierarchical bins, and the window to the left allows a choice of experimental data and pathway visualizations.

By way of comparison, the MapMan display in Figure 16 uses the same data as the "ABA" line of the GeneLights display in Figure 11. Whereas Sungear uses a histogram to represent the general pattern of expression in an experiment, MapMan represents the expression for each gene in the context of some larger pathway. Like Sungear, MapMan has a data-neutral architecture: although it is distributed with pathways and mapping for Arabidopsis, its text file-based architecture makes it a much more general tool for associating any type of continuous values with hierarchical locations in an image.

Here is a use scenario for MapMan: starting with a spreadsheet containing a list of genes and log-fold change in expression for one or more experiments, we export that list as a tab-delimited file. We then open MapMan, choose a pathway (e.g. Metabolism_overview) and the mapping appropriate for our data (e.g. AGI), and take a look at the display. We look, either in the pathway display or in the bin list, for one or more bins with an unusual pattern of under- or over-expressed genes. We can further examine these genes using one of the link-out data sources, or make a list of them for further analysis using tools such as Genevestigator.

Genevestigator

Genevestigator starts with a small number of genes specified by a user and gives information about those genes based on a set of microarray data also chosen by the user (Zimmermann et al., 2004). The available microarrays are annotated

according to four ontologies, which appear throughout Genevestigator: anatomy, development, stimulus, and mutation. The conceptual core of Genevestigator is the "meta-profile" of a gene, which is defined as its expression across the different categories within an ontology (the Genevestigator documentation refers to the categories within an ontology as "conditions", but we will use the word "categories" to avoid confusion with experimental conditions). For example, the "anatomy" meta-profile of a gene is its expression, grouped by anatomical structure, across the microarrays chosen by the user; the "stimulus" meta-profile of a gene is its expression across microarrays grouped by experimental stimulus.

Once an organism (there are currently 8 species available) and a set of microarrays have been chosen, Genevestigator provides four interrelated analysis tools: Meta-Profile Analysis, Biomarker Search, Clustering Analysis, and Pathway Projector. Starting with a list of genes provided by the user, the Meta-Profile Analysis tool gives a number of different visualizations of the meta-profiles of the genes of interest. Two windows show expression values of selected genes over the entire set of chosen microarrays, or a subset thereof, in more detail. Four additional windows show meta-profiles of the selected genes across categories in each of the four ontologies, with different display options (e.g. heat map or scatter plot). The Biomarker Search tool is designed for the case where the starting point is not a known list of genes but a desired meta-profile, such as differential expression in one or more tissues or up/down-regulation by different stimuli. Differential expression is determined by

comparing a set of target categories to a set of base categories over all selected microarrays in the microarray set. For example, selecting the “Development” tab, then selecting all stages as the base and seedling as the target, will specify a desired meta-profile of “differential expression in the seedling relative to its expression across all developmental stages.” After specifying a meta-profile, the tool will search for matching genes, which can then be added as groups for the Meta-Profile Analysis tool.

Meta-profiles derived using either the Meta-Profile Analysis or Biomarker Search tools can be further analyzed using the Clustering Analysis tool (with a meta-profile on the screen, simply switch to the Clustering Analysis tool, select some options, and click 'Run'). This tool can perform biclustering or hierarchical clustering (Figure 17) of genes and categories, with several different options (e.g. up/down regulation for biclustering, and distance measure for hierarchical clustering).

The fourth component, Pathway Projector, allows for visualization of expression data (“projection”) on different pathways. It is somewhat separate from the other three tools: it starts with a “comparison set” of expression according to base and target categories within an ontology (similar to the Biomarker Search tool) from one of the four ontologies. Genes differentially expressed in this comparison set are shown on a pre-determined pathway tree selected by the user; an interface is also provided for building a new network. While not as fully-featured as Cytoscape, the Pathway Projector provides quick network visualization within a familiar interface.

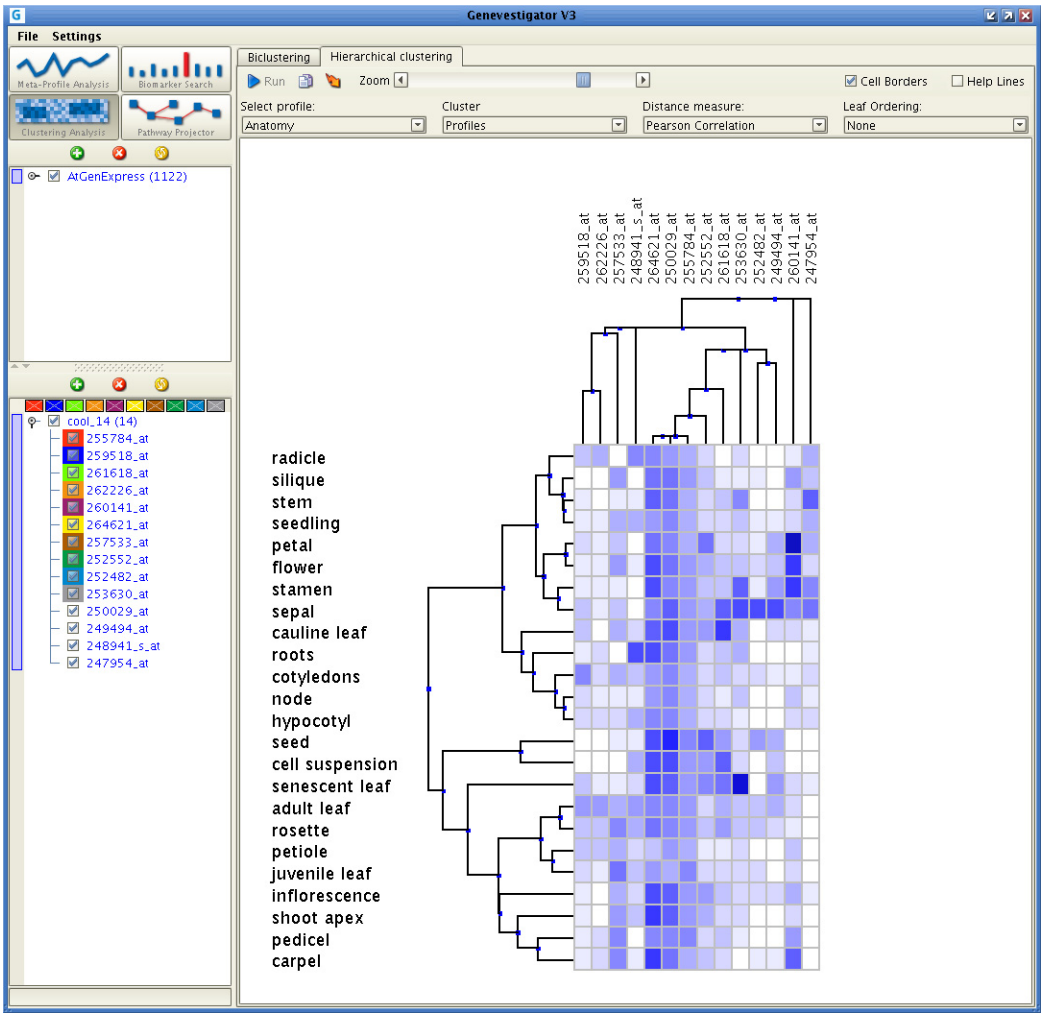

Figure 17 Genevestigator's Clustering Analysis tool

The main window shows 14 genes (from the "coolest" Sungear vessel in the Sungear case study), with hierarchical clustering. Expression values are shown from light to dark blue, with darker blue representing greater differential expression. Genes (across the top of the main window) are clustered according to their expression profile across anatomical structures, and categories from the stimuli ontology (along the left of the main window) are clustered by expression profile across genes. The microarray sets (all the AtGenExpress ATH1 22k arrays) are shown in the upper left window, and the query genes in the lower left window.

A use scenario for Genevestigator might be as follows: a small number of Arabidopsis genes are identified as interesting (e.g., membership in a "cool" vessel as determined by Sungear). In Genevestigator, we first specify all of the AtGenExpress microarrays as our microarray set, then enter the AGI IDs of the identified genes. We start with the "Meta-Profile Analysis" tool, and see a scatter plot of the expression values of the specified genes across all arrays. We can then examine the "meta-profile" of these genes according to different ontologies – in this case, we pick "Stimulus" as the ontology, and examine the heat map of expression of these genes across different treatments. Switching to the "Clustering Analysis" tool, we perform a hierarchical clustering of our genes of interest across the different stimuli. Noticing that several of our genes are strongly up-regulated for some subset of treatments, we switch to the "Biomarker Search" tool, and search for other genes that are strongly up-regulated under the same treatments. This provides a new list of genes, which can be added to the original list or used to form a new group, and we return to the "Meta-Profile Analysis" tool to continue our analysis.

Cytoscape

Cytoscape helps to visualize networks where the nodes and edges can be defined and annotated by the user (Shannon et al., 2003). A paradigmatic use of Cytoscape is to visualize regulatory networks among some subset of the genes of a genome. When used in concert with Sungear, one can imagine identifying a set of

genes through Sungear and then visualizing the protein-protein interaction edges in that set through Cytoscape. Cytoscape is a comprehensive tool with many options and plug-ins; we provide here only an introduction to its full range of capabilities.

Cytoscape is built around its visualization of networks. The networks can be, for example, protein-protein or protein-DNA interactions, with nodes representing either protein or DNA and an edge representing some type of interaction between two nodes. Networks can be created from scratch using Cytoscape's editing tools, but a more common route is to start with a network from a public source. Many such sources exist; for example, the Saccharomyces Genome Database (SGD project, http://www.yeastgenome.org/) and BIND (Bader et al., 2001) will both export interaction data in Cytoscape-compatible formats. A third option is to use a Cytoscape plug-in like cPath (Cerami et al., 2006) or Agilent Literature Search (Vailaya et al., 2005) to create a network based on a literature search for user-specified terms (e.g. a protein of interest).

Once a network has been loaded, it usually needs to be organized on the screen in a human-readable format. Cytoscape provides a wide range of layouts for this purpose: layouts based on spring forces, layouts that try to detect certain types of graph structure, annotation-based layouts, and many others. Which layout is best depends on the task and specific network topology. Once a layout is applied, networks are easily browsed: nodes and edges can be selected, and their attributes examined; and nodes can be searched for by ID.

As mentioned previously, nodes and edges may include additional information, referred to as attributes. As with the networks themselves, there are many available sources of node and edge attributes; one of the strengths of Cytoscape is its ability to incorporate external, publicly-available data sources. Examples of node attributes are a common name, which can then be used for clarity in the network display; an alternate identifier, such as the AGI identifier where the node ID is the probe set identifier, to allow cross-referencing with data indexed by AGI identifier; a set of Gene Ontology annotations; or expression values and p-values for a set of experiments. These attributes form the basis for Cytoscape's more powerful and interesting features. Once additional attributes have been loaded into the network, nodes and edges can be filtered based on their attributes. Filters can be used to select sets of nodes and edges comprising sub-networks, which can then be manipulated. For example, the sub-network comprised of all the edges of a certain type, along with their corresponding nodes, can be used to create a new network; or all nodes with a p-value above a threshold can be removed from the network.

Cytoscape can alter nearly every aspect of the network's appearance based on attribute values: node shape, node color, node size, edge color, and edge line type are just a few of the appearance aspects that can be mapped to discrete or continuous attribute values. A set of such mappings of attributes to appearance is called a visual style. A canonical use of visual styles is to display expression data on a network, where nodes are colored along a red-green continuum by expression value, and node

shape is determined by a p-value cutoff. These separate appearance attributes can be combined to convey multiple channels of distinct information in one network, greatly increasing the amount of information communicated to the viewer.

The real power of Cytoscape lies in the combination of its annotation and visualization capabilities with sophisticated analysis, which is often provided by plug-ins. As usual, there is a wide range of plug-ins, providing functionality such as finding clusters (MCODE (Bader and Hogue, 2003)) and active sub-networks (jActiveModules (Ideker et al., 2002)) or performing literature searches (cPath, Agilent). The BiNGO (Maere et al., 2005) plug-in provides an excellent example of combining annotation, analysis, and visualization to produce a meaningful display.

Among the annotation types that Cytoscape can import are the Gene Ontology (GO) annotation terms. Once these annotations have been loaded, the BiNGO plug-in analyzes the annotations associated with network nodes for significantly enriched terms; the result is a new graph where the nodes are GO terms, with directed connections reflecting the hierarchy of the terms, node coloring based on the calculated degree of significance, and node size based on the number of nodes in the original graph annotated with that term.

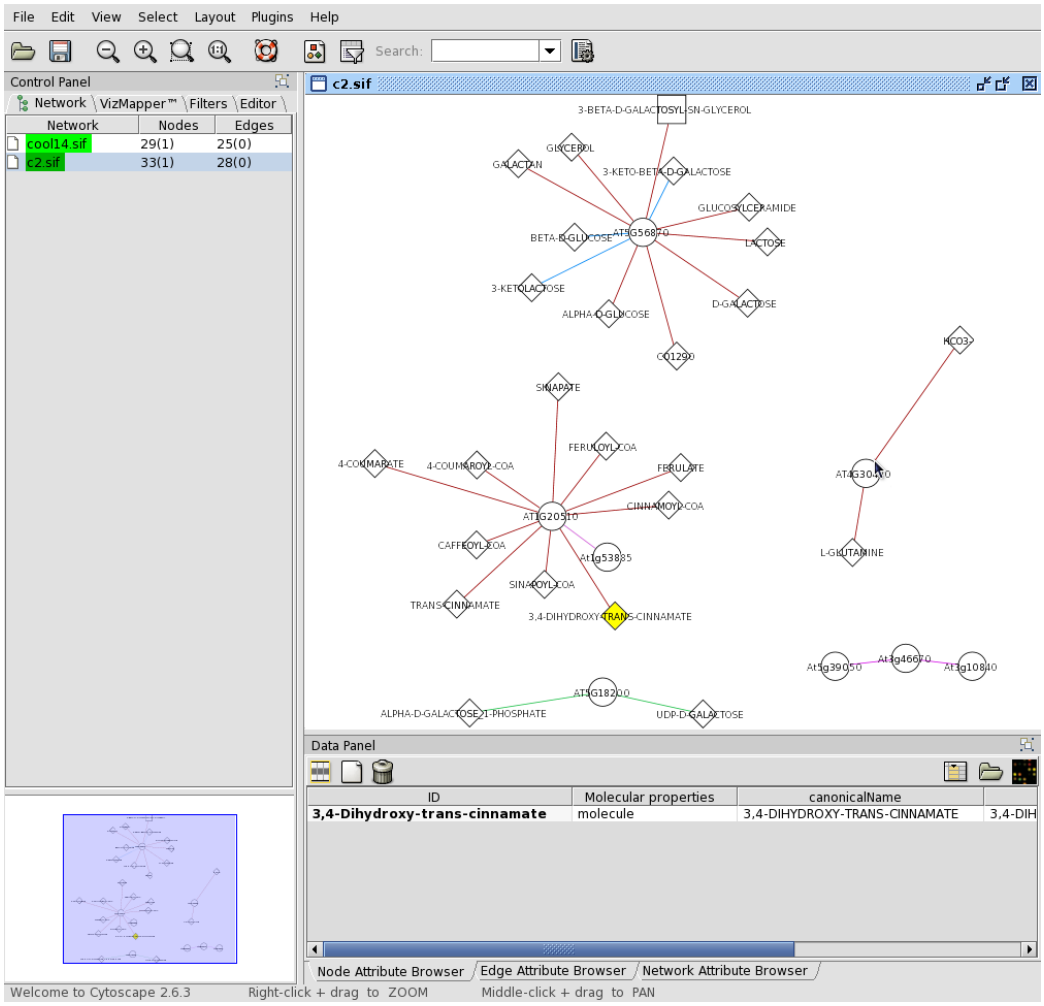

Figure 18 A Cytoscape network

Enzymatic reactions from KEGG and Aracyc for five of the fourteen genes from Figure 17. Genes are drawn as circles, and proteins as diamonds. Different interaction types are represented with different line colors. This network was generated from VirtualPlant using the "Gene Networks" analysis tool.

There are many use scenarios for Cytoscape, but one might be as follows: a set of interesting genes is identified. VirtualPlant's "gene networks" function produces a network that associates our genes with enzymatic reactions (a simple example of such a network is shown in Figure 18 (Gutiérrez et al., 2007)). We add to this network the

expression data whose analysis yielded these genes in the first place, and modify the network display to show up- and down-regulation. We may then be lucky enough to find – either visually or by using a plug-in – that several of our genes are involved in a sub-network, which suggests further experiments to characterize the relationship among those genes. Network analysis of connectivity has been used to identify putative regulatory hubs that were experimentally validated in Arabidopsis (Gutiérrez et al., 2008).

VirtualPlant

VirtualPlant is a web-based tool (currently annotated for *Arabidopsis thaliana* and *Oryza sativa*) that functions as both a data warehouse and a base of operations for a wide range of analysis tools. Items in the VirtualPlant "warehouse" include many of the experimental and annotation data sets mentioned before: Affymetrix probes for Arabidopsis Affymetrix AG and ATH1 chips, TAIR and GO annotations, and BIND protein interactions among others. VirtualPlant uses an e-commerce metaphor: users can browse or search through these data sets, and place virtual "orders" for sets of genes. These genes are then placed in the "gene cart" for further investigation and analysis (see Figure 19 for an example of VirtualPlant in use).

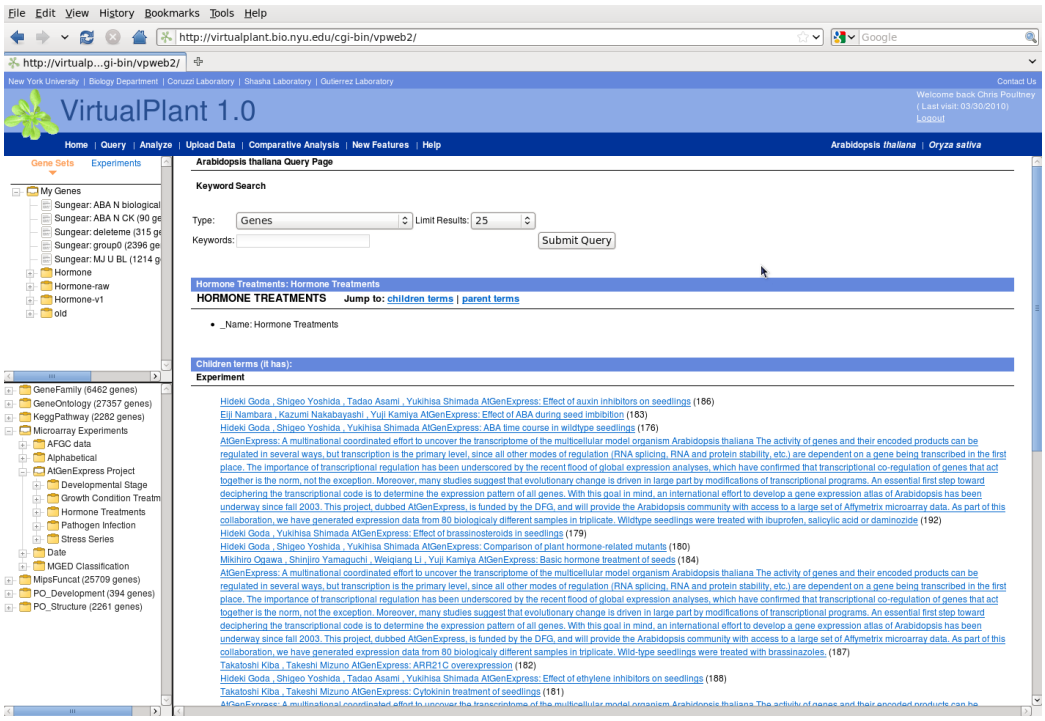

Figure 19 VirtualPlant

The main window shows the query interface and the top-level categories for browsing. The upper left window shows the gene cart, including the group "Sungear: group0" which was exported from Sungear back to VirtualPlant. The lower left window shows the browse tree, expanded to show the folder containing the AtGenExpress hormone treatment experiments. The main window shows a list of some of the hormone treatment experiments.

Genes can be placed in the gene cart in one of three major ways. The first option is to upload one or more lists of genes of interest (e.g. favorite genes studied in your own lab or genes differentially expressed in a published microarray experiment). The second option is to browse through one of the VirtualPlant annotations (e.g. GO annotations) to find a category of interest (e.g. "abscisic acid biosynthesis"), or to perform a text search within one or all of these annotations. Once the category is found, its associated genes can be added as a set of genes to the gene cart. The third

option is to browse or search the available microarray experiments, each of which contains multiple slides. The experiment is first added to the gene cart as an experiment, but is converted to a gene set using one of several methods for comparing slides and thresholding (e.g. log base 2 ratio or Rankprod (Hong et al., 2006)).

Once there are several sets of genes in the cart, VirtualPlant provides a wide range of analysis and visualization tools, including Sungear as well as on-the-fly network creation and display using Cytoscape. Most importantly, VirtualPlant not only brings together a wide range of tools, but is designed so that most of its tools can feed their output back to the gene cart, allowing for centralized, iterative analysis involving different tools, including new ones. The following extended use scenario illustrates the power of the centralized gene cart.

We start by navigating the microarray experiment hierarchy to find the AtGenExpress hormone time course treatments. From this list, we choose the following experiments: Brassinolide time course in wildtype and det2-1 mutant seedlings, GA3 time course in wildtype and ga1-5 mutant seedlings, and IAA time course in wildtype seedlings. We will refer to these as BL, GA, and IAA from here on. For each of these three, we select the experiment, and choose the “Create Experiment” option to add it to the gene cart. We select each experiment in turn from the gene cart, then select the mock treatments as the baseline and the hormone treatments as the treatment, being careful to avoid the mutant treatments in the GA and BL experiments; we then perform a log base 2 ratio test with a cutoff of 1.0. This will find genes in

each experiment that are differentially expressed at one or more time points, albeit in a less sophisticated way than (Nemhauser et al., 2006). When these three analyses are complete, we choose “Analyze” from the top menu, select our three new groups, and send them to Sungear for analysis.

Sungear now executes with our three groups as its anchors. Only one vessel corresponds to more than one condition (BL and IAA); we perform a union over it and several GO terms, then Narrow to create a new group. To send these genes back to VirtualPlant, we check the box next to the new group and click “Export”. This group will now be listed among the sets of genes in the gene cart. We could now select that group for exploration by other VirtualPlant tools (such as viewing networks constructed from that set of genes, or searching for overrepresented GO terms within that set); or save it as a file for analysis by a tool not directly callable from VirtualPlant.

Conclusions

Our experiences designing Sungear and observing other visualization tools have led us to the following core principles for designing tools for systems biologists:

1. Make the interface as transparent as possible. Good visualization software makes it straightforward for a skilled investigator to apply his or her experience to exploring the data. The data setup should be straightforward and

the first image insightful. After that, the investigator drives the exploration process.

2. Build a tool that does a small set of things well and with an intuitive interface. This reduces the length of the coding/feedback cycle, making it more likely that the tool will suit the investigators' needs.
3. Support different levels of abstraction (“conceptual zoom”) to allow users to find additional data. It’s usually impossible to show all the relevant details for a data set on-screen at one time without creating a display that’s meaningless, overwhelming, or both. Allowing the users to pan and zoom in a conceptual landscape effectively reduces the complexity of a visualization, making pertinent details available only on demand.
4. Make it easy for the tool to export and import data, so it can interact with complementary tools.
5. Make the tool as data-generic as possible. Assumptions about the types of data that your first users care about may unnecessarily limit the usefulness of the software for later users, so ensure your assumptions are minimal.

So much for the lessons for tool builders. What guidance can we give to users who must choose among tools? The obvious question to ask of any visualization is what it illustrates using its screen real estate. The less obvious question is the way in which the visualization reduces large numbers of items to a conceptually manageable small number. To appreciate this second point, consider a world without any

visualization. You have just done 10 experiments, each on 20,000 genes. Do you stare at a table of text?

Using MapMan you might place gene expression on a particular pathway. Once you choose a pathway, each gene takes a very small amount of screen real estate, so MapMan can handle an experiment with thousands of genes. Using Cytoscape, you might want to find a small set of genes according to some criterion and then display some kind of interaction data, or find sub-networks with distinguishable biological meaning within a larger set. Unless the focus is on super-node analysis, you eventually need to reduce the set of genes to a small set to render the network comprehensible. Using Genevestigator, you again choose a small set of genes to do profile analysis. Using VirtualPlant, you can use multiple tools in a centralized iterative fashion to analyze lists of genes or microarray data to construct biological hypotheses.

Any tool that displays a small number of items from a larger set requires another tool to select that small number. A relational query language can serve this purpose but it requires too much knowledge (e.g. I want those genes that show up in the intersection of these three experiments, but how do you know which three experiments?). Sungear is an alternative in that it enables selection by experiment, functional category, or commonality among experiments and/or functionalities. It does so in an interactive way and in which most of the visual field is filled with circular objects whose size connotes quantity and thus suggests the queries to ask. As the

amount of biological data explodes, the need for this type of conceptually rapid filtering only increases.

Temperature-Sensitive Mutant Prediction

Mutations, Phenotypes, and Temperature-Sensitive Mutants

A mutation in the DNA sequence of a cell's genome – whether it is a point mutation of a single nucleotide, the insertion or deletion of several nucleotides, or a larger-scale mutation such as an amplification of a genomic region – can be described in a number of ways, such as its effect on the DNA sequence, its effect on the eventual transcript resulting from that gene, its effect on the abundance of protein or transcript, or its effect on function (e.g., gain of function or lethality). In this work, we have chosen to study and make predictions based on protein sequence and structure, so it is the effect on protein sequence that we are interested in. As a result, we are by default considering mutations that change only a single amino acid in the protein product: mutations to the DNA that change an amino acid and do not truncate the protein. In terms of the effects on protein sequence, this excludes frameshift mutations, nonsense mutations, and silent mutations, leaving only missense mutations and perhaps neutral mutations.

Our work is at the protein level, so from this point on, we will use the term "mutation" to refer to a change in the amino acid sequence of a protein and "point mutation" or "substitution" to refer to a change to a single amino acid, unless the context clearly indicates otherwise. The effect of a mutation on an organism is often

deleterious and can be fatal. In other words, the phenotype – the observable effect – of the mutation is death, often in an early stage of development. This demonstrates that the protein in question has a role in early development, but the mutation that causes early death cannot be used to study the role of that protein in the organism at later stages in its development.

Certain mutations exhibit wild-type (or near-wild-type) behavior under a "permissive" set of conditions but a mutant phenotype under a different "restrictive" or "nonpermissive" set of conditions. These conditional mutations provide a way to circumvent the "early death" issue. Temperature-sensitive (ts) mutations are a particularly useful type of conditional mutation where the restrictive environment is simply at a different temperature from the permissive environment (generally the restrictive temperature is higher, making the mutations heat-sensitive, though cold-sensitive mutations also exist). Thus a researcher can grow an organism with a temperature-sensitive mutation to maturity, then "switch on" the mutation by raising the temperature to the restrictive temperature. Moreover, in general they can then "switch off" the mutation by returning to the permissive temperature.

At the restrictive temperature, a temperature-sensitive gene exhibits a drop in the level or activity of the gene product (Chakshusmathi et al., 2004), typically as a result of unfolding, destabilization of a feature critical to function, or partial unfolding that results in clearance by the proteolytic machinery. A more thorough description from (Sandberg et al., 1995) states that a ts phenotype may result from "decreases in

stability, solubility, or resistance to proteolysis, as well as reduced function... The ts phenotype may also result from failure to accumulate sufficient quantities of active protein at the nonpermissive temperature, either because of poor expression, failure of the polypeptide to fold, or aggregation." Our method is most likely to detect ts proteins that cause a change in the Rosetta (Rohl et al., 2004) energy function, e.g., those with decreased stability or difficulty folding.

Motivations

Finding temperature-sensitive mutants is traditionally a highly random process, though more targeted methods have emerged. Methods for finding ts mutations break out into three broad categories: random mutagenesis of the entire genome or part of the genome, rational methods that target a particular gene of interest, and predictive methods that suggest specific mutations for a given gene. The random methods require the most time and work, while the predictive methods require the least. Most discovery of ts mutations has been through the random methods; the rational methods are improving, but surprisingly little work has been applied to predictive methods. This gap becomes increasingly important as knowledge of the roles of individual proteins, and hence the need to study specific proteins, increases. Our goal, therefore, is to construct a method that makes highly accurate predictions to reduce work and lab time.

Prior Work

Techniques for generating temperature-sensitive mutations involve some combination of the following steps:

1. Make random or designed mutations to the genome of the target organism
2. Screen the mutants from step 1 for the desired temperature-sensitive phenotype
3. Sequence the mutants that exhibit the desired phenotype to determine the mutation(s) responsible for the phenotype

The different types of method – random, rational, and predictive – vary in the actual steps required and in the amount of time required for each step. Each method has some type of setup cost (e.g., preparing vectors and introducing mutations), and sequencing is often performed even with the non-random methods as a verification step. However, the main difference among the methods, in terms of time and effort, is in screening the mutants. The time required for screening depends mainly on the number of mutants to be screened: random methods typically generate many thousands of mutants (random methods in the papers reviewed below generate from 15,000 to 250,000 mutants), while predictive methods generate fewer than 100. Screening time also depends on the type of screen: screening yeast cultures for impaired growth is much easier than visual examination of individual fly wings.

Methods also vary in the frequency (success rate) with which they generate viable temperature-sensitive mutations and in the specificity of the mutation being

made (i.e., whole-genome or single-gene). We discuss a range of existing techniques and compare them according to the criteria above.

Random Mutagenesis

The earliest method for producing ts mutations is mutagenesis of the entire genome of an organism, using any of a number of agents or techniques: chemical mutagens such as ethyl methanesulfonate (EMS); X-rays or γ-rays; and a host of others. This is followed by a screening process to identify the temperature-sensitive mutants. Because of the large number of mutants generated, screening is generally very time-consuming for organisms on the order of complexity of *D. melanogaster*, and impossible for more complex organisms. Unless the goal was simply to create a mutant with a temperature-sensitive phenotype, the mutated genome must then be sequenced to find the mutation responsible for the ts phenotype.

A good example of the above process is provided by (Suzuki et al., 1971), where the desired outcome was a ts mutant that exhibited reversible immobilization at the restrictive temperature and was capable of producing offspring. Approximately 250,000 mutant flies were grown and screened for reversible immobilization at the restrictive temperature. Of these, only 293 exhibited immobilization at the restrictive temperature, 200 of which had died; of the remaining 93 that recovered mobility at the permissive temperature, only one was fertile and carried the temperature-sensitive paralytic mutation to which the rest of the paper is devoted.

This process is obviously hit-or-miss, and is essentially useless if one is interested in mutations of one particular gene. A more targeted method is a variant of polymerase chain reaction called PCR mutagenesis, which introduces mutations into a DNA sequence during copying. The advantage of PCR mutagenesis is the ability to restrict mutations to a particular gene. Though the genetic material to mutate must be first isolated and the mutated DNA reintroduced into the organism being studied, the overall reduction in work is clearly advantageous. Mutated genes must still be sequenced afterwards if the actual mutation is to be known. Some of the rational methods discussed below incorporate PCR mutagenesis as an intermediate step.

Rational Methods

A number of methods exist to single out a gene of interest and make a mutation to that gene that exhibits conditional behavior: heat-inducible degron fusion (Dohmen et al., 1994), conditionally splicing inteins (Zeidler et al., 2004), plasmid shuffling (Sikorski and Boeke, 1991), the “diploid shuffle” (Ben-Aroya et al., 2008), replacement of a gene's promoter with a different rapidly-repressible promoter (Mnaimneh et al., 2004), or techniques for destabilizing the target mRNA (Schuldiner et al., 2005). These methods are essentially lab procedures for generating conditional mutations that either a) do not use any prior knowledge about which specific mutations are likely to exhibit conditional behavior, or b) exploit mechanisms other than straightforward point mutations. We discuss several of these methods below.

N-degron fusion

Heat-inducible degron fusion (N-degron fusion) (Dohmen et al., 1994) can be used to make a protein temperature-sensitive by expressing the protein as a fusion with a temperature-sensitive N-degron. An N-degron is an intracellular degradation signal whose activity is determined by the N-end pathway, which regulates proteolysis via ubiquitination. The technique is based on the creation of a ts N-degron, Arg-DHFR^{ts}, that is stable at the permissive temperature but becomes unstable at the restrictive temperature, activating the N-degron and triggering rapid proteolysis. This ts N-degron can be expressed as a fusion with a protein of interest, allowing any protein that can tolerate N-terminal extension to be made temperature-sensitive. Assuming the N-terminal extension is tolerated, this alleviates the issue of "leaky" ts mutants that, while showing a ts phenotype, also have a weak phenotype at the permissive temperature. In this initial paper, the method was limited to intracellular areas where the N-rule pathway operates, namely cytosol and the nucleus.

N-degron fusion has some interesting features. It relies on a clever trick: the plasmid used includes a ubiquitin moiety immediately before the N-terminal Arg. The ubiquitin allows almost any amino acid (Arg, in this case) to be placed at the N-terminus, but is cleaved almost concurrently with translation. N-degron fusion mutants can be used in ways other than the usual testing at permissive and restrictive temperatures. One alternative is to test the mutated protein at the restrictive temperature in cells of two different strains of the target organism, one of which

carries a mutation that disables the N-end pathway. Another is to test the mutated protein at the restrictive temperature, but also use varying concentrations of an Arg-DHFR^{ts} inhibitor (methotrexate), essentially creating a mutant that is conditional but not temperature-sensitive.

Ts Intein Splicing

Another process for creating temperature-sensitive mutations relies on a mechanism similar to the splicing of intron regions of DNA: the post-translational splicing of a part of a protein called an "intein". A temperature-sensitive intein will be spliced at the permissive temperature, but not at the restrictive temperature. If the unspliced intein results in an inactive protein, temperature-sensitive splicing results in a temperature-sensitive protein.

Ts intein splicing (Zeidler et al., 2004) starts with a well-know yeast intein, Sce VMA, that is inserted into Gal4 to create Gal4INT^{WT}. To check its viability, they introduced Gal4INT^{WT} into a yeast strain (FY760) that is dependent on rescue by Gal4-expressing plasmid. They also verified that a mutant non-splicing VMA (inserted into Gal4 to produce Gal4INT^{Dead}) does not rescue the organism. They then performed low-fidelity PCR on Gal4INT^{WT} and screened for a ts phenotype, yielding several ts variations that can collectively be referred to as Gal4INT^{TS}. When the ts variants were transformed into the rescue yeast strain, the result was a ts-lethal mutant, verifying that a ts intein had been created. In subsequent experiments, the ts inteins isolated above

were transferred into yeast Gal80 and two Dmel genes, demonstrating that the intein can work in different genes and species.

There are several caveats to this method. Some leakiness is possible; wise choice of the intein insertion site may mitigate this. In addition, splicing is context-dependent. In their first paper, the source intein was immediately upstream of a Cys and only worked when inserted upstream of a Cys in the target protein; and not all tested Cys-upstream insertion sites in the target protein worked well.

The "Diploid Shuffle"

A third and final example of rational ts creation is the "diploid shuffle" technique (Ben-Aroya et al., 2008). This technique was motivated by the desire to find a systematic way to study yeast essential genes, which cannot be studied using the gene deletion projects yeast knockouts (YKO) as cells will not survive without essential genes. Conditional mutants provide an excellent way to study these essential genes. The method involves the following steps:

1) Start with DNA containing the gene of interest (loosely referred to as YFEG, "Your Favorite Essential Gene")
2) Perform PCR mutagenesis on YFEG (in this paper, ~15,000 mutations were generated for each gene)
3) Clone mutants into the plasmid SB221+Topo-TA (which contains URA3 flanked by KanMX 3' and 5' regions) to create a library of mutagenized YFEG

4) Transform this library into *E. coli*, then digest to yield isolated mutant strands
5) Transform the mutant strands into a "haploid-convertible heterozygous" diploid strain, which has a bar-code-flanked knockout of YFEG
6) Sporulate and select for haploids at 25° (a standard technique that selects for cells expressing Ura+ and hence the mutation as well)
7) Select temperature-sensitive candidates, then replica plate and incubate at 25° and 37° and screen for ts phenotype

The result of the above process was a MATa strain from the YKO background with the gene at its endogenous location, flanked by bar codes. This method generalizes to any yeast gene, and is made significantly easier by the existence of YKO and the ease of transformation with SB221+Topo-TA. Its main drawbacks are the time required to screen for ts mutations after PCR mutagenesis and the fact that the method is only applicable to yeast. On the other hand, the method yielded real ts mutants (as opposed to, e.g., tet-regulatable promoters, which are binary on/off), allowing the study of ts alleles at intermediate expression levels.

Predictive Methods

As discussed in Prior Work, each of the methods listed above has a cost and success rate associated with it. Any method that depends on random mutagenesis, either of the whole genome or of a gene, will require the screening of many mutants for the desired phenotype. These methods also tend to have a low success rate,

generating many mutations in order to isolate a handful of ts mutations, though PCR mutagenesis at least allows the targeting of a specific gene. The diploid shuffle method has a very high success rate, but it still depends on PCR mutagenesis – perhaps even multiple rounds of mutagenesis – and therefore has a high screening cost. The other rational methods discussed do not carry the high screening cost – for example, the intein splicing method used PCR mutagenesis for the initial step of finding the ts inteins, but does not require mutagenesis to use the inteins – but these methods are inherently limited in other ways. N-degron fusion and ts intein splicing generate at most several mutations, and a few as one mutation, for any gene; if none of this small number of mutations shows a ts phenotype, then the technique fails for that gene. For example, the N-degron method achieved a success rate of 60% on a sample of yeast essential genes (Kanemaki et al., 2003); on another gene set, the tet-regulatable promoter method resulted in a ts phenotype in about 50% of the genes (Mnaimneh et al., 2004); and on yeast essential genes, DAmP achieved a ts phenotype rate of just over 30% (Ben-Aroya et al., 2008).

The ideal technique for generating temperature-sensitive mutations would yield the maximum number of ts mutations, ideally to a specific gene of interest, with the minimum amount of screening. Predictive methods aim to do just this: they focus on a gene of interest and generate a small list of high-likelihood predictions for that gene, thereby eliminating the large-scale screening processes associated with the random methods. The success rate is high, and depends on the accuracy of the

predictions: the success rate is the fraction of genes for which at least one viable ts mutation is found among the set of predicted mutations. An important bonus for the predictive methods is that the end result is still a traditional temperature-sensitive point mutation, which can be studied *in vivo* without otherwise perturbing the protein, its regulatory pathways, or its environment.

Burial, Mutations, and Stability

(Sandberg et al., 1995) describe relatively early work that lays an encouraging foundation for the prediction of ts mutations. Their goal was familiar – to eliminate the time-consuming purification and screening of candidate proteins – but they posed the question differently: can *in vivo* activity act as a reporter for attributes that can be measured *in vitro*? In other words, can *in vivo* activity predict *in vitro* measurements of stability and DNA binding affinity? Mutants of bacteriophage f1 Gene V protein (an 87-residue homodimer that binds to DNA and RNA and is essential for phage growth) were assayed for phage growth at permissive and restrictive temperatures *in vivo*. Those results were compared to *in vitro* measurements of stability and DNA binding affinity. The results for surface (>10% side chain solvent accessibility) and buried substitutions were analyzed separately, with the following results: while no correlation could be found between *in vivo* activity and *in vitro* measurements for surface substitutions, there was a strong correlation between *in vivo* activity and decreased stability ($\Delta\Delta G°$) for buried substitutions.

Furthermore, in this study, 85% of studied mutations at buried hydrophobic positions were temperature-sensitive (though this figure does not seem to be supported by later papers). A detailed study was made of all possible combinations of apolar substitutions at two sites known to be tolerant to substitutions (i.e., mutations are not inactive, but may be ts); 29/64 mutations were inactive, while 28 were ts. In all cases but one, the double mutant activity was the same as or less than that of either single mutation.

Across the 3 proteins discussed – Gene V, T4 lysozyme and λ-repressor – proteins with buried substitutions were more unstable as a group. Buried substitutions may so frequently be ts because they alter interior hydrophobicity and/or packing; also, apolar buried substitutions can cause subtle rearrangements that propagate to the surface and perhaps even affect binding. As a general rule (from λ-repressor N-terminal domain studies), interior substitution(s) were tolerated if 1) side chains were generally non-polar and 2) total hydrophobic core volume did not change significantly from WT. Surface substitutions did not destabilize strongly, except for hydrophobic substitution at an exposed site. In fact, active site mutations often increased stability; however, this is only an issue for the (generally small) fraction of active site positions – roughly 13/87 in the case of Gene V.

Prediction of Burial and Temperature Sensitivity from Sequence

The correlation between mutations at buried sites and a temperature-sensitive phenotype provides an excellent starting point for ts mutant prediction. In the first of several papers to address ts mutant prediction, (Varadarajan et al., 1996) propose a method for predicting ts mutants that hinges on accurate prediction of burial from sequence, based on the average hydrophobicity and "hydrophobic moment" of the candidate highly buried sites. For each predicted buried site, five mutations are suggested based on the native residue to give a range of ΔG values at that position such that at least one mutation should result in a ts mutant.

The basic premise, as explained earlier, is that substitutions at buried residues will destabilize the protein; therefore prediction of buried sites is the first step in the method. Only the residues designated as the most highly buried – Cys, Phe, Ile, Val, Tyr, Met, and Leu – were considered as candidate sites. Each residue has a 0-100 hydrophobicity H(n) associated with it, which was used to calculate two quantities: the average hydrophobicity H_{av} and the hydrophobic moment H_{mom}. H_{av} is the average of H(n) over a 7-residue window, and H_{mom} is the second moment of H(n) over a 9-residue window with a secondary-structure dependent phase term optimized to find amphiphilic regions. Rules for burial prediction were derived using a training set of known PDB structures. Using H_{av}, H_{mom}, and H(n) at a given site, and H_{av} or hydrophobicity of neighboring sites, sites that are less than 15% accessible were predicted with 80% accuracy on a separate test set.

For each putative highly-buried site, a range of substitutions was suggested based on the native residue at that site. These substitutions were based on expected ΔG values; actual ΔG values of the mutants will vary and are dependent on ΔH, ΔC_p, and native structure ΔG,. The goal is to destabilize the protein to some intermediate degree, based on the fact that the free energy of unfolding at room temperature is 5-15 kcal/mol. Substitutions were split into "conservative" (e.g. methylene group addition/deletion) and "non-conservative" (e.g. buried hydrophobic => charged/polar/glycine). Substitution counts were minimized by keeping charged and polar substitutions roughly the same size as the native and having them add a single negative charge and hydroxyl group, respectively.

For the 3 proteins tested, 16 buried sites were predicted; 14 of the sites were actually buried in the structure. Five of the six known ts mutations at those sites were among their (presumably 80) suggested substitutions. Many of the other predictions at these sites appear not to have been studied as of the writing of this paper.

More recent work by the same group (Chakshusmathi et al., 2004) used the same prediction method, with a few small changes, to make and test mutations to *E. coli* and yeast. When the method was applied to the *E. coli* cytotoxin CcdB, mutations at four predicted buried sites yielded 6 ts mutations. In yeast, predictions made at 3 sites in a shortened version of Gal4 called miniGal4 yielded 8 ts mutations, 4 of which were active in the full Gal4. The mutations were the first reported ts mutations for

CcdB and Gal4. The method is as follows, with measures becoming increasingly difficult in steps 2-4:

1. Identify putative buried sites
2. Introduce Asp independently at 2 predicted burial sites; if this makes the protein inactive, try the other 18 substitutions at those sites
3. Introduce positively charged, polar, small & large hydrophobic residues at 4 sites (Lys, Ser, Ala, Trp)
4. Express WT or mutant in an inducible system, and check for intermediate ranges in cases where extremes yield active and inactive phenotypes

In *E. coli* CcdB, ten buried sites were predicted, and tested by substitution with Asp. Eight sites showed inactivation; they corresponded exactly with the sites that were found to be correctly predicted as >90% buried in the CcdB crystal structure, which was solved independently during the research (Loris et al., 1999). Of these 8 sites, four were chosen for further experimentation. Since they all showed inactivation by Asp, the other 18 amino acids were substituted as per step (2) in the procedure above, yielding 6 ts mutations (along with 28 active/neutral mutations and 42 inactive/lof mutations). CcdB was also expressed in an inducible system under the control of arabinose; while useful, the manipulation of expression level outside of the usual regulation pathway made this data less useful. In all cases, the permissive and restrictive temperatures were 37° and 42°. This is quite high, which raises the

possibility – which is discussed by the authors – that increased proteolysis due to heat shock response played a role in ts behavior.

In yeast, Gal4 was chosen as the protein of interest. To simplify testing, a shorter version (~200 residues) of Gal4 called miniGal4 was made that included the DNA-binding domain, a 7-residue linker, and the activation region. Only five buried sites were predicted; of these, three were chosen and mutated to Ala, Ser, Trp, and Lys, as per step (3) in the method. Ts screening was performed at 21°, 30°, and 37°. Only one ts mutant resulted from the limited substitution, though a total of 8 were found by substituting all possible amino acids. Of these eight mutations, four were found to be ts in full-length Gal4.

As with (Sandberg et al., 1995), a few general rules emerged: Asp and Glu mutations were always inactive; Lys and Arg were better tolerated, perhaps because their long, flexible side chains can reach the surface; substitution of a buried hydrophobic by a smaller hydrophobic was generally tolerated; and bulky aromatics, especially Trp, were more likely to inactivate. But no clearer correlation between substitutions and phenotypes emerged.

One key yeast experiment showed the advantage of predictive methods over random methods. Random mutagenesis of miniGal4 was carried out using error-prone PCR; of the more than 20,000 transformants generated, **none** were found to be more than very mildly ts (several active and inactive transformants were identified as well). It is highly significant that 20,000 mutations to a ~200 residue protein produced

essentially no ts mutations, while a predictive process that made 57 substitutions at 3 sites in that same protein produced 8 ts mutants.

Methods

Goals

As stated above, our goal is to take the structure of a protein of interest and generate a ranked list of candidate amino acid substitutions that may yield a mutated protein with a temperature-sensitive phenotype. In practice, this would give, for example, a "Top 5" list of mutations taken from the 5 highest-ranking predictions. Following the approach of Varadarajan, rather than trying to predict exhaustively all possible ts mutations to a protein, we instead focused on making a small number of highly accurate predictions.

Procedure

The outline of our prediction procedure is as follows:

1) Start with the known structure, or a high-quality homology model, of a protein of interest.
2) Find the buried sites in the protein, and model mutations to all possible amino acids at those sites.
3) For each model generated in step 2 (i.e., for each mutation at each buried site), apply an algorithm that allows the protein to relax to a lower-energy state to

accommodate the mutation. Since there is randomness inherent to the algorithm, run it multiple times for each model, producing an ensemble of relaxed structures for each mutation. Generate an ensemble for the starting "wild-type" (wt) structure as well.

4) Generate a score file with one line per mutation ensemble by comparing the relaxation algorithm's score terms for the mutation ensemble to those for the wild-type structure ensemble. Add several other relevant features, such as ACC and conservation of the native amino acid.
5) Using the score file from step 4, either:
    a. Train one or more machine learning algorithms to discern ts mutations from non-ts mutations.
    b. Using the machine learning algorithm(s) trained in step 5a, predict likely mutations to a new protein. These predictions have a confidence or estimated probability associated with them, from which we compile our "Top 5" list.

Training Set

Per step 5 above, our method clearly relies on training machine learning algorithms using a reliable set of known ts and non-ts samples. We could easily have added "Step 0: gather training samples" at the beginning of the above outline. Our training samples were taken from a several sources, as enumerated below, comprising

ts and non-ts alleles from worm (*C. elegans*, Cele), yeast (*S. cerevisiae*, Scer), and fly (*D. melanogaster*, Dmel). The database scraping and literature searching were performed by Kris Gunsalus and Michelle Gutwein.

- Worm mutants were culled from WormBase (Harris et al., 2001) (http://www.wormbase.org/) v200 and WorTS. We filtered the lists of putative ts and non-ts samples to achieve the highest possible confidence by removing the following: putative nonsense mutations, unsequenced or uncurated mutations, frameshifts, deletions and insertions, and mutations affecting splice sites. In addition, we applied the following specific rules:
    - WormBase ts samples: Only a handful of alleles are directly annotated as ts; most are annotated as ts for a particular phenotype. In cases where an allele had multiple ts phenotypes, only the first phenotype was used. After applying all filters, 107 alleles remained.
    - WormBase non-ts samples: "Not ts" is not an explicit feature tracked in WormBase; therefore, we had to derive heuristic rules for conservative designation of non-ts alleles. We started with a list of alleles that are not annotated as ts, but are mentioned in papers that describe other ts alleles, assuming that these were more likely to have been screened for a temperature-sensitive phenotype. This process yielded a final list of 249 alleles.

- WorTS ts samples: though there were about 4000 ts alleles in the database, only 140 had molecular data. After applying all filters, 61 alleles remained, 55 of which were already included in the WormBase data.

- Yeast mutants were derived from the Saccharomyces Genome Database, Textpresso (Muller et al., 2004) searches, and the Histone Systematic Mutation Database (HistoneHits) (Dai et al., 2008; Huang et al., 2009).
    - With the help of SGD curators, a list of 86 temperature-sensitive mutations was derived, of which 53 met the screening criteria.
    - Textpresso search (keywords "temperature sensitive" (with quotes), category "allele") was used to find papers describing temperature-sensitive mutations. Mutations described in the papers as ts were included as ts samples, and mutations from those same papers that were not described as ts were included as non-ts samples. This yielded 177 ts mutations and 229 non-ts mutations.
    - We gathered data for proteins H3 and H4 from the HistoneHits web site, using the growth rate TS-39 assay (temperature-sensitive phenotype at 39°). We then eliminated all mutations to the flexible tail regions and all deletions. We considered all mutations scored 0 (same as wild type) as non-ts and all mutations scored -2 (severe loss of

function) as ts. We ignored mutations scored -1 (mild loss of function). This yielded 25 ts and 225 non-ts samples.

- Fly mutants were extracted from FlyBase (Tweedie et al., 2009).
    - Temperature-sensitive alleles were identified using a multi-step process. First, we downloaded from FlyBase a list of alleles that: 1) were annotated heat-sensitive using their controlled vocabulary; 2) had molecular data; and 3) had molecular data on the nature of the lesion in the allele. We then trimmed this list to exclude insertions, deletions, mutations explicitly induced using a heat shock promoter, and mutations with unclear annotations of the resulting amino acid substitution. This left a list of 204 ts samples.
    - There are no non-ts alleles for fly in the data set at this point. We attempted a procedure conceptually similar to the one we used for worm and yeast, starting with alleles that are mentioned in the same publications as ts-annotated alleles but that are not annotated as ts, and taking the further step of restricting these to genes for which we already had models. However, this caused a drastic decline in discrimination between ts and non-ts samples in fly. Our hypothesis is that this method is too coarse-grained; since a lack of ts annotation does not mean that an allele is not ts, the only way to ensure that putative non-ts alleles are

actually not ts is to read the papers in which they appear. We have not taken the time to implement this step yet.

Overall, this gave the sample counts shown in Table 2. However, these numbers were considerably reduced by the availability of homology models, as will be seen in the next section.

Species	Ts samples	Non-ts samples
Worm	113	249
Yeast	255	454
Fly	204	0

Table 2 Total number of samples provided by database and literature searches.

Starting structures

Our method requires a protein structure as a starting point for the prediction process. The user must provide this structure; the source (known structure from the PDB (Bernstein et al., 1977), homology model) does not matter to us. Our training set, however, consisted almost entirely of proteins of unknown structure; these proteins needed to be modeled to be incorporated into the training set.

A high-quality homology model is often based on a template generated from a multiple alignment of known structure that is then fed to a program like MODELLER (Eswar et al., 2006). While this process produces the best models, it requires significant user input. Given the number of models needed to build our training set,

and the time available for modeling, we decided to use a less rigorous but far less time-consuming protocol:

- Run each sequence through ModWeb (Eswar et al., 2003), a web front-end to MODELLER, using the "Slow" setting for search speed (sequence-sequence, profile-sequence, and sequence-profile methods) to give high-quality models.
- Keep any homology model with an identity of 70% or greater to some domain of the target protein (this sometimes resulted in multiple models for different domains of a single protein).

In this way, we were able to generate sufficient number and quality of models to provide training samples for our learning algorithm without shifting the focus of the project entirely to the generation of homology models. However, many alleles did not have models of sufficient quality, or had mutations that fell outside of the protein regions with high-identity models. After homology modeling, the set of training samples was as shown in Table 3.

Species	ts samples	Non-ts samples
Worm	8	57
Yeast	156	299
Fly	43	0

Table 3 Number of samples with viable homology models.

Mutant Generation

Our method takes advantage of the same fact exploited in the Varadarajan et al papers, namely that mutations at buried positions are more likely to destabilize a protein. In addition, limiting our method to buried positions reduces the complexity of our model, as we do not have to account for the many surface effects that can affect temperature-sensitive behavior (e.g. participation in protein-protein interactions, which requires identification of interface residues which will vary per interaction). Therefore, the first step in generating mutations was to identify the buried positions in the protein. Since we had actual structures (albeit often via homology modeling), we could accurately calculate burial using standard methods. We chose to use Probe (Word et al., 1999) to calculate the percent solvent accessibility (ACC) for each residue, including backbone and side chain atoms. We considered residues with ACC of 10% or less to be buried, and made mutations only at those sites. The ACC cutoff further reduced the training set to its final size, shown in Table 4. Figure 20 shows a comparison of sample counts through the set building process, and Figure 21 gives more detail on the final training set.

Species	Ts samples	Non-ts samples
Worm	5	38
Yeast	48	92
Fly	22	0

Table 4 Number of samples meeting all criteria for training set.

Figure 20 Comparison of training sample counts

Training sample counts for each stage of the sample collection process: initial counts from literature sample, samples with viable homology models, and samples meeting the burial cutoff (final training set).

Once the buried sites were determined, we used PyMOL (DeLano, 2002) to generate mutations at those sites: starting with the "wild-type" structure, a Python script generated a PDB for each of the 19 possible substitutions at each buried site for a given protein.

Model Relaxation

For each mutation made, we needed to see if the protein could accommodate the change – it may tolerate the change, "blow up", or be destabilized to some intermediate degree. Prediction of protein structure after mutation, even in the case of making a single mutation to a starting structure, is a complicated process: each amino acid has at least two effective rotational degrees of freedom, and the angles at many of these may change as a result of a mutation. This quickly adds up to hundreds or thousands of degrees of freedom even for simple proteins. The protein modeling software Rosetta allowed us to simulate this complex process via its "relax" protocol.

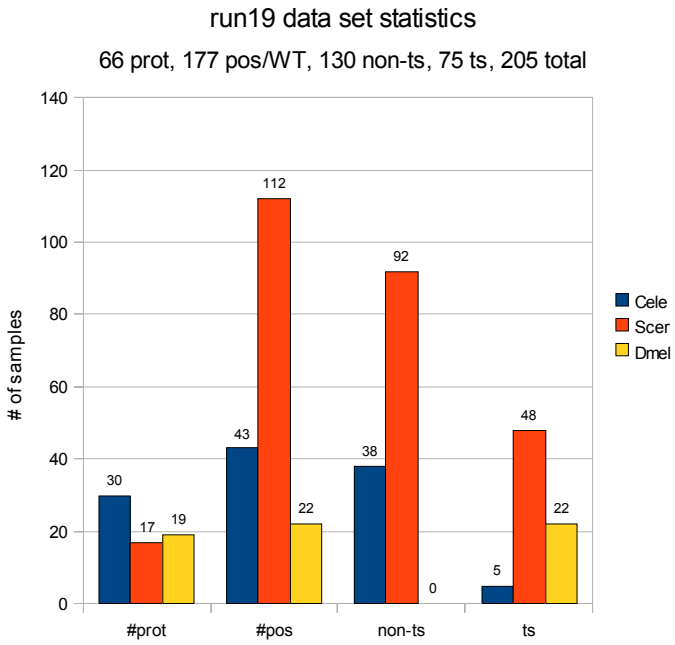

Figure 21 Statistics of the final training set
Counts are given for the total number of proteins (#prot), positions (#pos), and non-ts and ts samples.

Rosetta is a comprehensive tool for protein modeling: its major functions are *de novo* prediction of protein folding, prediction of protein structure, protein design, and protein-protein docking. These and other functions are implemented as different protocols, such as structure prediction, loop modeling, clustering, scoring, and

relaxation. Rosetta comes in two versions: the original codebase, Rosetta 2.x, and the newer version, Rosetta 3.x, often referred to as “mini”. We used Rosetta mini, which has been completely redesigned to allow for modular creation of new protocols and easy modification of existing ones.

At the core of Rosetta is its scoring function, which is used during program execution to guide the Monte Carlo process used in relaxation and is critical to us for evaluation of results. The scoring function has low-resolution and high-resolution modes, roughly corresponding to residue- and atom-level scoring, which allows a trade-off between speed and accuracy. For example, the low-resolution function is generally used in the first stage of docking, during traversal of the energy landscape of possible positions and orientations of the mobile protein; once a favorable position has been found, the high-resolution function is used to make more accurate local moves. Each function is broken down into its own component terms. For example, the low-resolution function includes a component called “env” that approximates residue solvation based on the residue type and number of neighboring residues. The high-resolution function uses the component “solv”, which calculates solvation at atomic resolution (Lazaridis and Karplus, 1999).

Simple substitution of one residue for another in the protein model may introduce steric clashes, where several atoms are too close and repel each other strongly, or other effects that are unfavorable to the protein’s overall energy. The relax protocol allows “accommodation” of a mutation by searching the space of rotational

degrees of freedom to find a lower-energy configuration – for example, allowing the atoms in the vicinity of a steric clash to move sufficiently to alleviate the clash. Some mutations are accommodated with little or no change to the structure; others cause significant structural changes or a sizable increase in energy. In this way, the relax process gives us a means of quantifying the effect on structure of a given mutation.

For each mutation generated in the "generation" step (i.e., for each of the 19 possible mutations at each buried position), we performed a set of relaxation runs. We also performed a set of runs on the wild-type structure to generate a "wt" ensemble. The result was an ensemble of runs for each mutation and for the wild-type; each individual run produced a final structure and, more importantly for us, a score which is broken into its component parts. Final score terms were generated by running the scoring protocol on the final model, then merging in two terms that were generated by the relax protocol but not by the scoring protocol. After some manipulation described in the next section, these component score terms became the basis of the feature vector for training our machine learning algorithm.

We perform relax runs on the wild-type structure mostly because of the natural variability of protein structure, stability, and energy. Different proteins will have different starting energies when evaluated by Rosetta, as proteins vary tremendously in structure and function. Furthermore, inherent properties of the proteins, such as the change in energy (ΔG) at which unfolding occurs, vary across proteins. In order to make valid comparisons between wt and mutant ensembles, we need to know how

much of the change introduced by the relax runs is due to these varying inherent properties of the protein and how much is due to the mutation itself. The relax run ensemble derived from the wild-type structure helps us differentiate between these.

In a typical prediction run, we performed the relaxation process 50 times for each input structure, wt or mutant). This is a time-consuming process, and most of the CPU cycles consumed in the prediction process were spent running relaxation simulations. It also generated a large amount of information in the form of PDB files for each structure: for a protein with 100 buried sites, there would be 100 sites × 19 mutations × 50 runs = 95000 files, which take up several GB even when compressed. However, the files themselves were not actually needed after re-scoring of the structures and were only kept in case they were needed later to verify earlier work.

Score File Generation

Structure-Based Terms

Once the relaxation runs were completed, we analyzed each ensemble, comparing it to the ensemble generated by the wt runs. For a given mutation ensemble, we used the quartile method described below to compare the distribution of each score term across that ensemble to the distribution of the same score term for the wt ensemble. The result was one set of Rosetta-derived features for each mutation ensemble. To this we added the ACC of the native amino acid at the current position. We also added secondary structure membership terms, derived from PyMOL,

depending on whether the native amino acid at a position participated in an α-helix, a β-sheet, or a loop region. We refer to these terms, which depend on having the protein structure, as structure-based terms, to differentiate them from the sequence-based terms described later that require only the protein sequence.

Quartile-Based Analysis of Scores

Each run ensemble generated a distribution over each score term, and each mutant distribution must be compared to the wt distribution for that protein. The main difficulty in analyzing our run ensembles was that the analysis required the comparison of distributions of Rosetta score terms between the two ensembles. This was challenging because of the varying inherent properties of the different proteins, as discussed in Model Relaxation, and how those variations were reflected in the unnormalized Rosetta scores.

We tried several methods to analyze the ensemble scores – including a Z-score based approach that normalized across all mutations at a given position and a median/variance method that directly compared the median and variance of the mutation and wt ensembles – before settling on a quartile-based approach (Figure 22). The key attribute of the quartile-based approach was that it allowed comparisons without making assumptions about the underlying distribution of the data.

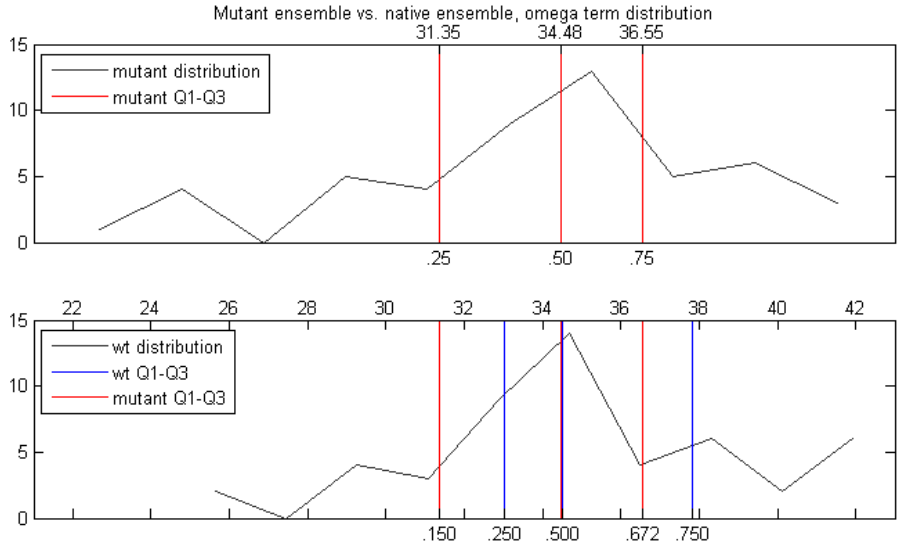

Figure 22 Calculation of score terms using the quartile method

Mutant ensemble quartiles 1-3 were calculated for the mutant ensemble distribution (top). Q1-Q3 are indicated by red lines, with the corresponding values above and percentiles below. The mutant Q1-Q3 values were then mapped to locations in the wild type (wt) ensemble distribution (bottom). Q1-Q3 of the mutant distribution are again indicated by red lines, with their percentiles relative to the wt distribution shown below. Wild type ensemble Q1-Q3 are shown in blue for reference.

For a given ensemble E and set of values S for a single score component of E, we found the first, second, and third quartiles (Q1, Q2, and Q3) of S. We then found the percentiles of those Q1, Q2, and Q3 values within the set of values S' of the same score term in the wt ensemble. The percentiles were represented as fractions between 0 and 1; values in S lying outside of the range of S' were clamped at 0 or 1, as appropriate. Therefore each score component of the mutant ensemble generated three values in the score file, ranged [0,1], which represented the locations of Q1-Q3 of the mutant score term distribution within that term's distribution in the wt ensemble.

As an example, the analysis of omega score terms shown in Figure 22 proceeds as follows. The omega score values from the mutant ensemble are analyzed, and found to have (Q1,Q2,Q3) = (31.35,34.48,36.55). These three values are then located within the wt ensemble, and each is given a number corresponding to its percentile with respect to the omega term distribution in the wt ensemble. The result is (0.150,0.499,0.672). The collection of omega scores from the individual runs in the ensemble is now represented by these three percentile values.

Sequence-Based Terms

In addition to the structure-based terms produced by the quartile method, we derived a number of purely sequence-based terms. First, we categorized both the native and the mutant amino acids according to their attributes. One group had four categories: small hydrophobic, large hydrophobic, polar, and charged; the other group was slightly more fine-grained, with seven categories. We then added fields to the output, one for the 4-category group and one for the 7-category group, as follows: 0=same amino acid, 1=same category, 2=different category.

In order to leverage the large proportion of potential training samples (1070/1275) that had sequence data but no structure data, we constructed a log odds table for changes from one class to another (e.g., large hydrophobic to polar). Using all 898 single-mutation samples, we tallied the number of times each class change resulted in a ts mutant and the number of times that change resulted in a non-ts mutant,

starting with a pseudo-count of two. We then calculated $log_2 \frac{\#\ ts\ changes}{\#\ non{-}ts\ changes}$ for each class change. The log odds table for the 4-group classification is shown in Table 5.

		Mutant			
		Small Hydrophobic	Large Hydrophobic	Polar	Charged
Wild Type	Small Hydrophobic	-2	-2	-1	0
	Large Hydrophobic	-2	0	0	0
	Polar	-2	-1	0	-2
	Charged	-2	0	-1	-1

Table 5 Log odds ts/non-ts by category

A third set of sequence-based terms was derived using BLAST (Altschul et al., 1990) and PSI-BLAST (Altschul et al., 1997). For each protein in the training set, one iteration of BLAST (Basic Local Alignment Search Tool) was run on the NCBI non-redundant protein sequences (nr) database (Pruitt et al., 2007) using the native protein sequence. Next, one iteration of PSI-BLAST (Position-Specific Iterative BLAST) was performed starting with the previous BLAST results to generate the position-specific scoring matrix (PSSM) for that protein. Runs were performed with e-value=1, inclusion_ethresh=.001, comp_based_stats=1, num_descriptions=3000, num_alignments=300, and pseudocount=2. The high e-value is offset by

inclusion_ethresh, which actually determines inclusion in the alignment used for the PSSM.

The PSSM gives the log odds of an amino acid (including the native amino acid) occurring at a specific position in the input sequence considering the distribution at that position. This PSSM score indicates both the degree of conservation of the native amino acid at that position and the likelihood of a given mutation at that position. The PSI-BLAST run also gives a number of other statistics including frequency of occurrence of each amino acid at each position and overall information content per position. For a given mutation, we derived seven terms from the PSI-BLAST run: the log odds of the native and mutant residues, the frequencies of the native and mutant residues, the difference in log odds and frequency between the native and mutant residues, and the information content at that position.

Final Score File Assembly

The final score file includes all structure- and sequence-based terms, along with several text fields describing the source protein, position, and mutation for later reference. Score files are generated by a pair of programs, one in AWK and one in Java, that read the score and PSSM files for each ensemble, perform the processing described above, and create two similar output files: one in file CSV format for quick examination in spreadsheet programs or other processing and one file in the ARFF input file format used for machine learning. The full score file contained 113 columns:

93 from the quartile analysis of the 31 components of the score function, 7 from the PSSM, 2 amino acid category change columns, 2 amino acid change log odds columns, one ACC column, 3 secondary structure columns, 4 text description columns, and one column giving the class label. Some of these columns were found to be redundant and eliminated; see Choosing and Refining Classifiers in the Results section for details.

Machine Learning: Training

After analyzing the score files according to the above steps, we had an input file with one line for each mutation ensemble. Each line consisted of multiple fields, as given by the column descriptions in the previous section. This constituted the feature vector for training the machine learning algorithms.

We used the Weka (Hall et al., 2009) suite of machine learning tools for all of our machine learning. We initially used four different classifiers, as implemented in the Weka package: two decision tree variants and two SVM variants. After the first few rounds of testing and evaluation, we eliminated the decision tree algorithms (see Results).

The decision tree algorithms, which we refer to as “C45” and “C45R”, are both based on the C4.5 algorithm (Quinlan, 1993). A trained decision tree operates by sending samples down the tree. At each interior node starting from the root, a data attribute is queried, and depending on the value, one of the branches of the tree is

followed. This process continues until a leaf node is reached, from which the sample label is determined. Classification probability is estimated from the training sample distribution at that node. The C4.5 algorithm is a particular type of decision tree that uses a gain ratio impurity measure to decide on the attribute and (for continuous variables) threshold to choose for each node in the decision tree. C4.5 builds a decision tree that accommodates both discrete and continuous attributes and supports multi-way splits (decision nodes with more than two children) – hence the gain ratio impurity measure, which normalizes to counter the bias towards larger splits. As with many decision tree algorithms, it builds a large (over-fitted) tree, then prunes the tree by removing decision nodes whose elimination causes only a tolerably small increase in impurity.

The algorithm which we refer to as "C45" is exactly as described above. It does pruning based on the statistical significance of the split at each node compared to a random split at that node – nodes whose splits are above a confidence threshold (default 0.25) are eliminated, i.e., the decision node and its children are collapsed into a leaf node. The pruning process can also replace a decision node with one of its children (a technique called "subtree raising"). The "C45R" method differs in that the default pruning method (including subtree raising) is replaced by a simpler, more aggressive "reduced error pruning" method. Reduced error pruning sets aside part of the training data as a validation set, and then evaluates the effect of pruning tree nodes

using that validation set. In practice it gives smaller trees that, in theory, generalize better.

We also used two variants of the ubiquitous support vector machine, which we refer to as “SMO” and “SVM”. SMO refers to Platt's sequential minimal optimization method (Platt, 1999), which solves the large quadratic programming (QP) problems central to support vector machine training by breaking them into the smallest possible QP sub-problems and solving them analytically. SMO is considerably faster than traditional support vector machine training techniques, particularly for linear kernels. Our “SMO” algorithm used sequential minimal optimization to train an SVM with normalized input and a linear kernel (technically a first degree polynomial kernel). Our “SVM” algorithm used the libSVM library (Chang and Lin, 2001) via Weka, which provides a broad range of support vector machine types and options (it also uses a form of the SMO algorithm under the hood, but we maintain the “SMO” and “SVM” designations for our two classifiers as a convenience). We used the C-SVC SVM for 2-class classification with normalized input and radial basis function (RBF) kernel $e^{-\gamma(|\vec{u}-\vec{v}|^2)}$. Both SMO and SVM algorithms have a tunable complexity parameter C that represents the penalty in the loss function for non-separable samples. A higher complexity parameter value assigns a higher penalty for non-separable samples, thereby increasing the number of support vectors used. High complexity values can lead to overly complex machines and over-fitting, while low values can cause under-fitting. The SVM also has a γ parameter that affects area of influence of each radial

basis functions. Larger gamma values cause a faster fall-off of RBF influence, and are therefore more flexible and prone to over-fitting, while smaller values give each RBF a "wider" base. We explore tuning these parameters in Choosing and Refining Classifiers.

We trained each of the four algorithms – C45, C45R, SMO, and SVM – using the ARFF format file from the "score file generation" process above. Cross-validation accuracy and other results are described in Results below.

Machine Learning: Prediction

Given a new protein of interest, the testing process for that protein (i.e., predicting temperature-sensitive mutations) is very similar to the training process. Generation of mutants at buried positions, relaxation runs, and score file generation all proceed the same way, creating an ARFF format file for prediction. The difference between this ARFF file and the training ARFF file is, of course, that the labels for the generated mutations in the prediction ARFF file are unknown. Once the learning algorithms have been trained as described above, testing is simply a matter of running the trained algorithm on the new data.

Results

Training Set Cross-Validation

Our primary means of gauging the validity of our method was examining ts predictions from cross-validation runs of the different classifiers on the training set. To generate the results used in this section, we performed training runs on the training set "run19", which contained 95 input features, as not all of the terms described in Final Score File Assembly were available during the earlier stage of investigation (see Table 6 for a list of training files and features).

For each of the four initial learning algorithms, we started with Weka 10-fold cross-validation: using 10 data splits of 90% training, 10% testing, we obtained a prediction for each sample in the set. We then performed 10 iterations of the cross-validation procedure for each classifier using different random seeds to create the 90%/10% splits, allowing us to calculate a mean and a rough variance for each method across the cross-validation runs. We evaluated these sets of cross-validation runs using three criteria: precision, significance, and correlation. In all plots with error bars to follow, the value plotted is the mean, with error bars extending from Q1 to Q3 of the quantity being measured across the 10 iterations of cross-validation.

Precision

Accuracy was measured by the mean precision of cross-validation of the classifiers on the training set. Precision on the entire data set (all species) for each classifier is shown in Figure 23. Each method is significantly better than random, with the tuned SVM performing best.

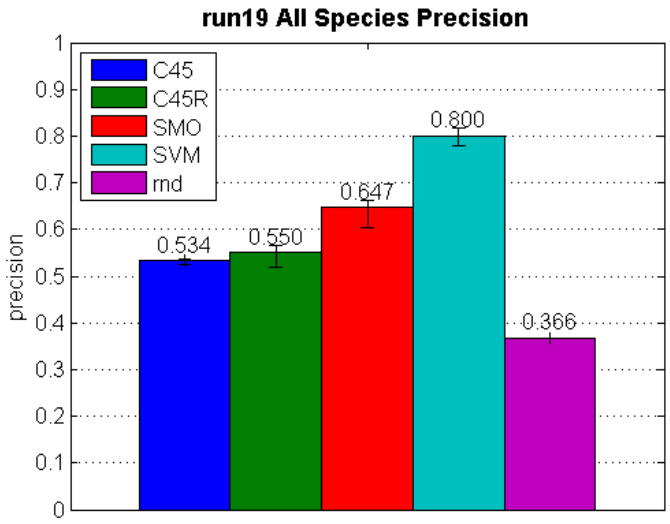

Figure 23 Overall precision per method by classifier type

The SVM classifier was tuned for the run19 data set. The other classifiers are as described earlier.

We then separated these predictions by species and calculated the error rate separately for each species and training method as shown in Figure 24. Precision was lower for *C. elegans*, where the number of ts samples is very small (5), and perfect for *D. melanogaster*, where there were no non-ts samples. Each method outperformed

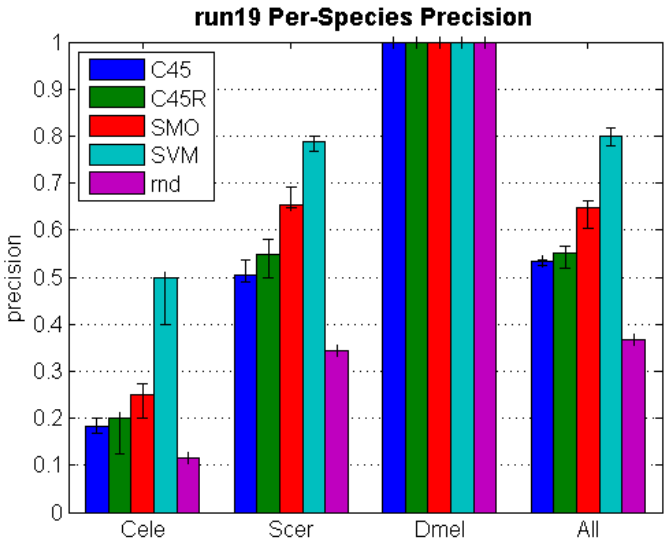

Figure 24 Precision by classifier and species

random guessing on a per-species basis (except on species Dmel, which has no non-ts samples).

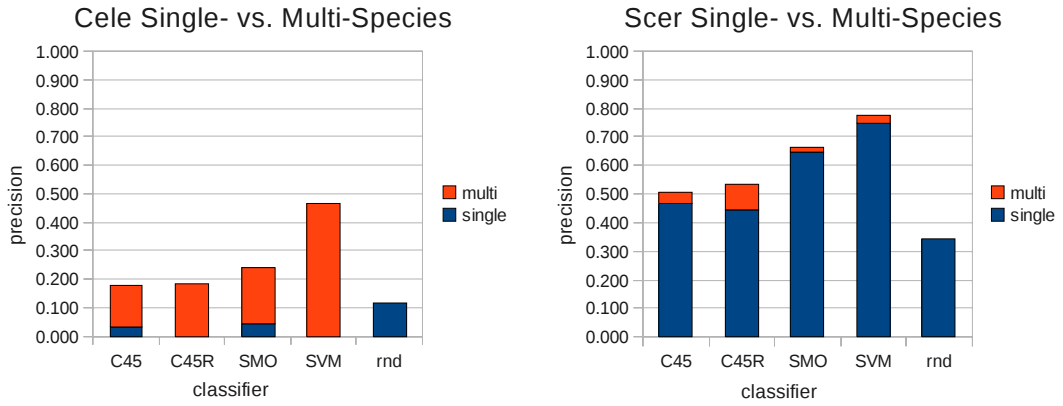

Figure 25 Single- versus multi-species training

Comparison of cross-validation precision on single-species vs. multi-species training sets. In both plots, the red areas show the precision gain from multi-species training.

Cross-species training also made a difference. Figure 25 compares single-species (i.e., limited to samples from a single species) cross-validation results with the multi-species results just shown. For both yeast and worm, each method shows a decline in precision for single-species cross-validation compared to multi-species; in *C. elegans*, the precision dropped to nearly zero using single-species cross-validation. While there were clearly species-dependent effects shown in Figure 24, we believe that the increase in accuracy from cross-species training indicates that our technique is extracting some species-independent rules. Even if the improvement is simply a function of training set size, these results do not seem to reflect species-dependent effects – otherwise, using multiple species would not have rescued *C. elegans* prediction rates.

Significance

To quantify the improvement over random provided by each prediction method, we used a heuristic method to estimate p-values for each method and species (Figure 26). Again, *C. elegans* has the weakest results; most other methods/species are significantly better than random, with C45 generally performing the worst and SVM performing best.

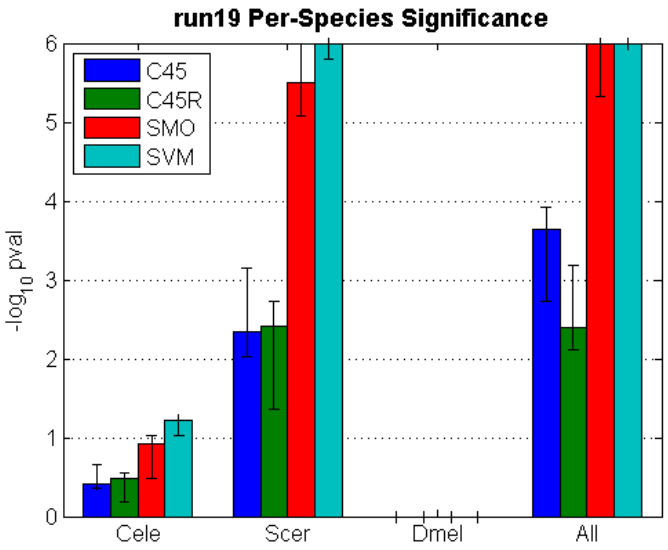

Figure 26 Precision p-values

Significance of precision compared to random, shown as $-\log_{10}(p)$, clamped at $p=10^{-6}$.

The algorithm we used for p-value estimation is as follows:

- Find the number P of predicted ts mutations made on set S (e.g., *S. cerevisiae*), and the number C of those guesses that were correct
- Perform the following steps N (usually 10^7) times:
    - Randomly draw, without replacement, P samples from S
    - If the number of ts samples in P is greater than or equal to C, increment a counter R
- R/N is the p-value, i.e., the fraction of times the random process performed at least as well as the prediction algorithm

Point-Biserial Correlation

Each classifier assigns a confidence, or estimated probability, to each of its predictions that we use to rank its predictions. We wanted to know, for each classifier, how well this confidence correlated with actual correctness of prediction. To do so, we calculated the point-biserial correlation between the correctness of the predictions (represented as 0=incorrect, 1=correct) and the assigned confidence of the predictions. We define X as a vector of 0s and 1s representing prediction correctness, Y as an equal-length vector of confidences for the corresponding predictions, and M_S as the mean confidence (Y) of predictions in the S=0 and S=1 categories. The slope B of the regression line between the means can be calculated as:

$$B = \frac{M_1 - M_0}{1}$$

The point-biserial correlation r_{XY} can then be calculated as:

$$r_{XY} = \frac{B}{\sigma_Y}\sigma_X = \frac{M_1 - M_0}{\sigma_Y}\sigma_X$$

The point-biserial correlation for each method (across all species) is shown in Figure 27. C45 and SMO confidence measures are reasonably well-correlated with correctness: 0.318 and 0.353, respectively. SVM, despite having the highest precision, is less well correlated at 0.136, while C45R is only at .079. The dip in correlation for SVM may be due to over-training resulting from parameter optimization; this is also reflected in the higher variance, which may be brittleness caused by over-training. The

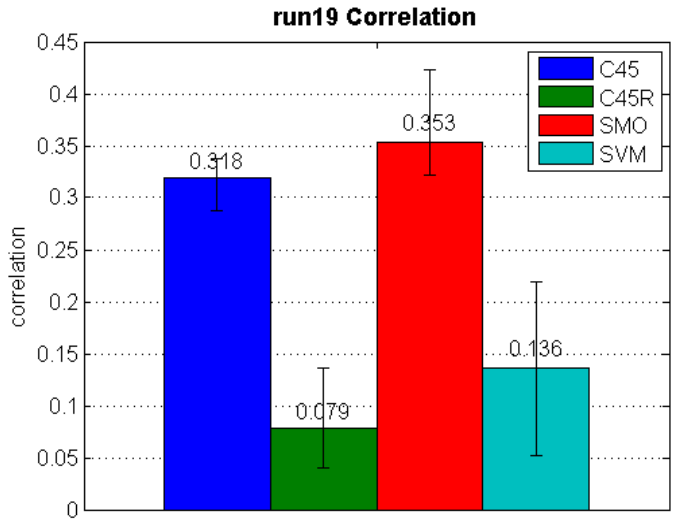

Figure 27 Point-biserial correlation

Correlation between prediction confidence and actual correctness for all learning algorithms.

low correlation of C45R, despite the method's reasonable precision and significance, is disturbing (some individual cross-validation runs were actually anti-correlated with correctness); we suspect this is a result of the reduced error pruning algorithm, which is less sophisticated than the pruning used by C45 and also reduces the amount of data made available for actual training (since part of the data is held out as a validation set for pruning).

Choosing and Refining Classifiers

Our initial results (evaluated in terms of precision, significance, and correlation) were encouraging: most classifiers scored well in at least two of these categories, and each consistently performed at least somewhat better than random. From the initial set of classifiers, we chose to further pursue the support vector machines and eliminate the tree-based classifiers. Both C45 and C45R had significance results that were too low – they were insufficiently better than random. In

addition, C45R confidence was not well-correlated with correctness of prediction. SMO performed solidly across all three metrics, and while SVM had lower correlation and generally higher variance, we pursued it as well because of its outstanding precision.

Ultimately, our goal was to find the one or two methods that provide the best performance on the training set while still promising the best possible generalization to new data. To find the optimal classifier, we explored both the parameter spaces of the SMO and SVM algorithms as well as the choice of features in the training set.

Choosing Features

There are 113 possible features for each entry in the training set, as described in Final Score File Assembly. Given the nature of the Rosetta scoring function, some of these features are likely to be highly correlated and perhaps even completely redundant. Other features were found to universally degrade performance and were removed from the training vector. Determination of features to remove was done in a very straightforward manner. After selecting a feature of interest, we performed multiple cross-validation runs on sets with and without that feature and compared performance using 5 different classifiers: SMO with default parameters, SMO optimized for run19, SVM with default parameters, SVM optimized for run19, and a bagged version of the SVM optimized for run19.

Set name	# features	Description
all	113	All possible features (reference only)
run19	95	Used for initial training; did not include sequence-based features, secondary structure features, or 6 features derived from 2 relax run terms
run25	102	Starting set for final correlation-based feature elimination.
run26	86	Final set after elimination of highly-correlated features.

Table 6 Training set sizes and descriptions

We first tested several of the non-Rosetta structure-based features, such as ACCP, that were not derived from Rosetta runs but still relied on the protein structure. This group included the log odds amino acid group change features (acodds, acgodds) and the secondary structure membership features (ss_S, ss_H, ss_L). These terms were tested together simply because the programmatic means of including or excluding them from the training file were similar. We found that all five of these features consistently reduced precision across all classifiers and removed them from the training set.

Next we generated a table of correlation coefficients between all features across all the entries in the training set (a visualization of this table for a subset of features is shown in Figure 28). From the table of coefficients, it was immediately apparent that the irms Rosetta score term provided no information and that the maxsub term was reporting exactly the same information as the gdtmm7_4 term. We removed

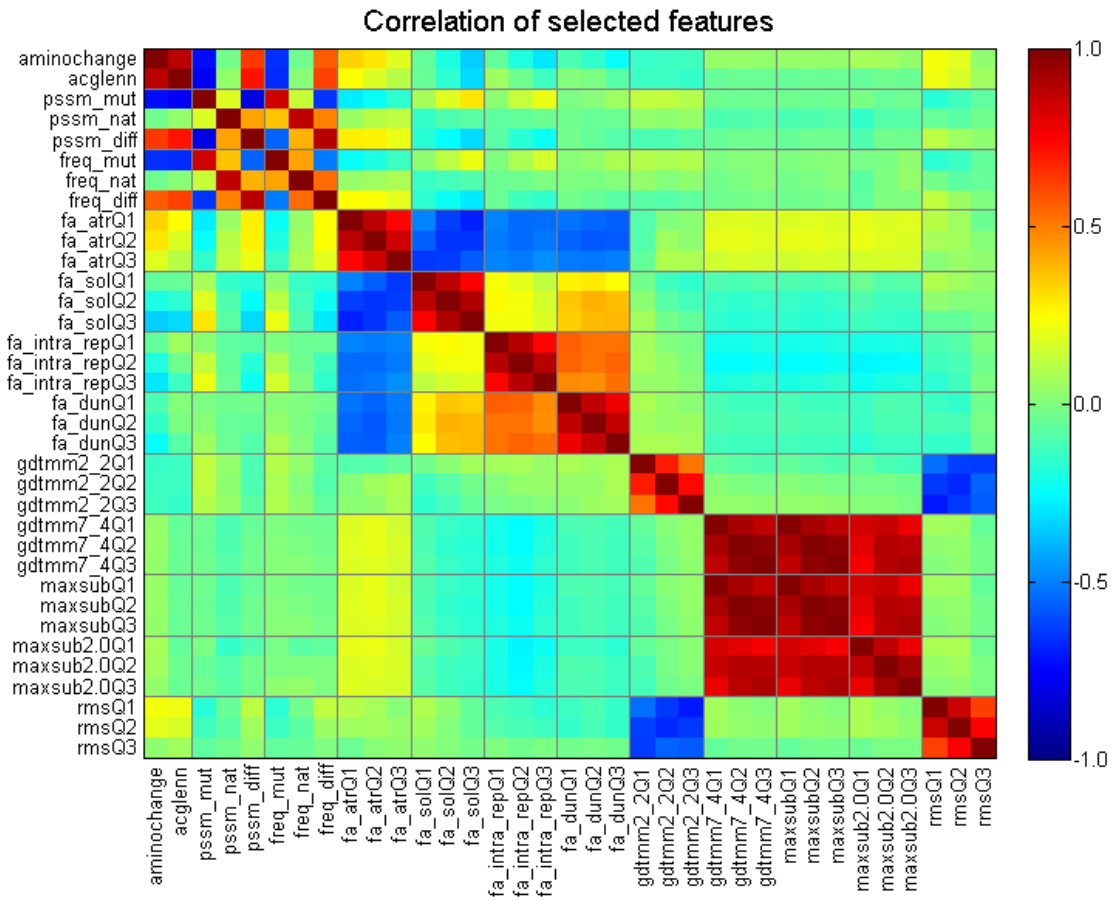

Figure 28 Correlation of selected input features

Correlation of a selected set of features, some pairs of which are redundant. Correlation coefficients are shown from blue (-1.0) to red (1.0).

the six features derived from the two terms, yielding an increase in accuracy in 4 of the 5 classifiers and a very small decrease in the other. The removal of the five structure-based features described earlier and the six features above resulted in the "run25" training set, which was the basis for all further investigation of potentially redundant features.

We then used the same "remove and test" method to remove a further 16 features, consisting of 4 sequence-based features and 12 features derived from four

Rosetta score terms. The strongly (anti-)correlated pairs involved are shown in Figure 28. As an example, aminochange and acglenn, both terms describing amino acid class change, were found to be strongly correlated, and "aminochange" was removed. Similarly, the rms and gdtmm2_2 feature sets were strongly anti-correlated – which makes intuitive sense, as rms describes the deviation of the relaxed model from the native structure, while the gdtmm2_2 term describe the fraction of atoms in the relaxed structure that remained within 2.2Å of the native – so the "rms" features were removed.

Optimizing Parameters

As described in Machine Learning: Training, the SMO and SVM classifiers have tunable parameter(s) that affect classifier performance. After generating the run25 and run26 training sets, we sought to optimize the parameter values for on each training set by exploring the 1- or 2-dimensional space of parameter values. As before, our criteria for evaluation was the average precision over 10 cross-validation runs using different random seeds for data splitting.

The SMO method has one tunable parameter, namely the C (complexity) parameter. We performed two series of runs on the run25 and run26 data sets with varying values for the C parameter. The coarse run was used to probe a wide range of C values to find the highest-precision area for further investigation, and the fine run

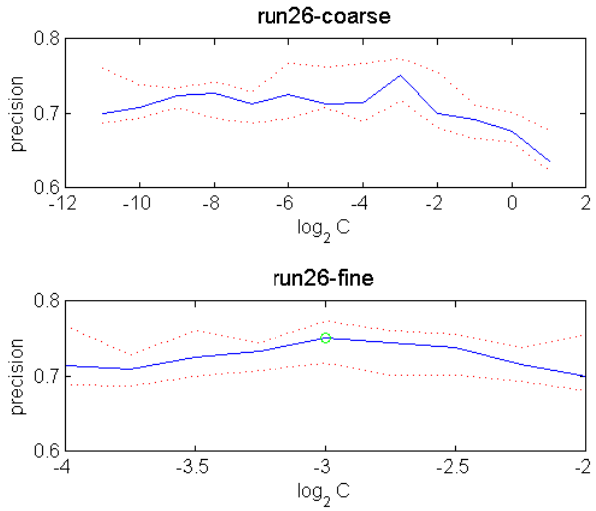

Figure 29 SMO C-parameter tuning on run26

Average precision (blue line) of SMO runs with the given C parameter value, with 1st and 3rd quartiles shown by red dashed lines. The final parameter choice of 2^{-3} is shown by the green circle in the lower plot.

was used to identify the optimal parameter value within the highest-precision area. The best results were for data set run26 with $C=2^{-3}$, as shown in Figure 29.

The SVM method has two tunable parameters, C and γ. We therefore performed a grid search, varying the two parameters, and using both coarse- and fine-grained parameter spacing as with the SMO tuning. Figure 30 shows the grid search for both coarse and fine setting for run26, and the final C and γ choices. We chose two sets of parameters based on the grid search. One set, the "best" parameters, correspond to the highest precision in the plot, at $C=2^{3.5}$, $\gamma=2^{-3.75}$. The other set, the "safe" parameters, are based on the grid search plot for all (ts and non-ts) samples (not shown) as well as the ts-only plot, with $C=2^{2.5}$, $\gamma=2^{-4.25}$.

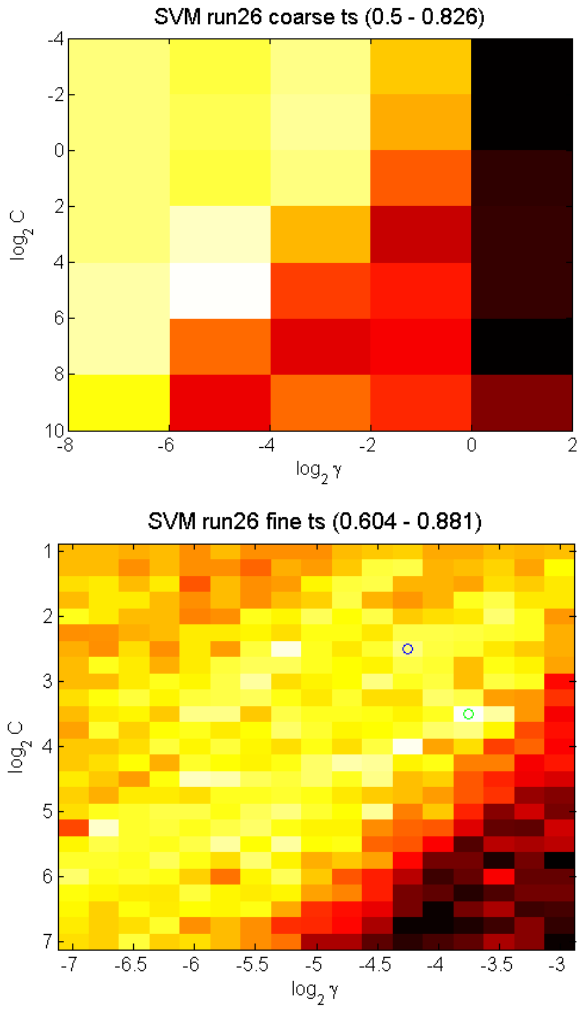

Figure 30 SVM C and γ parameter tuning

Average precision of SVM runs, shown as a heat map. Precision ranges are given in the plot titles. Two sets of final parameters were chosen: "best", marked by the green circle, and "safe", denoted by the blue circle.

One feature of note in the plots in Figure 30, particularly the fine-grained plot, is that there appears to be a line of higher accuracy extending roughly from the lower left to the upper right of the plots. This line describes an area of inverse correlation between C and γ. This makes sense, given the roles of C and γ in determining machine complexity and over- or under-fitting as discussed in Machine Learning: Training. The effect is rather subtle, but might bear further investigation: perhaps the parameter space is essentially one-dimensional, with a simple function relating the two parameter values. Though this is interesting, we have not further investigated this hypothesis.

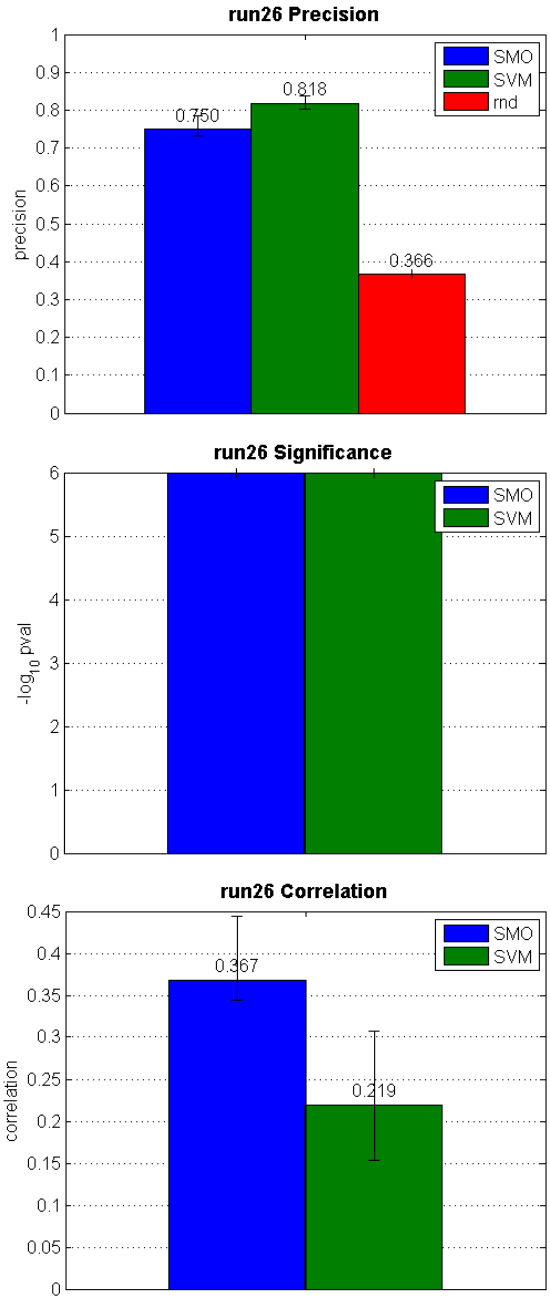

Figure 31 Tuned classifier results

Precision, significance, and correlation for the tuned classifiers (SMO and SVM).

Figure 31 shows the results of parameter tuning for the best two methods, both based on the smaller "run26" training set: the tuned SMO ($C=2^{-3}$), and the tuned SVM using the "safe" parameter set ($C=2^{2.5}$, $\gamma=2^{-4.25}$). Compared to the results on the initial, exploratory set "run19", both SMO and SVM method have improved in both precision and correlation, while significance remains pinned at $p<10^{-6}$.

These two classifiers, the tuned SMO and the tuned SVM, are the base classifiers that will be used from here on.

Using Sequence Terms Alone

Previous methods (Varadarajan et al., 1996; Chakshusmathi et al., 2004) have predicted temperature-sensitive mutations from sequence alone. Since our training sets include both sequence and structure terms, it is important to quantify the advantage of adding the structure-based terms, which are more computationally intensive to compute.

We carried out our analysis of sequence-only training in the same fashion as for the entire (sequence and structure) set: starting with a training set consisting of all sequence-based features, we chose the optimal set of input features, tuned the SMO and SVM methods, and looked at the final

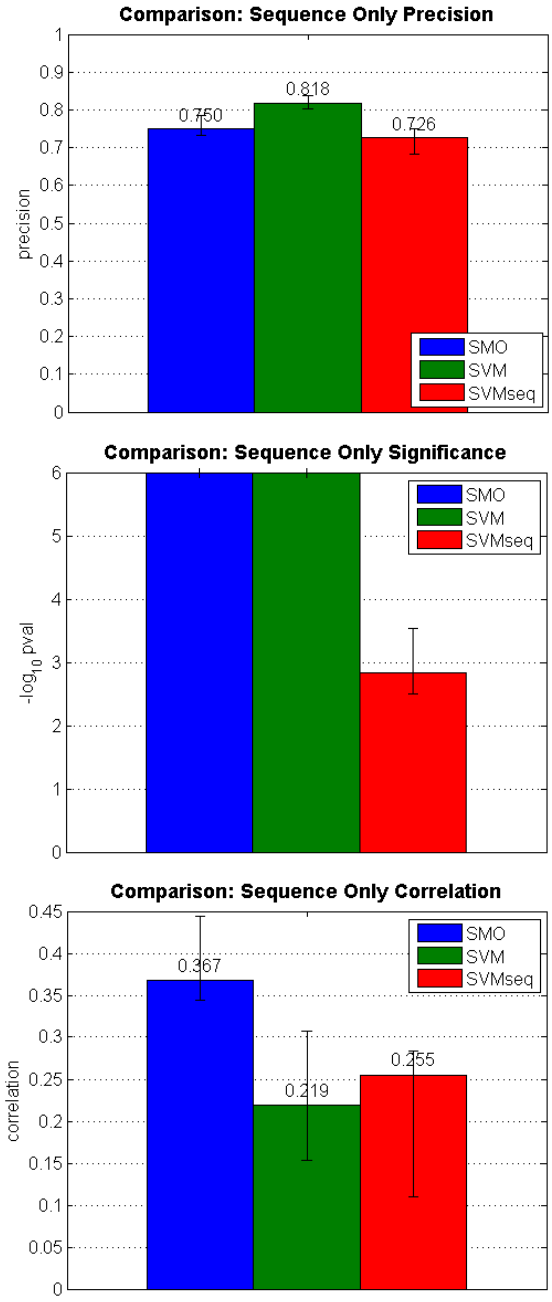

Figure 32 Sequence-only results

SMO and SVM are the tuned classifiers previously described. "SVMseq" is the best sequence-only classifier.

precision, significance, and correlation. The results are compared with the results on the optimal sequence & structure set (run26) in Figure 32. The best sequence-only method, referred to as "SVMseq", is the tuned SVM with parameters C=2, $\gamma=2^{-5}$, as found by the grid search procedure.

The graphs reveal several important differences. While sequence-only precision compares favorably with SMO precision on the run26 set, and sequence-only correlation is slightly better than SVM correlation on run26, significance is at least 3 orders of magnitude worse. Furthermore, the margin of error of both precision and correlation figures is higher for the sequence-only set. This corresponds with the brittleness that we noticed during the construction of the sequence-only training set: small changes, particularly the introduction of an unfavorable term such as log odds amino group change, had a large and often detrimental effect on results. The smaller, sequence-based set seems at this point to be inherently unstable.

In essence, the addition of structure vastly improves significance, and increases precision and correlation as well. If high precision is desired, the tuned SVM with run26 (the best sequence & structure set) is preferable to the sequence-only approach; if high correlation is more important, tuned SMO on run26 is better. While the sequence-only results are encouraging for possible future work on purely sequence-based prediction, we believe that the structure terms are integral for achieving the best results.

Meta-Learning Algorithms

A number of so-called "meta-learning" algorithms exist that aim to improve classifier results independent of the actual classifier being used. In an attempt to improve the performance of our tuned classifiers, we tested AdaBoost (Freund and Schapire, 1996), bagging (Breiman, 1996), and two types of cost-sensitive classification.

AdaBoost and bagging use multiple instance of a base classifier to improve performance. AdaBoost uses multiple instances of a base classifier in series, using the current series of classifiers to weight the training samples so that incorrectly-classified samples are more likely to be chosen by the next classifier in the series. Essentially, as classifiers are added, each new classifier is trained to distinguish among the samples that are "harder" to classify. The final choice of label for a sample is generally by voting among the series of classifiers. By contrast, bagging uses multiple parallel instances of a base classifier, each trained on a subset of the data. Final classification is again by voting by the ensemble of classifiers. The data "subset" may be 100% of the data; for cross-validation, this is the same principle as performing multiple CV runs with different seeds for data splits.

Cost-sensitive methods make the base classifier sensitive to the "cost" associated with making different classifications (usually misclassifications). They do so by either reweighting the training samples or classifying by minimum expected cost instead of likelihood. By default, the penalty for a correct classification is zero, and the

penalty for any incorrect classification is one. In our case, we assigned a higher cost to incorrectly labeling non-ts samples as ts, and classified by minimum expected cost. Cost-sensitive methods are often improved by using a bagged version of the base classifier which, because of the resampling involved, often improves the probability estimates of the base classifier. The MetaCost (Domingos, 1999) algorithm does this explicitly; we also ran a basic cost-sensitive algorithm with and without bagging the base classifier.

Specifically, we tried the following meta-methods: AdaBoost with 10 iterations; bagging, with 10 or 30 iterations and 60, 70, 80, 90, or 100% bag size for resampling; cost-sensitive classification with a penalty of 2, 3, 4, or 6 for misclassification of non-ts samples as ts; bagged versions of the above cost-sensitive classifier with the same penalties; and the MetaCost algorithm with the same set of misclassification penalties. We tried each of these meta-learning methods with both the tuned SMO and the tuned SVM discussed in Optimizing Parameters.

Surprisingly, none of these meta-methods provided sufficient improvement across all three measures of performance to supplant either of the original tuned classifiers. AdaBoost had lower precision, and as well as either lower correlation or significance, depending on the base classifier. Bagging results were roughly equivalent across all three measures when based on SMO; when based on SVM, the result was higher correlation at the expense of lower precision. Cost-sensitive training without bagging resulted in lower precision, as well as lower correlation and higher

variance in correlation that both increased with larger penalties. Cost-sensitive training with bagging sometimes yielded higher precision, but with much lower correlation. In addition, the bagged cost-sensitive methods tended to predict fewer ts samples overall; not only did this reduce significance, but the resulting ratio of ts:non-ts predictions was vastly lower than we believed should be expected. MetaCost methods sometime yielded higher precision, but had the reduced significance issues of the bagged cost-sensitive classifiers as well as the low correlation and high variance in correlation of the cost-sensitive classifiers.

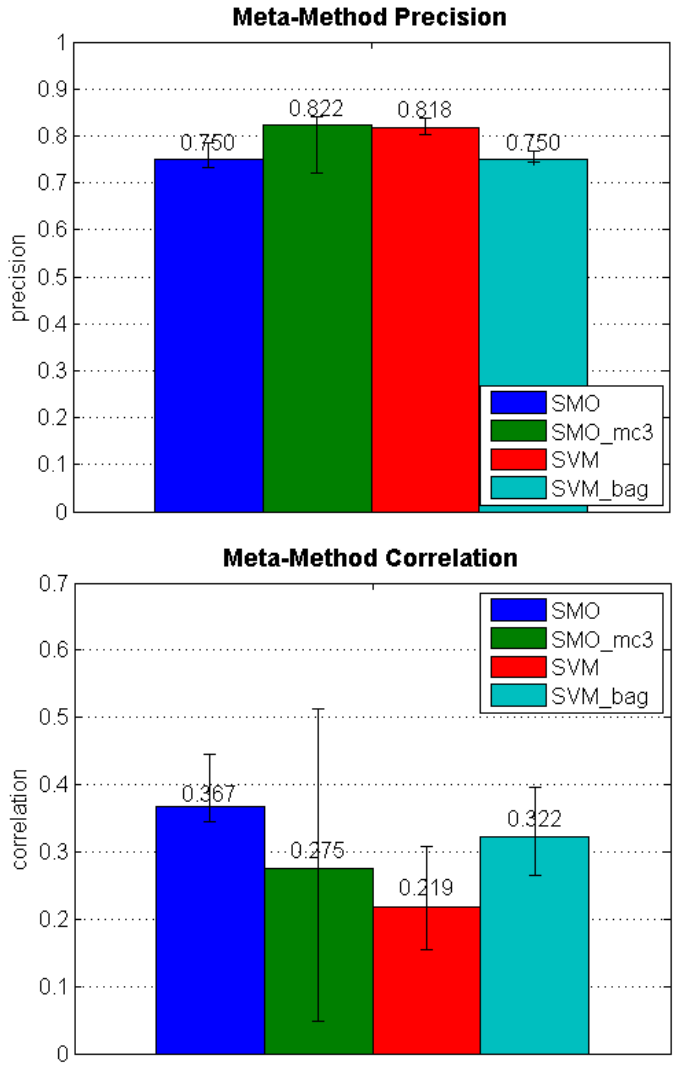

Figure 33 Comparison of base and meta-learning methods

Precision and correlation are shown for the tuned SMO, SMO with MetaCost using a penalty of 3 (SMO_mc3), tuned SVM, and bagged SVM (SVM_bag).

In order to clarify our

decision to stay with the base classifiers instead of any of the meta-learning methods, we show results from the best two meta-learner candidates along with results for the plain base classifiers in Figure 33. The two selected meta-methods were MetaCost with a penalty of 3 and the tuned SMO as base classifier, and bagging with 100% bag size and the tuned SVM as base classifier. Of all the methods, these were the only two that offered potential gains over the base classifiers. However, while the MetaCost method gives the highest precision, its lower correlation and wide variance in correlation, coupled with reduced significance (not shown) disqualify it. The bagged SVM offers significantly increased correlation over the base SVM, but its precision is reduced to roughly that of the tuned SMO while its correlation is still lower than that of the tuned SMO. In other words, the tuned SMO is superior to the bagged SVM.

Final Evaluation

We are left with two classifiers: the tuned SMO, which has precision of .750, p-value < 10^{-6}, and correlation of .367; and the tuned SVM, with a precision of .818, p-value < 10^{-6}, and correlation of .219. Rather than choose one of the two methods as appropriate for all situations, we suggest the following: use the SVM for data similar to data in the training set (e.g., globular yeast proteins), where an increase in precision is worth the risk of decreased correlation, and use the SMO for data less similar to the training set, where correlation is likely to be more important for getting a correct prediction in the top few predictions.

Protein-Based Statistical Analysis

All analysis so far – precision, significance, and correlation for the different classifiers and training sets – has been across all 66 proteins in the training set. However, since our goal is to produce a concentration of ts mutations at the top of a ranked list for a single protein of interest as opposed to a set of samples from different proteins, we needed to verify that our method was behaving correctly on a per-protein level – in other words, consistently placing temperature-sensitive mutations among the highest-ranking predictions for each of a number of different proteins. Within the limitations imposed by our training set, we tested our ranking as follows.

Analysis of Top-Ranked Predictions

We wanted to verify that our method's top-ranked predictions for a given protein were fact the most likely to be temperature-sensitive. We started by identifying the three proteins in our data set with both ts and non-ts mutations: yeast histone H3-H4 complex, yeast actin (YFL039C), and yeast YLR442C. For each of these proteins, we trained our SMO and SVM classifiers on a version of the training set without that protein's samples (a "leave-out" set), and tested on just that protein's samples (e.g., trained on the run26 set without actin samples, then tested on just the actin samples). We then looked at the precision within the top 5 ts predictions. The results are shown in Figure 34.

The results reflect somewhat the difficulties posed by the prediction task. For the histone complex, the SMO method fails to predict any correct ts sample in the top 5, and the 1 sample found by the SVM is essentially equivalent to random. However, this is probably due to the fact that histone is somewhat of an outlier in our training set: it is the only protein that was constructed from multiple chains of a larger complex (chains A and B of 1ID3, a complex of H2A.1, H2B.2, H3, H4, and DNA). The histone samples were nearly excluded from the training set because of their constructed nature and the fact that their inclusion slightly worsened performed on yeast; however, they improved overall accuracy and were thus kept. In addition, histone samples account for nearly one-third of the training set, and reduction on this scale of an already small training set is very likely to affect precision. YLR442C is also difficult: it has one ts sample and four non-ts samples, so precision will plunge if that one ts sample is incorrectly predicted or if (as happened in this case) a few non-ts samples are classified as ts. Actin is probably

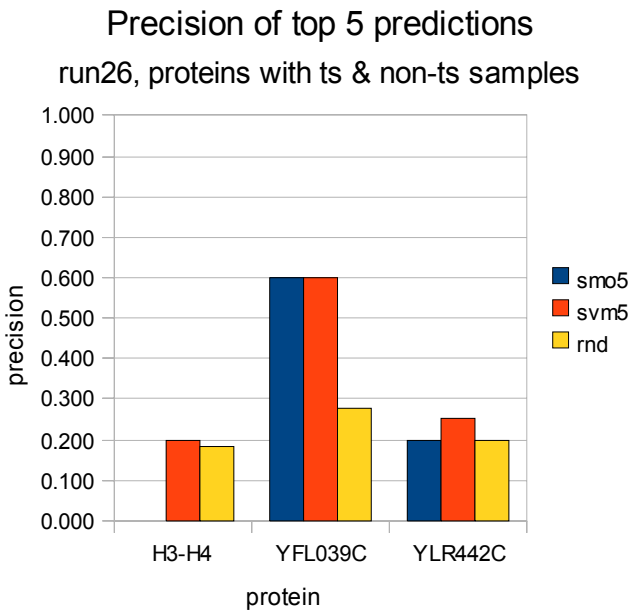

Figure 34 Precision of top 5 predictions

Prediction accuracy in the top 5 predictions for three different proteins, for both classifiers as well as random choice.

the best example, with 5 ts samples and 13 non-ts samples. In this case we do fairly well, with both methods finding three correct ts samples in the top five predictions – double what would be expected by chance.

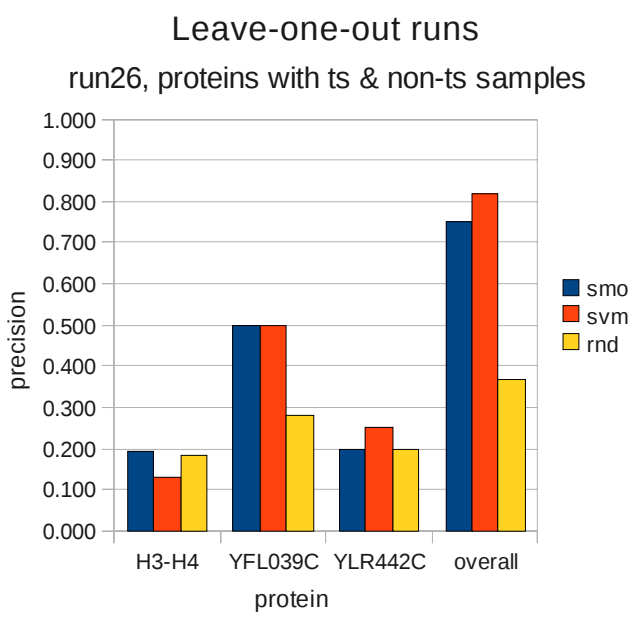

Figure 35 Leave-one-out run results

Precision for leave-one-out runs on the three proteins with non-ts and ts samples. Full-set CV results are shown for reference.

We also looked at precision over all known samples for each of these three proteins, performing "leave-out-one" runs for these proteins using the same leave-out sets as above. The results are shown in Figure 35. The results are similar to the top5 results. Histone continues to perform poorly, likely because of its outlier status and the fact that it makes up such a large part of the training set. YLR442C accuracy is poor because of the small number of samples, while actin performance is fairly good.

While none of these individual protein tests is as accurate as the classifier on the overall test set, the top 5 results, particularly for actin, do show that we can expect to accurately predict temperature-sensitive mutations in the top 5 in most cases.

Analysis of Expected Prediction Frequency

We wanted to know the distribution of high-confidence predictions across different proteins. Confidence should vary somewhat depending on protein size, fraction of buried residues, and other factors, but we needed to quantify the variation, and be sure we were not in a situation where some proteins had many high-confidence predictions while others had none.

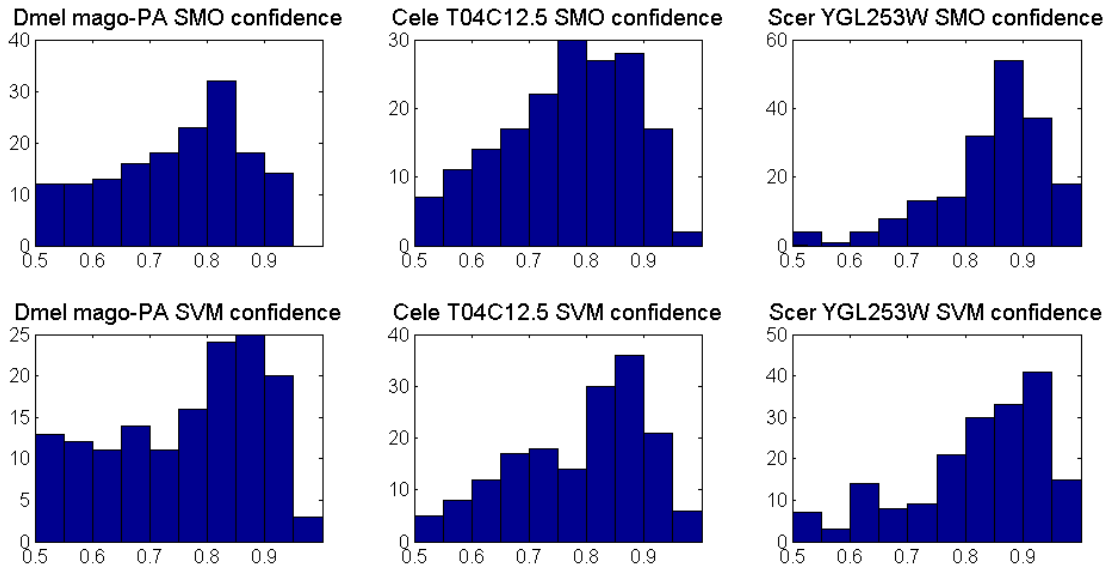

Figure 36 Distribution of ts prediction confidence scores

Distribution of the confidence scores of mutations predicted to be ts. Distributions are shown for three proteins using both classifiers.

Since most of our training data consisted of proteins with only a few known mutations, we needed to perform additional steps to determine what the variation might be on real runs, each of which will consist of large number of mutations at many sites on a single protein. In our training set, yeast histone was the only protein with

sufficient samples to give a credible distribution, and we eventually discarded it as having insufficient samples. Ideally, we would use data from a comprehensive set of mutations over several proteins (see Incorporating Systematic Data). Since this type of data is not yet available, we instead augmented our set for distribution analysis as follows: we chose three proteins, one from each species, based on homology model quality (mago-PA for fly, T04C12.5 for worm, and YGL253W for yeast). For each of these proteins we chose 10 buried sites at random that were not already part of the training set, generated models for all possible mutations at those sites, performed relax runs, generated score files, and made predictions using both SMO and SVM. The absence of labels for these mutations was not an issue, since all we needed was the confidence value assigned to the predictions made. This gave us a set of 190 confidence scores for each of the proteins chosen. The results are shown in Figure 36. The gross shapes of the distributions are similar, with some similarities to be seen across classifiers or proteins. More importantly, though the actual numbers change from protein to protein, there are sufficient samples of confidence .90 or better for each protein for our top 5 method to work.

Testing Predictions

We are in the process of testing our system by making predictions on the yeast actin protein (ACT1/YFL039C). Actin is an essential yeast gene that has been extensively studied and characterized. We chose to make predictions on actin because

1) some of our predictions may be verifiable via literature/database search instead of in-lab screening, and 2) if our hypotheses about prediction of ts mutants are correct, we may be able to find heretofore undiscovered ts alleles of this gene.

We performed prediction runs on the actin monomer, which is 375 residues in length and has 184 residues with 10% ACC or less. Our model was based on chain A of 1YAG, a complex of yeast actin and human gelsolin segment 1, with 99.2% sequence identity.

We generated two lists of ranked predictions, one from the SMO method and one from the SVM method (the actual classifier parameters and training set differed somewhat from the ones described above as these predictions were made before the final parameter tuning and training set selection). We combined the top 5 SMO predictions and the top 6 SVM predictions into an overall top 10 list (1 prediction was common to both lists), then chose 7 for actual testing based on our evaluation of the plausibility of the predictions. None of the mutations in the top 10 list had been previously characterized.

The ranked list has been given to David Gresham, who is performing lethality assays on our 7 chosen mutations to screen for ts phenotypes. At the time of this writing, these results are not yet available.

Future Work

Frequency of Temperature-Sensitive Mutations

Further analysis is required to evaluate the predictive power of our algorithm. The different learning algorithms tested varied somewhat in the fraction of mutations they predicted to be ts. If the expected fraction of ts mutations were known, we could account for this during training to give a better balance in the output predictions. But what is the proper fraction of actual ts mutations we can expect? According to (Sandberg et al., 1995), 85% of substitutions of one hydrophobic residue for another at buried positions were temperature-sensitive for the proteins they studied. But this assertion, in fairly early work, has not been borne out by other studies that we know of. We would also expect the fraction of ts mutations to be lower when not restricted to hydrophobic residues only. At the other extreme, one of the strongest arguments in favor of predictive methods – the PCR mutagenesis study of the ~200-residue miniGal4 from (Chakshusmathi et al., 2004) – showed no ts mutations among the 20,000 generated, suggesting that ts mutations may be very rare.

We suspect the answer is somewhere in the middle, but there is not sufficient data to perform a large-scale analysis. As discussed in the Training Set section, explicit "temperature-sensitive" annotation of alleles in the major yeast, worm, and fly databases is still inconsistent, and we have not found any method for explicit "non-ts" annotation. The larger issue is that many mutations are not screened for temperature

sensitivity, so that without reading the original reference for an allele that is not annotated as ts, the strongest statement we can make about that allele is that we don't know whether or not it is displays a ts phenotype. For example, in a survey of alleles of roughly 15 fly genes, at least one allele turned up as both ts and putative non-ts (i.e., not annotated as ts). A further complication is that determination of ts behavior depends upon the particular assay being done; an allele might show a ts phenotype in an assay for methylation, but not in an assay for growth.

The best solution to this issue lies in more data; specifically, in systematic data collected across all mutations at all positions for a handful of proteins (see below). (Bajaj et al., 2008) came close to providing this data for CcdB, but their use of an inducible system made the data unreliable for our purposes. At present, we will have to rely on one or more rounds of prediction and in-lab testing to gain a clearer idea of what fraction of mutations we can expect *a priori* to be ts. In the meantime, we hope that our work, in addition to simplifying the process of generating ts mutations, will also advance the state of ts annotation in databases – through the data curation we have performed so far, and by making explicit ts studies more approachable and therefore increasing the quantity data correctly annotated for ts or non-ts behavior.

Improving Results

Increasing Training Set Size

Additional data, presuming that they provide reliable signal (unlike the first attempt at fly non-ts samples) have consistently improved prediction results. In particular, fly non-ts samples and a well-balanced collection of ts and non-ts samples from other organisms should improve our prediction considerably and give specific feedback about the precision of our fly predictions.

Another option for generating more samples would be to use the log odds scores from the PSSM to find positions in training set proteins that do not have a strong preference for any amino acid. The mutations at these positions could be assumed to be tolerated and added to our training set as putative non-ts samples.

Incorporating Systematic Data

Additional data improve accuracy, and as mentioned earlier in Frequency of Temperature-Sensitive Mutations, the best type of data for prediction would come from systematically making every possible amino acid substitution at every buried position for several proteins. In collaboration with the Gresham lab, we hope to have access to just such a wealth of training data, which will most probably be generated using a life-or-death assay on a collection of yeast essential genes.

Improving Speed

As discussed in Model Relaxation, most of the time required for our method is spent executing the Rosetta relax runs. The time required for each run on a protein is a function of the length, in residues, of the protein: the longer the protein, the more time the run requires. Proteins with over 300 residues can take days of cluster time to complete. It is not clear, however, that the relaxation algorithm needs to be run on the entire protein for each mutation. On the one hand, as discussed in Sandberg et al, a single amino acid substitution may cause a change in structure that propagates through the protein, e.g., from a buried residue to a surface residue. On the other hand, inconsequential variation in a loosely-constrained loop region well away from the mutation site may cause noise in the score terms that obscures the signal arising from the mutation. In the former case, a global algorithm that considers all residues would be more accurate; in the latter, a local algorithm that considers only residues within a constrained neighborhood of the mutation would be preferred.

We have made a version of the Rosetta relax protocol that performs local relaxation by constraining the relaxation algorithm to only those residues listed in an additional input file. This constraint file is generated by a PyMOL script that finds all residues within a given "neighborhood" of the mutation in question. We have verified that the "local relaxation" protocol runs at 5-10 times the speed of the global protocol, depending on neighborhood size and overall protein size; however, we have not done a rigorous comparison between global and local relaxation results. Initial results with a

neighborhood of 8Å from the mutant position Cα atom indicate that the size of the neighborhood around the mutated residue is key: if it is too small, accuracy will be unacceptable; if it is too large, speed will suffer.

One caveat is that the time savings is offset by the need to perform two runs for each mutation: one for the mutation itself, and one to generate a "local wt" ensemble for that mutation's neighborhood. Comparisons between wt and mutant run ensembles only make sense if relaxation was performed on the same set of residues. For the global algorithm, one wt ensemble was sufficient, since all runs considered all residues, but for the local algorithm, each set of residues generated by a mutation requires its own "local wt" ensemble for comparison on just those residues. In the general case, this means two sets of runs as opposed to one for each mutation.

Web Server

Ultimately we want to make our prediction process publicly available on the internet. We intend to provide a simple web front end to our method, which will:

1) Allow users to specify several parameters, such as the ACC cutoff to use or how many Rosetta relax iterations to run. ACC cutoff specification may be interactive and include visualization of the model with coloring to show the residues that will be included.
2) Distribute jobs to a cluster, perform job management, and collect output into score files.

3) Notify the user on completion and make a ranked list of predicted ts substitutions, perhaps accompanied by consensus models of those mutations, available for viewing and/or download.

References

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. J Mol Biol **215:** 403-410

Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res **25:** 3389-3402

Bader GD, Donaldson I, Wolting C, Ouellette BF, Pawson T, Hogue CW (2001) BIND--The Biomolecular Interaction Network Database. Nucleic Acids Res **29:** 242-245

Bader GD, Hogue CW (2003) An automated method for finding molecular complexes in large protein interaction networks. BMC Bioinformatics **4:** 2

Bajaj K, Dewan PC, Chakrabarti P, Goswami D, Barua B, Baliga C, Varadarajan R (2008) Structural correlates of the temperature sensitive phenotype derived from saturation mutagenesis studies of CcdB. Biochemistry **47:** 12964-12973

Ben-Aroya S, Coombes C, Kwok T, O'Donnell KA, Boeke JD, Hieter P (2008) Toward a comprehensive temperature-sensitive mutant repository of the essential genes of Saccharomyces cerevisiae. Mol Cell **30:** 248-258

Bernstein FC, Koetzle TF, Williams GJ, Meyer EF, Jr., Brice MD, Rodgers JR, Kennard O, Shimanouchi T, Tasumi M (1977) The Protein Data Bank: a computer-based archival file for macromolecular structures. J Mol Biol **112:** 535-542

Breiman L (1996) Bagging predictors. *In* Machine Learning, pp 24--22

Cerami EG, Bader GD, Gross BE, Sander C (2006) cPath: open source software for collecting, storing, and querying biological pathways. BMC Bioinformatics **7:** 497

Chakshusmathi G, Mondal K, Lakshmi GS, Singh G, Roy A, Ch RB, Madhusudhanan S, Varadarajan R (2004) Design of temperature-sensitive mutants solely from amino acid sequence. Proc Natl Acad Sci U S A **101:** 7925-7930

Chang C-C, Lin C-J (2001) LIBSVM: a library for support vector machines.

Dai J, Hyland EM, Yuan DS, Huang H, Bader JS, Boeke JD (2008) Probing nucleosome function: a highly versatile library of synthetic histone H3 and H4 mutants. Cell **134:** 1066-1078

DeLano WL (2002) The PyMOL Molecular Graphics System. DeLano Scientific, Palo Alto, CA, USA

Dohmen RJ, Wu P, Varshavsky A (1994) Heat-inducible degron: a method for constructing temperature-sensitive mutants. Science **263:** 1273-1276

Domingos P (1999) Metacost: A general method for making classifiers cost-sensitive. *In* Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD, pp 155--164

Eswar N, John B, Mirkovic N, Fiser A, Ilyin VA, Pieper U, Stuart AC, Marti-Renom MA, Madhusudhan MS, Yerkovich B, Sali A (2003) Tools for comparative protein structure modeling and analysis. Nucleic Acids Res **31:** 3375-3380

Eswar N, Webb B, Marti-Renom MA, Madhusudhan MS, Eramian D, Shen MY, Pieper U, Sali A (2006) Comparative protein structure modeling using Modeller. Curr Protoc Bioinformatics **Chapter 5:** Unit 5 6

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. *In* In Machine Learning: Proceedings of the Thirteenth International Conference, pp 148--156

Gene Ontology Consortium (2001) Creating the gene ontology resource: design and implementation. Genome Res **11:** 1425-1433

Gutiérrez RA, Lejay LV, Dean A, Chiaromonte F, Shasha DE, Coruzzi GM (2007) Qualitative network models and genome-wide expression data define carbon/nitrogen-responsive molecular machines in Arabidopsis. Genome Biol **8:** R7

Gutiérrez RA, Stokes TL, Thum K, Xu X, Obertello M, Katari MS, Tanurdzic M, Dean A, Nero DC, McClung CR, Coruzzi GM (2008) Systems approach

identifies an organic nitrogen-responsive gene network that is regulated by the master clock control gene CCA1. Proc Natl Acad Sci U S A **105:** 4939-4944

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD Explor. Newsl. **11:** 10-18

Harris TW, Antoshechkin I, Bieri T, Blasiar D, Chan J, Chen WJ, De La Cruz N, Davis P, Duesbury M, Fang R, Fernandes J, Han M, Kishore R, Lee R, Muller HM, Nakamura C, Ozersky P, Petcherski A, Rangarajan A, Rogers A, Schindelman G, Schwarz EM, Tuli MA, Van Auken K, Wang D, Wang X, Williams G, Yook K, Durbin R, Stein LD, Spieth J, Sternberg PW (2001) WormBase: a comprehensive resource for nematode research. Nucleic Acids Res **38:** D463-467

Hong F, Breitling R, McEntee CW, Wittner BS, Nemhauser JL, Chory J (2006) RankProd: a bioconductor package for detecting differentially expressed genes in meta-analysis. Bioinformatics **22:** 2825-2827

Huang H, Maertens AM, Hyland EM, Dai J, Norris A, Boeke JD, Bader JS (2009) HistoneHits: a database for histone mutations and their phenotypes. Genome Res **19:** 674-681

Ideker T, Ozier O, Schwikowski B, Siegel AF (2002) Discovering regulatory and signalling circuits in molecular interaction networks. Bioinformatics **18 Suppl 1:** S233-240

Kanemaki M, Sanchez-Diaz A, Gambus A, Labib K (2003) Functional proteomic identification of DNA replication proteins by induced proteolysis in vivo. Nature **423:** 720-724

Katari MS, Nowicki SD, Aceituno FF, Nero D, Kelfer J, Thompson LP, Cabello JM, Davidson RS, Goldberg AP, Shasha DE, Coruzzi GM, Gutiérrez RA (2009) VirtualPlant: A software platform to support Systems Biology research. Plant Physiol

Lazaridis T, Karplus M (1999) Effective energy function for proteins in solution. Proteins **35:** 133-152

Loris R, Dao-Thi MH, Bahassi EM, Van Melderen L, Poortmans F, Liddington R, Couturier M, Wyns L (1999) Crystal structure of CcdB, a topoisomerase poison from E. coli. J Mol Biol **285:** 1667-1677

Maere S, Heymans K, Kuiper M (2005) BiNGO: a Cytoscape plugin to assess overrepresentation of gene ontology categories in biological networks. Bioinformatics **21:** 3448-3449

Mnaimneh S, Davierwala AP, Haynes J, Moffat J, Peng WT, Zhang W, Yang X, Pootoolal J, Chua G, Lopez A, Trochesset M, Morse D, Krogan NJ, Hiley SL, Li Z, Morris Q, Grigull J, Mitsakakis N, Roberts CJ, Greenblatt JF, Boone C, Kaiser CA, Andrews BJ, Hughes TR (2004) Exploration of essential gene functions via titratable promoter alleles. Cell **118:** 31-44

Muller HM, Kenny EE, Sternberg PW (2004) Textpresso: an ontology-based information retrieval and extraction system for biological literature. PLoS Biol **2:** e309

Nemhauser JL, Hong F, Chory J (2006) Different plant hormones regulate similar processes through largely nonoverlapping transcriptional responses. Cell **126:** 467-475

Platt JC (1999) Fast training of support vector machines using sequential minimal optimization. *In* Advances in kernel methods: support vector learning. MIT Press, pp 185-208

Poultney C, Shasha D (2009) Software Tools for Systems Biology: Visualizing the Outcomes of N Experiments on M Entities. *In* G Coruzzi, RA Gutierrez, eds, Plant systems biology. Wiley-Blackwell, Chichester, West Sussex, U.K. ; Ames, Iowa, pp xvi, 360 p., [312] p. of plates

Poultney CS, Gutiérrez RA, Katari MS, Gifford ML, Paley WB, Coruzzi GM, Shasha DE (2007) Sungear: interactive visualization and functional analysis of genomic datasets. Bioinformatics **23:** 259-261

Pruitt KD, Tatusova T, Maglott DR (2007) NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic Acids Res **35:** D61-65

Quinlan JR (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc.

Rhee SY, Beavis W, Berardini TZ, Chen G, Dixon D, Doyle A, Garcia-Hernandez M, Huala E, Lander G, Montoya M, Miller N, Mueller LA, Mundodi S, Reiser L, Tacklind J, Weems DC, Wu Y, Xu I, Yoo D, Yoon J, Zhang P (2003) The Arabidopsis Information Resource (TAIR): a model organism database providing a centralized, curated gateway to Arabidopsis biology, research materials and community. Nucleic Acids Res **31:** 224-228

Rohl CA, Strauss CE, Misura KM, Baker D (2004) Protein structure prediction using Rosetta. Methods Enzymol **383:** 66-93

Sandberg WS, Schlunk PM, Zabin HB, Terwilliger TC (1995) Relationship between in vivo activity and in vitro measures of function and stability of a protein. Biochemistry **34:** 11970-11978

Schuldiner M, Collins SR, Thompson NJ, Denic V, Bhamidipati A, Punna T, Ihmels J, Andrews B, Boone C, Greenblatt JF, Weissman JS, Krogan NJ (2005) Exploration of the function and organization of the yeast early secretory pathway through an epistatic miniarray profile. Cell **123:** 507-519

Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T (2003) Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res **13:** 2498-2504

Sikorski RS, Boeke JD (1991) In vitro mutagenesis and plasmid shuffling: from cloned gene to mutant yeast. Methods Enzymol **194:** 302-318

Suzuki DT, Grigliatti T, Williamson R (1971) Temperature-sensitive mutations in Drosophila melanogaster. VII. A mutation (para-ts) causing reversible adult paralysis. Proc Natl Acad Sci U S A **68:** 890-893

Thimm O, Blasing O, Gibon Y, Nagel A, Meyer S, Kruger P, Selbig J, Muller LA, Rhee SY, Stitt M (2004) MAPMAN: a user-driven tool to display genomics data sets onto diagrams of metabolic pathways and other biological processes. Plant J **37:** 914-939

Tweedie S, Ashburner M, Falls K, Leyland P, McQuilton P, Marygold S, Millburn G, Osumi-Sutherland D, Schroeder A, Seal R, Zhang H (2009) FlyBase: enhancing Drosophila Gene Ontology annotations. Nucleic Acids Res **37:** D555-559

Vailaya A, Bluvas P, Kincaid R, Kuchinsky A, Creech M, Adler A (2005) An architecture for biological information extraction and representation. Bioinformatics **21:** 430-438

Varadarajan R, Nagarajaram HA, Ramakrishnan C (1996) A procedure for the prediction of temperature-sensitive mutants of a globular protein based solely on the amino acid sequence. Proc Natl Acad Sci U S A **93:** 13908-13913

Word JM, Lovell SC, LaBean TH, Taylor HC, Zalis ME, Presley BK, Richardson JS, Richardson DC (1999) Visualizing and quantifying molecular goodness-of-fit: small-probe contact dots with explicit hydrogen atoms. J Mol Biol **285:** 1711-1733

Zeidler MP, Tan C, Bellaiche Y, Cherry S, Hader S, Gayko U, Perrimon N (2004) Temperature-sensitive control of protein activity by conditionally splicing inteins. Nat Biotechnol **22:** 871-876

Zimmermann P, Hirsch-Hoffmann M, Hennig L, Gruissem W (2004) GENEVESTIGATOR. Arabidopsis microarray database and analysis toolbox. Plant Physiol **136:** 2621-2632