# Scalable Software Infrastructure for the Lab and a Specific Investigation of the Yeast Transcription Factor Eds1

Chase Mateusiak

Washington University in St. Louis

McKelvey School of Engineering

Department of Computer Science and Engineering

# Scalable Software Infrastructure for the Lab and a Specific Investigation of the Yeast Transcription Factor Eds1

By

Chase Mateusiak

A thesis presented to the McKelvey School of Engineering of Washington University in St. Louis in partial fulfillment of the requirements for the degree of Master of Science

May 2022

St. Louis, Missouri

Dedication

To my family.

## Acknowledgments

Michael Brent and the Brent lab as a whole have provided an unrivaled environment in which to learn how to manage biological data and conduct meaningful analysis. Thank you also to Seemay Chou and her lab, for giving me my start in this work.

I would never have known that I could contribute to science, however slightly, had I not met my partner, Laura DeVault. Her interest and dedication to discovery has been my inspiration.

My parents, Pam Gunkler and Wally Mateusiak, and my brother, Wesley Mateusiak, provide boundless love and support.

Finally, Boba, the dog, has reminded me to get up from the computer and go outside, which is always a better place to do some thinking.

## Table of Contents

## List of Tables

List of Figures

## Abstract

Scalable Software Infrastructure for the Lab and a Specific Investigation of the Yeast Transcription Factor Eds1

By

Chase Mateusiak

Master of Science in Computer Science

Washington University in St Louis, 2022

Research Advisor: Professor Michael Brent

Individual biology labs handle increasingly large data sets. Ensuring accurate data entry, consistent sample metadata, and ease of access to the data once it is stored, are critical for both the integrity of analysis as well as productivity of the lab. Chapter one of this thesis describes three implementations of software meant to facilitate handling data and metadata in the lab as the size of the data and complexity of analysis scale. The first piece of software is a database and entry interface for storing a large and varied amount of data on biological samples. The second is a software package which uses established bioinformatics tools to further build upon the database API to facilitate analysis. The third is a Nextflow pipeline for the Calling Cards protocol for assaying transcription factor binding via sequencing.

Chapter two focuses on *EDS1*, a gene encoding a transcription factor present in *Saccharomyces cerevisiae*. *EDS1* is a paralog of *RGT1,* an important regulator of sugar uptake. We show that Eds1p has both functionally redundant and divergent roles in response to external nutrients.

# Chapter 1: Scalable, Portable Software for a Biology Lab

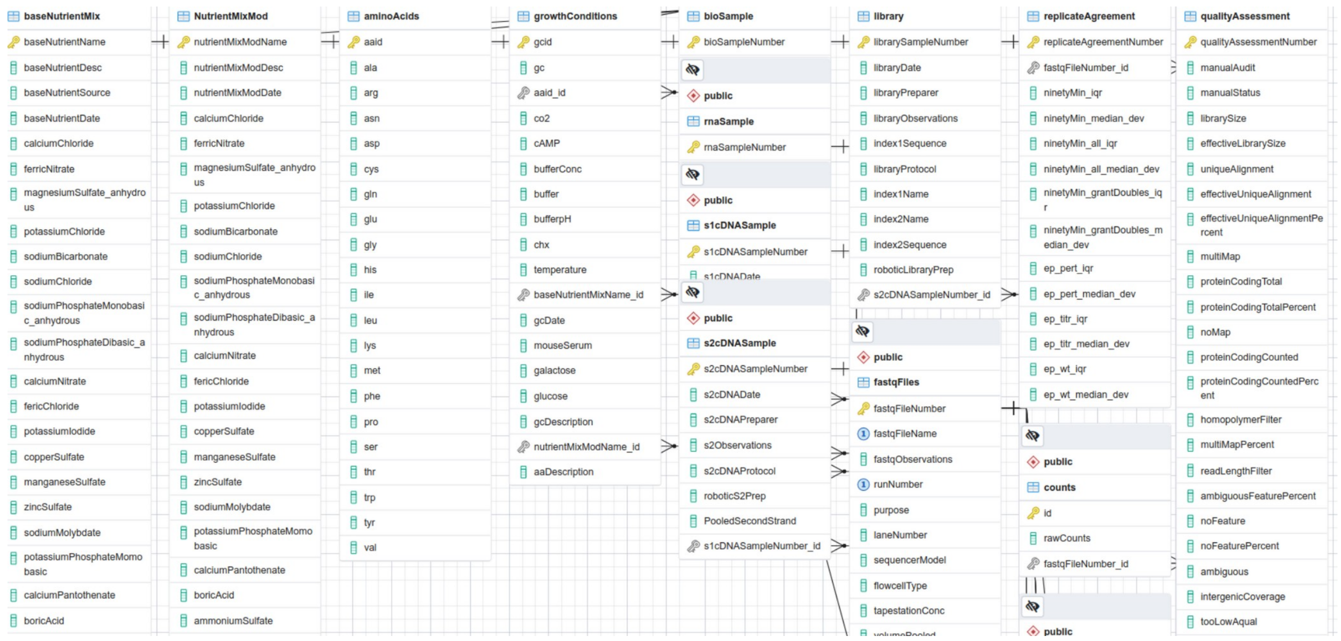

Figure 1: Schematic of the Brent lab sequencing database

## Database Management of Sequencing Data

To facilitate analysis of RNA sequence and other phenotype data generated on a large scale in the lab, I have implemented a database which tracks sample development over time, catalogs strains which corresponds to their storage in the freezer, allows for expressive and extensible description of growth and environmental conditions, and makes connecting additional data to each biological sample simple. The database is implemented in Django, which makes available to current and future developers of the codebase a rich set of tools and security measures which would be difficult for a single person to implement and maintain. The database table design conforms to standard database design principles, which has had a large and beneficial impact on data accuracy, both of historic data which was ported into this system, and in newly entered data.

I have implemented a browser based front end to facilitate data entry, and which further enforces accuracy standards. An implementation of the RNA sequence database may be viewed here:

https://github.com/BrentLab/KN99_database

The data entry front end may be viewed here:

http://18.190.159.45/

The code for the front end may be viewed here:

https://github.com/BrentLab/rnaseq_metadata_frontend

## brentlabRnaSeqTools: Package Infrastructure and Testing

To aid in the utilization of the data stored in the database, I have implemented the brentlabRnaSeqTools, an R package which extends the standard bioinformatics tools developed by Bioconductor[1]. This package provides a number of features which may be divided as follows: package infrastructure and testing, documentation, and finally application.

The brentlabRnaSeqtools package implements two methods of guaranteeing portability: continuous integration testing which is triggered every time new code is pushed, and containerization with Singularity. Continuous Integration testing simply means that the software package is installed onto a remote server, and then the tests are run, to verify that the software works as intended. In this case, every time new code is pushed to the brentlabRnaSeqTools repository, the codebase is built on the current version of the Mac OS, Windows, and Ubuntu. Singularity builds an ‘image’, which includes the OS, software dependencies, and the codebase. If the host system has Singularity installed, it may then run the image, regardless of what operating system the host system has, and what operating system the software requires. No other dependency management is required as it is all contained in the image file.

The purpose of implementing both of these features is to both ensure portability and also to provide concrete example to others in the lab of how to implement these features – the infrastructure of this package may be copied into any new project.

## brentlabRnaSeqTools: Documentation

Every function in the brentlabRnaSeqTools is documented for the user in such a manner that formatted, easy to read documentation is generated automatically. Continuous Integration encourages code documentation through  a test which requires that every function have at least a definition, description of each input, and an example usage. As one of the steps executed in the continuous integration protocol, the documentation is re-built  on every push of new code. The code itself is also well documented for future maintainers and developers, and care has been taken to make the code readable. Documentation may be viewed here, and is also available within  an active R session if the brentlabRnaSeqTools is installed:

https://brentlab.github.io/brentlabRnaSeqTools/

## brentlabRnaSeqTools: Application

The purpose of the brentlabRnaSeqTools package, beyond providing a template of a package primed for upload to either of the common R repositories, CRAN, or Bioconductor, is to facilitate the use of the data stored in the Brent lab database. The brentlabRnaSeqTools makes use of the fundamental objects of the Bioconductor software project, the SummarizedExperiment class, which stores assay data, sample metadata, feature metadata, and experiment metadata. Each of these attributes itself has object definitions from Bioconductor which facilitates operations such as extracting expression over user-defined promoter regions. SummarizedExperiment objects expose filtering and subsetting methods, among others, which greatly ease interaction with this data, and it allows the user to store all of this data together as a single file, rather than as many separate files. Utilizing SummarizedExperiment also provides both in-memory and on-disc based back ends, and a number of other features, all of which together provide an immediately scalable software environment for the user[1].

This package provides a framework for the implementation of contemporary, portable software. The goal with this package, in addition to its use as an interface to the Brent lab sequence database, is to provide a framework which makes the adoption of what are useful, but somewhat arcane, tools such as continuous integration, automatically generated documentation, and fundamental Bioconductor classes, more concrete and accessible.

## Calling Cards Pipeline Implementation in Nextflow

Nextflow is a domain specific language specifically built for bioinformatics pipelines[2]. It facilitates modular workflows by providing tools to chain the input and output of processes together by abstracting away the underlying system.

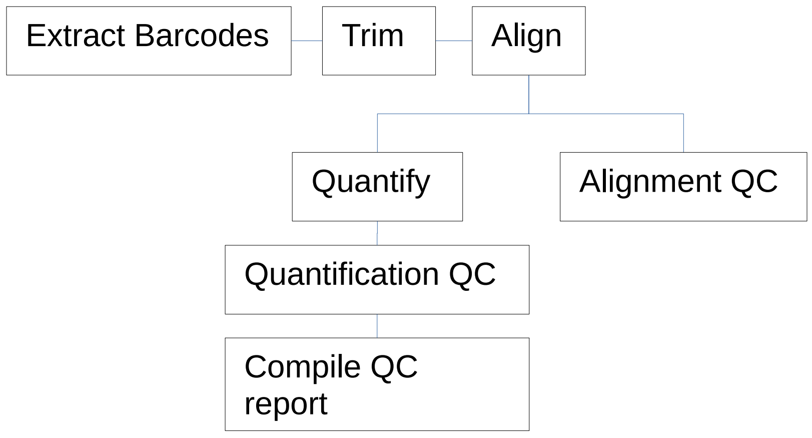

Figure 2: The Calling Cards pipeline

Every module is implemented using the standard open source tool commonly used for the same task. Each module is containerized and accessible via both bioconda, and a repository for Singularity images. The software is packaged in such a way that it complies with the guidelines set forth by nf-core, a public, though tightly managed, repository for Nextflow bioinformatics pipelines.

Calling Cards is a sequencing based method developed by Rob Mitra which is orthogonal to chromatin immunoprecipitation based methods to interrogate DNA binding transcription factors to discover their genomic targets. After sequencing a library of reads generated from the a Calling Cards experiment, the bioinformatics processing pipeline's first step is to extract the molecular bar codes from each read. The bar codes map the reads to a given transcription factor of interest. Next, reads are aligned. The aligned reads are then transformed into counts (depth) over each nucleotide in the genome (typically called 'piling up' the reads). This is transformed into a format which, given promoter definitions, allows fast quantification of reads over promoters by bar code (transcription factor). Quality assessment metrics are generated at the sequencer, alignment and quantification steps.

Each step in this pipeline is implemented in a fully containerized Nextflow module. Furthermore, each step is conducted by the software package which is considered the standard software for each task, and is fully containerized using both Conda and Singularity. The pipeline is implemented in such a way that it complies with the Nextflow repository, nf-core, standards, meaning it could be uploaded for public use.

Briefly, nf-core is a repository for Nextflow bioinformatics packages. It is tightly managed, and has stringent requirements in order to have a package considered for hosting.

The Calling Cards pipeline may be found here

https://github.com/BrentLab/callingcards

# Chapter 2: EDS1, A Novel Regulator at the Heart of the Hexose Transporter Expression Network

## Introduction

In nature, a yeast cell's external environment is chaotic. The number of competitors, their diversity, the availability of resources, and the presence of toxins change the homeostatic state of the cell, and provoke a cellular response to adapt and survive. Shifts in the yeast environment may rapidly change from nutrient poor to nutrient rich, as fruit ripens and breaks down, and as yeast population dynamics rapidly change the availability of nutrients and presence of metabolic byproducts. The speed and effectiveness of the response is under strong evolutionary selection, and the result is a highly interconnected set of pathways capable of effecting great and swift change in the transcriptome in response to the type and concentration of sugar available in the environment.

In addition to responding to external conditions, two highly conserved nutrient response pathways signal intercellular stress. One pathway uses cyclic AMP to activate Protein Kinase A (PKA) in the presence of excess glucose. The other uses AMP and ADP to activate the adenosine monophosphate-activated protein kinase (AMPK) complex when the cell's energy charge is low.

For yeast, the primary concern with sugar is how to get as much of it into the cell as possible as quickly as possible. Thus, the genome of *Saccharomyces cerevisiae* (yeast) harbors genes encoding at least 17 hexose transporters, *HXT1-11*, *HXT13-17*, and *GAL2*, which have a wide range of affinities and transport capacities for glucose, galactose, fructose, or mannose[3]. These genes are regulated by the PKA AMPK, HOG and TOR pathways as well as a yeast-specific pathway that begins with one of two external sugar sensors in the plasma membrane: Snf3p and Rgt2p. All of these pathways have been studied extensively, and the sequence-specific, DNA-binding transcription factors (TFs) on which they act to regulate *HXT* gene expression are thought to be well known, among which are Mig1p, Mig2p, and Rgt1p[45]. These factors also regulate each other as well as upstream components of the pathways, leading a network with multiple feedback and feed forward loops and complex dynamics[6]. In this paper, we reveal Eds1p to be a fourth player at the center of this densely connected network.

*EDS1* and *RGT1* exist as paralogs as a result of the whole-genome duplication that occurred an estimated 100 million years ago in the lineage leading to *S.cerevisiae*[7]. Rgt1p is an extensively studied, 1170 amino acid protein with an N terminus Zn(2)-C6 DNA-binding domain. In the absence of glucose, Rgt1p binds to and represses the expression of a number of genes including those encoding hexose transporters (HXTs), invertase (*SUC2*), and hexokinase isozyme 2 (*HXK2*), by recruiting Ssn6-Tup1, a general repressor complex. This repressive function requires Mth1p or Std1p which protects Rgt1p from hyperphosphorylation by protein kinase A (PKA). In the presence of glucose, Mth1p is degraded and *MTH1* transcription is repressed, exposing Rgt1p to hyperphosphorylation by PKA and causing it to dissociate from DNA[89]. In high glucose conditions, Rgt1p can functionally activate a subset of its target genes, including *HXT1*. Domains required for the repressive activity and recruitment of Ssn6p-Tup1p (Fig 3A, orange), inhibition of the repressive activity in high glucose (Fig 3A, red), the activation activity (Fig. 3A, green) have been mapped, as have phosphorylated residues[10]. Most of the Rgt1 protein shows significant conservation across the *Saccharomyces* species (Fig. 3A, bottom wiggle track).

Eds1p is 919 amino acid protein with an N terminus Zn(2)-C6 DNA binding domain. Although many regions of Eds1p are conserved, its overall conservation level (across similar species of yeast) is much lower than that of Rgt1, a highly conserved gene (Fig. 3A, top wiggle track). Consistent with the similarity of their DNA binding domains (DBDs; 60.7%), the DNA binding specificities of *RGT1* and *EDS1* are similar, although *RGT1* seems to be somewhat more specific (Appendix A1, Fig. 3B).

In contrast to *RGT1*, *EDS1* is a poorly studied gene which is not the focus of any previous publications. Even its name, "Expression Dependent on Slt1", is based on a paper that was never published, and existing evidence suggests that its expression is not dependent on Slt1[11]. It is non-essential, and the deletion mutant is reported to have slightly decreased competitive fitness in minimal medium (MM) with 2% glucose and increased competitive fitness in yeast peptone (YP) + 2% ethanol[12]. Several genome-wide, high-throughput data sets provide some clues to its function. In one study, it was described as being repressed by both Mig1p and Mig2p, and not showing substantial increases in expression under conditions of glucose repression unless both repressors are deleted[13]. It was subjected to binding location analysis by ChIP-chip and gene expression in a strains in which it is deleted has been assayed twice, all in high-glucose, repressing conditions[14][15][11]. Analyzing the older expression data set as well as models of Eds1p's DNA-binding specificity, Haynes et al. predicted that Eds1p directly represses most of the genes encoding proteins in the lysine biosynthesis pathway[15][16]. Indeed, the seven most significantly differentially expressed (DE) genes were all involved in lysine biosynthesis. However, a subsequently published study of expression in transcription factor (TF) knockout strains, this time in synthetic complete medium rather than yeast peptone, found that Lys20 was the only lysine-associated gene to be differentially expressed in the *eds1Δ* deletion strain, and it went slightly down rather than up in the mutant. Indeed, only one gene (*ASP3-3*) was called significantly and substantially DE by these authors, leading them to classify the Eds1 deletion strain as "non-responsive"[11].

At the RNA level, both *RGT1* and *EDS1* are more highly expressed  in growth-limiting glucose than in excess glucose, but the range of *EDS1* expression is greater than that of *RGT1*, being higher in limiting glucose and lower excess glucose (Fig. 3C). Furthermore, *EDS1* appears to be completely insensitive to the concentration of galactose, with both limiting and excess galactose being treated as intermediate between limiting glucose and excess glucose. *RGT1*, on the other hand, responds to galactose just as it does to glucose. Finally, induction of both *EDS1* and *RGT1* in limiting glucose is abolished in the *snf1Δ* mutant, consistent with previous observations that both *RGT1* and *EDS1* are repressed by Mig1p, which is inactivated by Snf1p in limiting glucose.

In this paper, we elucidate the function of Eds1p and its possible roles in the regulation of lysine metabolism and hexose transport. We started with a genome-wide study of its binding locations during growth on agarose plates with synthetic complete (SC) medium and galactose as the carbon source. We also subjected its paralog from the yeast whole-genome duplication, Rgt1p, and an authentic regulator of lysine biosynthesis genes, Lys14p, to binding location analysis in the same conditions (Table 1). The binding locations of all of three transcription factors (TFs) were also assayed two lysine conditions, with and without, in SC medium.

Rather than using the more conventional chromatin-immunoprecipitation sequencing (ChIP) method, we determined binding locations using Transposon Calling Cards, a purely nucleic-acid based method that does not involve chromatin handling or affinity purification[17]. We then studied the transcriptional response to deletion of *EDS1*, *RGT1*, *LYS14*, and two double mutants, under the same conditions.

Finally, we studied the effects of the eds1 mutation on the dynamic response when excess glucose is added to a culture growing in minimal medium (MM) in a glucose-limited chemostat. To put the role of Eds1p in the context of the known factors regulating hexose transporter expression, we also carried out parallel RNA-Seq experiments on single mutants of *RGT1*, *MIG1*, and *MIG2*, as well as double deletions and a quadruple deletion of all four genes, and *SNF1* deletion mutants, in response to addition of excess glucose.

These studies reveal Eds1p to be a novel player at the heart of hexose transporter regulation, with a both redundant and non-redundant relationship with Rgt1p.

Data

| Assay | Strain | Environment | Media | Sugar |
|---|---|---|---|---|
| TF binding | Eds1, Rgt1, Lys14 | Agarose plates | SC +/-lysine | Gal |
| Expression | WT, eds1Δ, rgt1Δ, lys14Δ, eds1Δ+rgt1Δ, eds1Δ+lys14Δ | Agarose plates | SC +/-lysine | Gal |
| Expression | WT, eds1Δ, rgt1Δ, mig1Δ, mig2Δ, snf1Δ, eds1Δ+rgt1Δ, mig1Δ+mig2Δ, mig1Δ+rgt1Δ, mig2Δ+rgt1Δ, eds1Δ+rgt1Δ+mig1Δ+mig2Δ | Glc limited chemostat | Minimal | Glc |
| Expression | WT, eds1Δ, rgt1Δ, mig1Δ, mig2Δ, snf1Δ, eds1Δ+rgt1Δ, mig1Δ+mig2Δ, mig1Δ+rgt1Δ, mig2Δ+rgt1Δ, eds1Δ+rgt1Δ+mig1Δ+mig2Δ | Excess Glc time course | Minimal | Glc |
| Expression | WT | Gal limited chemostat | Minimal | Gal |
| Expression | WT | Excess Gal | Minimal | Gal |

SC: Synthetic Complete; YPD: Yeast Extract Peptone Dextrose; Gal: galactose; Glc; glucose; WT: Wild Type; Metabolites: glucose, ethanol, glycerol

Table 1: Data used to interrogate the function of Eds1 and its relationship to Rgt1.

Samples for the Calling Cards experiments were prepared following methods and strains described in Mayhew and Mitra, 2016[17]. There are two batches in this data set, differentiated by the presence of absence of lysine in the media.

To replicate the calling cards conditions for the corresponding RNA sequencing assays, strains were initially grown on YPD. Single colonies picked in the wildtype and mutants listed in Table1. These samples were used to plate a lawn on SC media with galactose as a carbon source. Batches were determined by the presence of absence of lysine. A 3cm x 3cm square was scraped from each colony into 1.5 mL tube, and the RNA was isolated. RNA isolation was performed with Invitrogen RiboPure (Catalog number AM1926), mRNA isolation with NEBNext Poly(A) mRNA Magnetic Isolation module NEB E7420L, and library preparation was performed with NEBNext Ultra Directional RNA Library Prep Kit for Illumina NEB E7420L. Samples were sequences at MGI.

For the chemostat samples, cells were grown in liquid minimal media with a constant flow rate of media and atmosphere. Cells were sampled, and then 2% glucose is added to the media and the flow is halted. Cells were subsequently sampled at 45, 60, 90, 180, 300 and 1440 minutes, though the mutant samples were typically not healthy enough to perform RNA sequencing at 1440 minutes.

The RNA sequencing reads were processed with the nf-co.re RNAseq pipeline[18]. Briefly, alignment to the sacCer3 genome was performed using STAR. Quantification was performed using Salmon and the S288C-R64-3-1 genome annotations. Extensive QC was performed (see nf-co.re/rnaseq for details), and additional metrics concerning the coverage and expression of the knocked out loci in each mutant sample were performed. Samples with more than 25% coverage and greater than 2 log2 normalized counts were excluded, as were samples with less than a million protein coding reads.

Differential expression was performed using DESeq2, with effect sizes shrunk using the ashr package, while the calling cards data was analyzed using custom scripts[19][20]. Figures 4,5,6 and 8 use the DESeq2 variance stabilization transformation to describe counts data.

Calling Cards data is analyzed using hypergeometric p-values, which are proxies for binding strength of given TFs. We use a threshold of .001 for significance to filter for significantly bound genes.

## Results

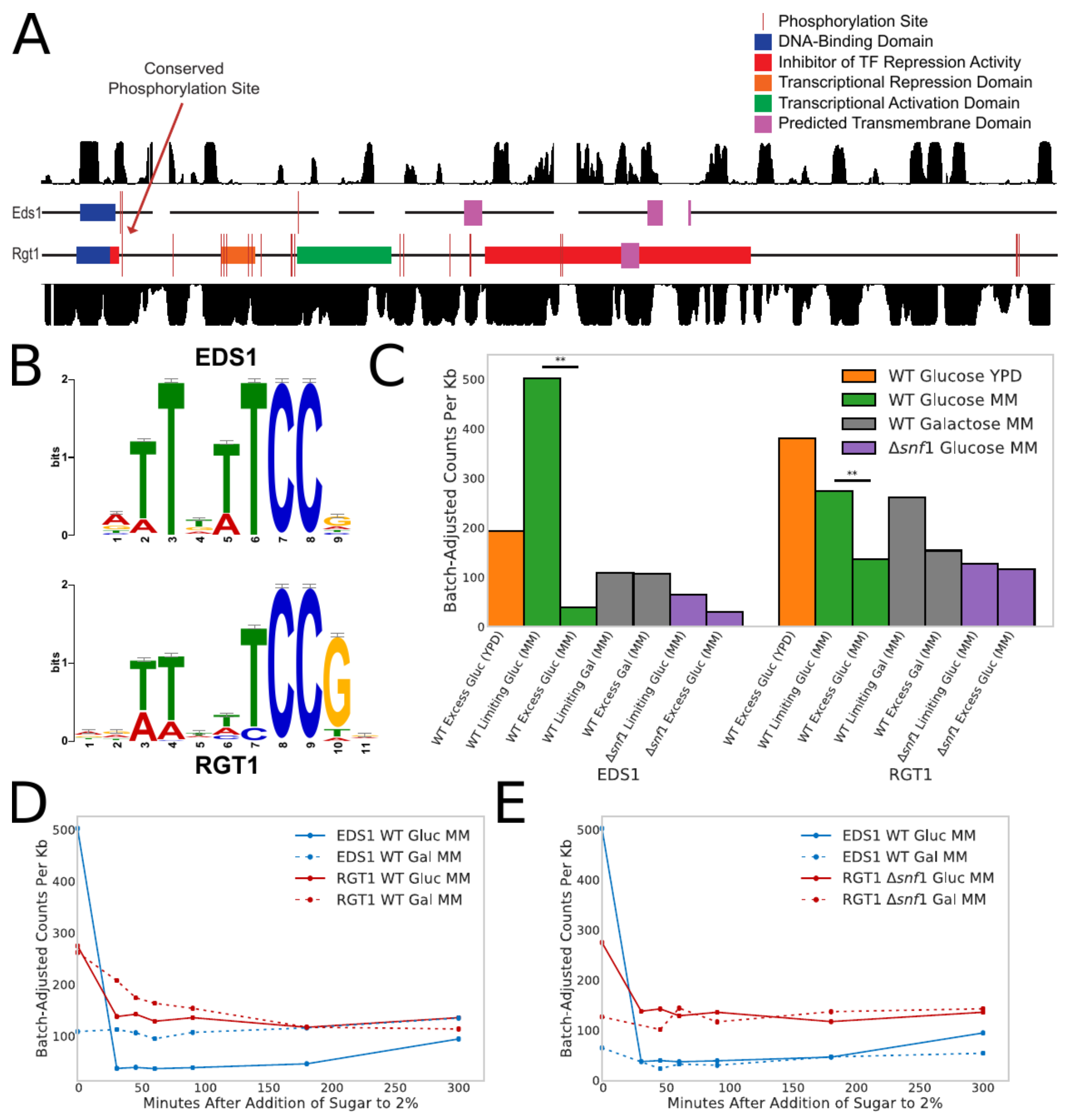

Figure 3: Eds1 and Rgt1 Sequence and Expression Overview.

(A) Protein alignment, of Eds1 (top) and Rgt1 (below). Colored boxes on the alignments correspond to known functional domains while vertical red lines signify phosphorylation sites. Above and below the alignment, phastCon conservation scores show conservation across the nucleotide sequences of each locus. (B) Binding Sequence motif for Eds1p and Rgt1p. (C) plots EDS1 expression (left set) and RGT1 expression (right set) in excess and limited glucose, galactose, and in both conditions in the snf1Δ mutant. (D) and (E) show Eds1 and Rgt1 expression in glucose and galactose (D) and in the snf1Δ mutant (E). In (E), solid lines represent wild type expression, dotted lines snf1Δ mutant expression.

Figure credit: Mike Toomey

**Eds1p differs from Rgt1p at a key serine which likely affect interaction with Mth1p**. S88 is a highly conserved serine in the Rgt1p sequence. This serine residue is critical to the effect of Rgt1p on *HXT1,* which is mediated by the Rgt1p and Mth1p interaction[21][2]. In limited glucose, Mth1p prevents Rgt1p from being phosphorylated by the PKA pathway. Hypophosphorylated Rgt1p interacts with Ssn6p-Tup1p repressor complex and binds to the promoter of *HXT1*, thus repressing *HXT1* transcription in low glucose. In high glucose, Mth1p is ubiquitinated and degraded, which diminishes Mth1p concentration and consequently allows Rgt1p to become phosphorylated. This disrupts the Rgt1p interaction with Ssn6p-Tup1p, which de-represses *HXT1*[22]. As Rgt1p becomes hyperphosphorylated, it becomes a direct activator of *HXT1* expression[6].

Eds1p, however, does not have a serine which aligns to the serine at position 88 in the Rgt1p sequence. Rgt1p S88 aligns to an alanine in the Eds1p sequence, and is otherwise in a gapped and less conserved local area. The flanking region is critical for both Rgt1p's ability to repress *HXT1* in limited glucose, and induce *HXT1* expression in high glucose , as shown by Polish, et al. through deletion of the region 80-90 in Rgt1p . Interestingly, they also show that mutating S88 to an alanine in Rgt1p disrupts the interaction between Rgt1p and Mth1p.

We show that Eds1p is a constitutive inducer of *HXT1* expression in both limited and high glucose. We speculate that this is due in part to the absent serine near Eds1p's DNA binding domain.

**Eds1p has fewer direct targets, but more indirect targets, than Rgt1p in SC media with galactose.** After filtering out gene features with low expression across all samples in our data, we consider a background gene set of 6,457 genes. In SC media with galactose and lysine, there are 545 more genes differentially expressed by at least 1 fold (padj < .05) in the *eds1Δ* mutant compared to the *rgt1Δ* mutant. In the same conditions, but without lysine in the media, there are 943 more differentially expression genes in the *eds1Δ* mutant compared to the *rgt1Δ* mutant. In limited glucose, there are a similar number of differentially expressed genes in *eds1Δ* and *rgt1Δ* mutants.

**Eds1p has a large effect on the transcriptome in the initial response to high glucose, even while being down regulated.** Unexpectedly, given that *EDS1* expression decreases significantly within 45 minutes in high external glucose, there are between 500 and 600 more DE genes in the *eds1Δ* mutant compared to the *rgt1Δ* mutant at time points 45 minutes and 90 minutes. After 180 minutes in high glucose, the *RGT1* deletion has a larger effect on the transcriptome. The DE genes in the *eds1Δ* mutant are significantly enriched for function in the TCA cycle, while the DE genes in the *rgt1Δ* mutants are not, indicating that Eds1p has a greater role in regulating aerobic respiration in both low and high glucose than Rgt1p.

## Eds1p and Rgt1p Regulation of Hexose Transporters 1 through 7

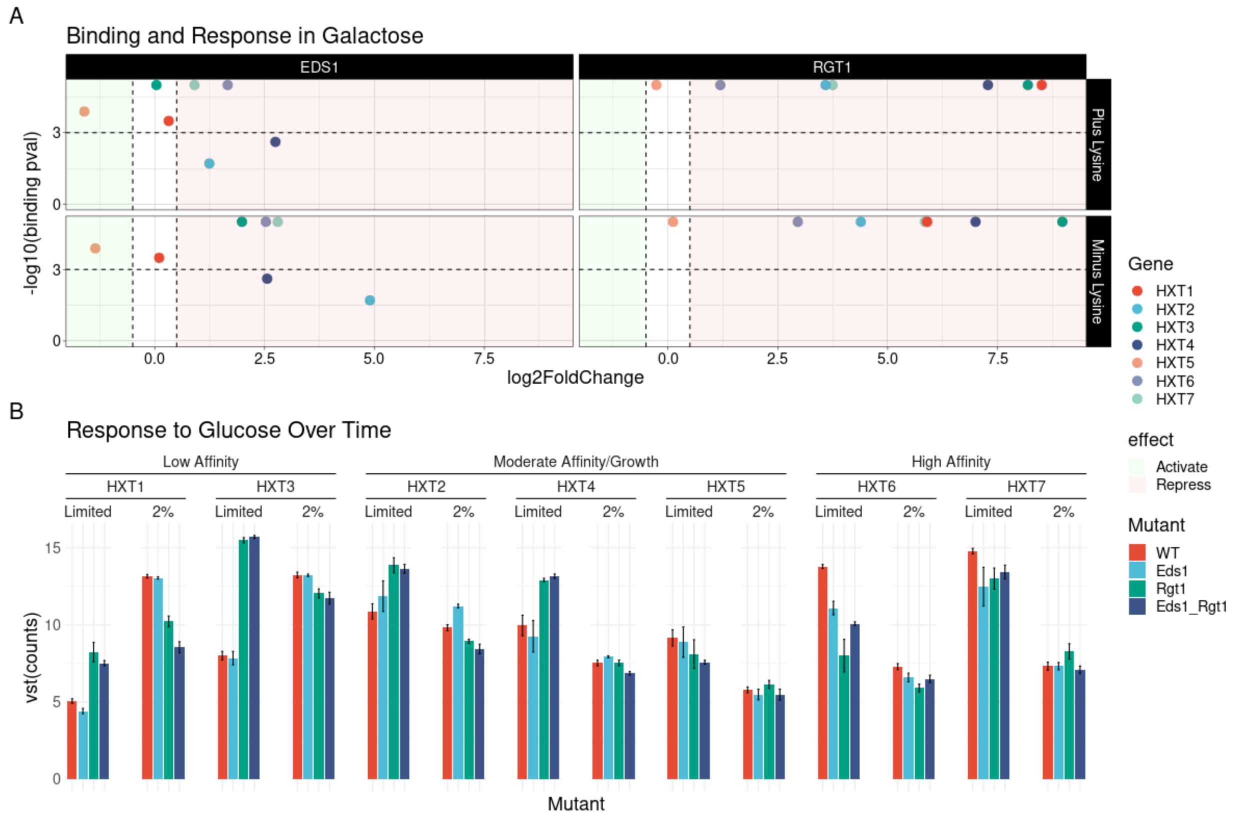

Figure 4: Eds1p and Rgt1p Regulation of Hexose Transporters 1 through 7.

(A) Gene expression of hexose transporters (HXT1-7) in eds*1Δ* mutant(left), *rgt1Δ* mutant (right). The x-axis represents shrunken log2FoldChange in the mutant compared to the wildtype expression. The y-axis represents the calling cards hypergeometric p-value, on a -log10 scale. The two vertical dotted lines are at +/- .5 log2foldChange, outside of which we consider the change to be significant, in conjunction with adjusted p-values less than .05. The horizontal dotted line is at .001, or 3 on the -log10 scale. Points above that line are considered significantly bound. Assay conditions for are SC media with galactose as a carbon source. Rows represent the plus lysine condition, and minus lysine condition. (B) Dynamic expression of HXT1-7 in response to glucose in wildtype (WT) and e*ds1Δ*, *rgt1Δ* and *eds1Δrgt1Δ* mutants. *HXT1-7* are grouped by their glucose transport affinity.

*S.cerevisiae* regulates the intake of glucose, its preferred sugar source, and other hexose sugars, through 17 highly similar hexose transporters. Hxt1-7 are both the most functionally prominent, and the most studied. The hexose transporters may be classified by their affinity to a given substrate. In relation to glucose, Hxt1p and Hxt3p are considered low affinity glucose transporters ($K_{m(glucose)}$ ~ 100 mM).

*HXT1* and *HXT3* expression is regulated in response to external glucose concentration, and internal metabolic cues. *HXT2*, *4* and *5* are moderate affinity glucose transporters ($K_{m(glucose)}$ ~10 mM). *HXT2* and *HXT4* respond to both glucose concentration and growth cues. *HXT5* responds to growth cues and is otherwise unresponsive to external glucose concentration. *HXT6* and *HXT7* are high affinity glucose transporters ($K_{m(glucose)}$ ~ 1 mM), and are both regulated by external glucose concentration[4].

*S.cerevisiae* adjusts the balance of hexose transporter expression in response to external glucose through at least two external sensor networks: the Snf3p/Rgt2p hexose sensors, and the cAMP/PKA membrane sensor Gpr1p[5]. Here, we focus on the Snf3p/Rgt2p pathway.

Snf3p and Rgt2p respond to increasing external glucose by emitting a signal which results in the phosphorylation and subsequent ubiquitination and degradation of Mth1p and Std1p. Decreased concentrations of Mth1p and Std1p in the nucleus allows PKA pathway to phosphorylate Rgt1p. This disrupts the interaction between Rgt1p and the Ssn6p-Tup1p repressor complex, thereby relieving the Rgt1p mediated repression of *HXT1* and *HXT3*. Phosphorylated Rgt1p in high glucose becomes an activator of *STD1*, *MTH1* and *MIG2* expression. This forms a feedback loop in which the Snf3p/Rgt2p pathway signal initially de-represses *HXT1* and *HXT3*, but also contributes to expression of the genes which will eventually re-establish the repression of *HXT1* and *HXT3* when the external glucose is exhausted[6].

**Eds1p directly activates *HXT1* in limited and 2% glucose.** The calling cards data shows that Eds1 and Rgt1 both bind the HXT1 promoter (Fig. 4A). Rgt1p represses *HXT1* in limited glucose and activates *HXT1* expression in high glucose. Rgt1p's biphasic behavior consistent with previously published literature, a result which our data confirms[6].

In limited glucose, *HXT1* expression is reduced in the *eds1Δ* mutant by .68 fold (padj < .05) over the wildtype expression (Fig 4B). We also note that the expression of *HXT1* trends greater in the *rgt1Δ* mutant than it is in the *eds1Δrgt1Δ* mutant, though the difference is not statistically significant.

In 2% glucose, *HXT1* expression is unaffected in the *eds1Δ* mutant. Knocking out *RGT1*, by contrast, suppresses *HXT1* induction, though notably not entirely as the expression of *HXT1* is still higher in the *rgt1Δ* background than it is in the wildtype in limited glucose. This is consistent with the observation that Rgt1p is an activator of *HXT1* in high glucose. However, the difference between *HXT1* expression in the *rgt1Δ* mutant and the *eds1Δrgt1Δ* is -3.45 fold (padj < .05, Fig 2B). This leads us to hypothesize that there exists a combinatorial activation effect of Eds1p and Rgt1p on Hxt1 in high glucose, but that Rgt1p alone is sufficient to induce *HXT1* to the level observed in the wildtype.

Overall, this strengthens the hypothesis that the interaction between Eds1p and Mth1p is different from the interaction between Rgt1p and Mth1p. Mth1p mediates the Rgt1p phosphorylation state, and thus whether Rgt1p acts as a repressor or an activator. If Mth1p is not able to block Eds1p phosphorylation, then given the evidence that Eds1p is bound to the promoter of *HXT1*, we would expect Eds1p to act as a constitutive activator, which consistent with our observations.

**Eds1p binds, but does not affect, *HXT3* in limited or high glucose**. Eds1 appears to bind *HXT3* in galactose in both lysine conditions. In media without lysine, Eds1p represses *HXT3*. However, in minimal media, limited glucose, and in minimal media, high glucose, *HXT3* is unaffected in the *eds1Δ* mutant.

**Eds1p may not directly regulate *HXT2* and *HXT4***. *HXT2* and *HXT4* encode moderate affinity glucose transporters which are regulated by both glucose and growth cues. Our data suggests that Rgt1p directly contributes to the repression of *HXT2* and *HXT4* in limited glucose, while in high glucose, *HXT2* and *HXT4* are repressed by Mig1p and Mig2p[13]. The binding signal between Eds1p and

*HXT2* and *HXT4* is comparatively weak and below our significance thresholds, suggesting that Eds1p does not bind *HXT2*, *HXT4* (Fig 2).

**Eds1p, and not Rgt1p, may directly regulate *HXT5***. *HXT5* expression, unlike the other *HXT1-7* genes, is shown to respond only to growth cues[23]. Eds1p and Rgt1p both bind to the *HXT5* promoter (Fig 2A). The strongest effect on *HXT5* expression, however, is in the *eds1Δ* mutant in SC with galactose where Eds1p acts as an activator. This leads us to hypothesize that *HXT5* is a direct and functional target of Eds1p, but possibly only under certain nutrient limitations which impacts growth. It is also possible that Eds1p exerts more control over *HXT5* expression, generally, than Rgt1p.

**Both Eds1p and Rgt1p directly repress *HXT6* and *HXT7* in limited glucose, and in galactose**. In limited glucose, and in both lysine conditions in on SC media with galactose, Eds1p and Rgt1p repress *HXT6* and *HXT7*. In high glucose, the effect of Eds1p and Rgt1p on both loci is minimal. *HXT6* and *HXT7* are strongly repressed by Mig1p and Mig2p in high glucose, thus the minimal effect in the *eds1Δ* or *rgt1Δ* is expected[13].

Eds1p and Rgt1p Regulation of the Hexose Transporter Network Regulators

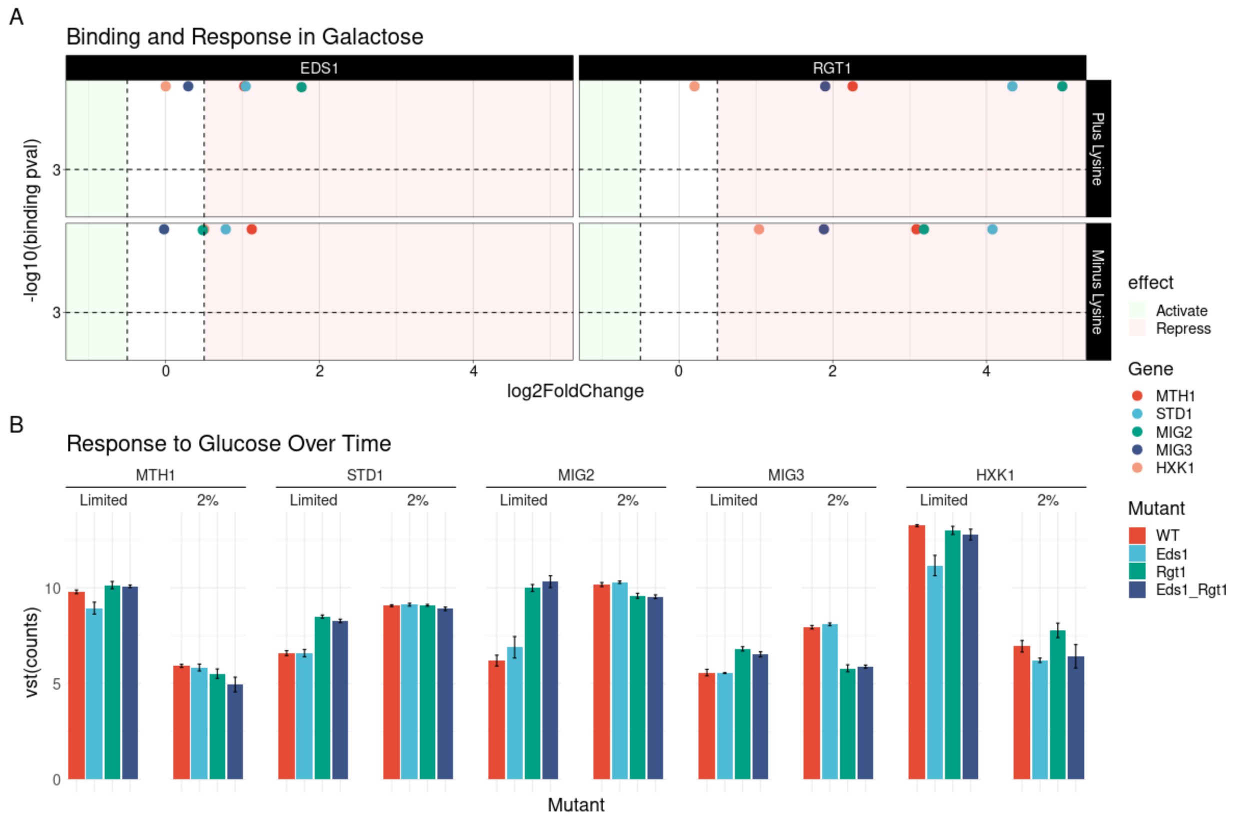

Figure 5: Regulation of known regulators of Rgt1 by Rgt1 and Eds1.

(A) Eds1p and Rgt1p binding data for known *RGT1* regulators *MTH1*, *STD1*, *MIG2*, *MIG3*, *HXK1*. The x-axis represents shrunken log2FoldChange in the mutant compared to the wildtype expression. The y-axis represents the calling cards hypergeometric p-value, on a -log10 scale. The two vertical dotted lines are at +/- .5 log2foldChange, outside of which we consider the change to be significant, in conjunction with adjusted p-values less than .05. The horizontal dotted line is at .001, or 3 on the -log10 scale. Points above that line are considered significantly bound. Assay conditions for are SC media with galactose as a carbon source. Rows represent the plus lysine condition, and minus lysine condition. (B) Dynamic expression of the same set of genes in response to glucose in wildtype (WT) and e*ds1Δ*, *rgt1Δ* and *eds1Δrgt1Δ* mutants.

**Eds1p and Rgt1p bind and regulate *MTH1*, *STD1*, *MIG2*, *MIG3* and *HXK1* under certain conditions**. Mth1p and Std1p regulate Rgt1p phosphorylation, and thus modulate Rgt1p's function as a repressor or activator. In high glucose, Mth1p and Std1p are ubiquitinated and degraded, allowing Rgt1p to be phosphorylated by the PKA pathway. This relieves Rgt1p mediated repression, and eventually leads to Rgt1p becoming hyperphosphorylated, at which point Rgt1p acts as an activator of certain targets, including *MTH1*, *STD1* and *MIG2.* Thus, Rgt1p activates *MTH1*, *STD1*, *MIG2* expression in high glucose[69]. Rgt1p continues to represses *HXK2* expression in high glucose[3]. We are not aware of data specifically on the relationship between Rgt1p and *HXK1*, which is a paralog of *HXK2* and is thought to function similarly.

We sought to understand if these known regulators, and targets, of Rgt1 were also regulated by Eds1 (Fig 3A). In our data, we observe that in galactose, both Eds1p and Rgt1p bind the promoters, and with the exception of Eds1p and *MIG3*, significantly repress expression of these genes in one or both lysine conditions.

Eds1p activity seems to depend heavily on environmental conditions. In glucose, Eds1p does not affect the expression of *MTH1*, *STD1*, or *MIG3* expression. In galactose, Eds1p does affect *MTH1* and *STD1*, but not *MIG3,* expression. Rgt1p also does not significantly affect *MTH1* expression in limited or 2% glucose, but Rgt1p is a strong repressor of *MTH1* in SC media with galactose. Rgt1p significantly represses *STD1* in limited glucose, and in the galactose conditions, but does not affect *STD1* expression in 2% glucose. Rgt1p significantly represses *MIG2* in limited glucose and galactose, and has a small, non-significant activation effect in 2% glucose. Rgt1p suppresses *MIG3* in limited glucose and galactose, and significantly activates *MIG3* in 2% glucose, which is in stark contrast to Eds1p, which does not affect *MIG3* in any of the assay conditions.

Eds1p represses *HXK1* in SC media with galactose without lysine, and in limited glucose. Rgt1p also represses *HXK1* in SC media with galactose without lysine, but does not significantly affect *HXK1* expression in limited glucose. The effect on *HXK1* in neither the *eds1Δ* mutant or *rgt1Δ* mutant is significant at 45 minutes in 2% glucose, but from 45 minutes to 300 minutes in glucose, Eds1p activates *HXK1* expression, while Rgt1p represses it, with effects greater than -1.5 fold change in the *eds1Δ* mutant over the wildtype (padj < .05). A similar effect is present in the *rgt1Δ* mutant on *HXK1*, except that *rgt1Δ* acts as an activator.

This result suggests that both Eds1p and Rgt1p do regulate the known regulators of Rgt1p. However, their effects on these target genes may not be strongly driven by the glucose signal alone, and Eds1p generally has less effect than Rgt1p.

Regulation of Eds1 by Snf1, Mig1 and Mig2

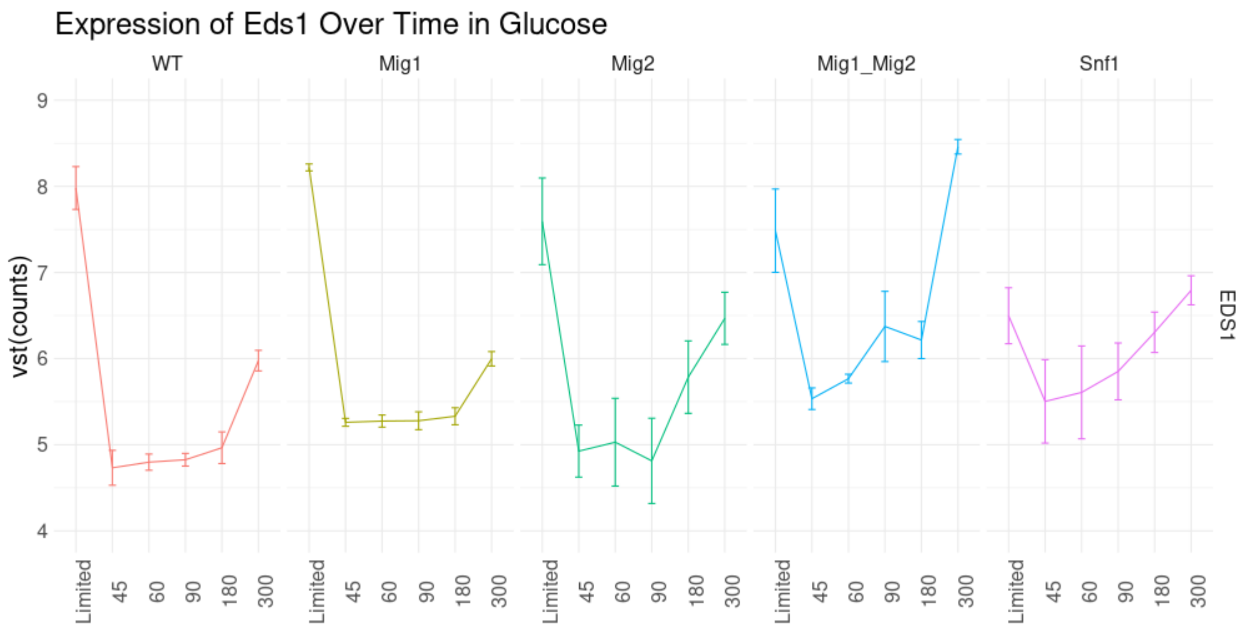

Figure 6: Regulation of Eds1 by Snf1, Mig1 and Mig2

Each panel represents the expression of Eds1 in a different genetic background, (wt, mig1, mig2, mig1_mig2, snf1, left to right). The first time point, noted "Limited" represents expression in minimal media, limited glucose, directly before 2% glucose is added to the media.

Snf1p is a serine/threonine protein kinase which responds to both external glucose signals, through the cAMP/PKA pathway, and internal metabolic signals through the Reg1p-Glc7p regulatory complex. Snf1p is activated in low glucose conditions, which allows Snf1p to phosphorylate, among other targets, Mig1p and Mig2p. Phosphorylated Mig1p and Mig2p are exported from the nucleus, which thus de-represses genes such as *HXT2*, *HXT4*, *HXT6* and *HXT7* which are necessary in low glucose conditions[4].

**Snf1p regulates *EDS1* expression**. In the transition from limited glucose to high glucose, *EDS1* expression dramatically drops (Fig 4). However, in the *snf1Δ* mutant, the regulation by glucose of *EDS1* expression is highly attenuated (Fig 1E and Fig4). This suggests that both induction of *EDS1* expression in limited glucose, and repression of *EDS1* expression in high glucose, depends on Snf1p.

**Snf1p affects *EDS1* expression through *MIG1* and *MIG2***. In 2% glucose, Snf1p exists in a hypophosphorylated state. Hypophosphorylated Snf1p does not phosphorylate Mig1p and Mig2p, which as a result concentrate in the nucleus, where they act as repressors of many genes which are up regulated in response to high glucose. Mig1p and Mig2p are previously shown to jointly directly regulate *EDS1* expression, which our results confirm[13].

There is no significant effect on *EDS1* expression in the *mig1Δ* mutant (Fig 4). In the *mig2Δ* mutant, *MIG1* expression is upregulated compared to the wildtype, and we observe that *EDS1* expression is reduced compared to the wildtype. Additionally, in the *mig2Δ* mutant, *EDS1* expression levels begin climbing sooner than in the wildtype. In the *mig1Δmig2Δ* double deletion, we observe a statistically significant difference between *mig1Δmig2Δ* and *mig2Δ* at time points 90 and 300, and significant difference in expression of Eds1 in the *mig1Δmig2Δ* mutant compared to the wild type at every time point. We use this evidence to corroborate previous work showing the direct regulatory relationship between Mig1p and Mig2 and *EDS1* expression, and suggest that this accounts for at least part of the effect of Snf1p on *EDS1*.

## Eds1 Regulation of Lysine Biosynthesis

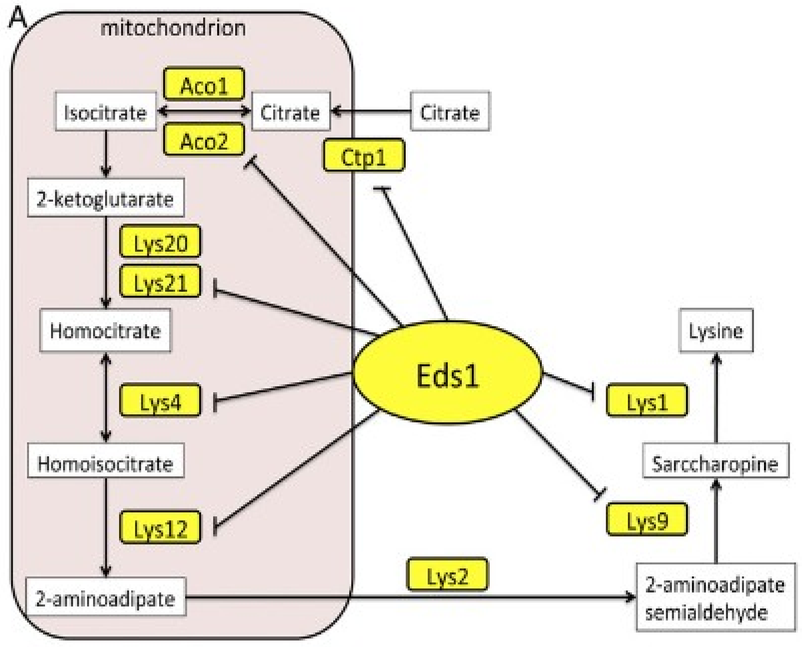

Figure 7: Lysine biosynthesis pathway, and the predicted effect of Eds1, from Haynes et al. 2013

Previously, we reported that Netprophet 1.0, a transcription factor network inference algorithm which uses differential expression to infer functional and direct targets of transcription factors, found that the inferred targets of Eds1p are enriched for genes in the lysine biosynthesis pathway[24]. Lysine biosynthesis branches from the TCA cycle, and we hypothesized that Eds1p links activity of the lysine biosynthesis pathway to glucose availability, and the metabolic state of the cell[25]. We used our data to test the relation of Eds1 to known lysine biosynthesis pathways.

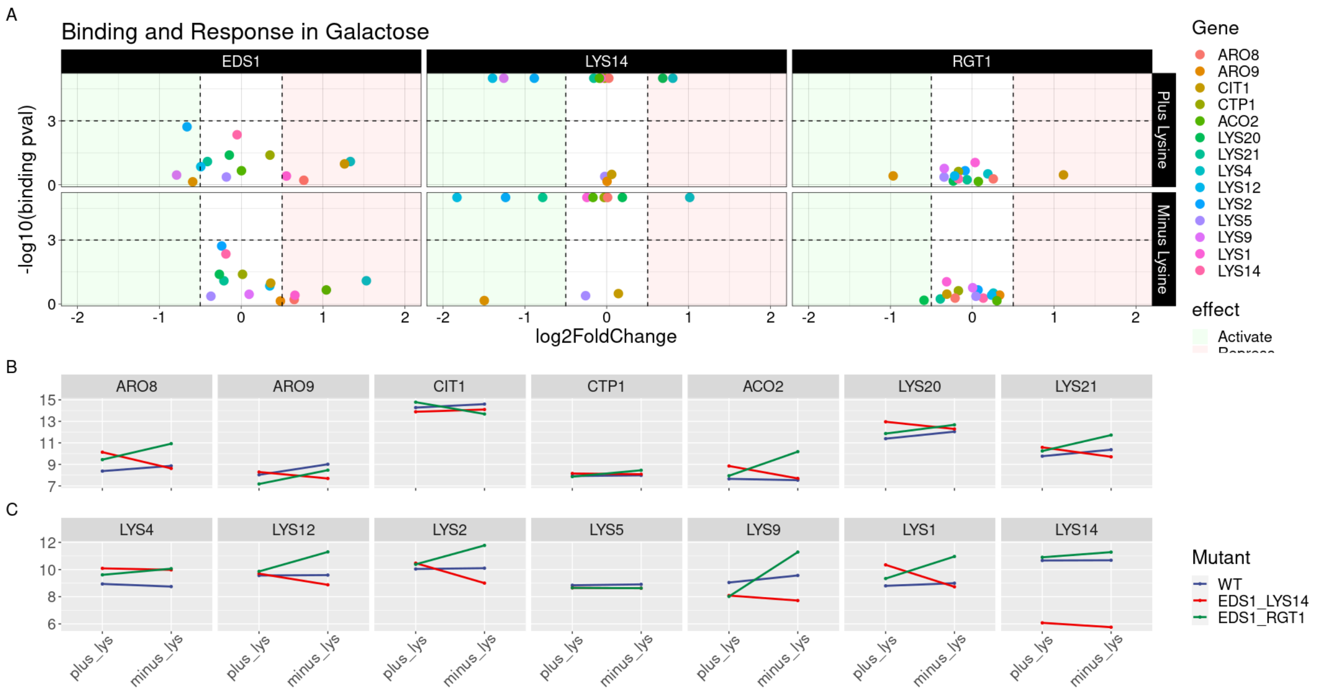

Figure 8: Eds1 Regulation of Lysine Biosynthesis.

(A) Eds1p and Rgt1p binding data for the lysine biosynthesis pathway genes, including those which precede the lysine biosynthesis pathway in the TCA cycle, *ARO8*, *ARO9*, *CIT1* and *CTP1*. The x-axis represents shrunken log2FoldChange in the mutant compared to the wildtype expression. The y-axis represents the calling cards hypergeometric p-value, on a -log10 scale. The two vertical dotted lines are at +/- .5 log2foldChange, outside of which we consider the change to be significant, in conjunction with adjusted p-values less than .05. The horizontal dotted line is at .001, or 3 on the -log10 scale. Points above that line are considered significantly bound. Assay conditions for are SC media with galactose as a carbon source. Rows represent the plus lysine condition, and minus lysine condition. (B) The expression of each gene in plus (left on each plot) and minus (right on each plot) gene from panel A in the wildtype, *eds1Δlys14Δ* and *eds1Δrgt1Δ* mutants. The line connecting each point describes the effect of lysine on expression of the given gene in a given mutant.

**Eds1p primarily acts as a repressor of the lysine biosynthesis pathway in media when lysine is present. The effect is, however, indirect.** The binding signal between Eds1p and the lysine biosynthesis pathway genes is generally not above our threshold of significance, though it is notable that Lys2 and Lys14 are near the threshold. By comparison, none of the lysine biosynthesis pathway genes are near the significance threshold for Rgt1p. We verify our previous results that Eds1p activity represses *CTP1*, *ACO2*, *LYS4*, *LYS12* and *LYS9*, but we show here that this effect is indirect.

**Deleting *EDS1* in either the *rgt1*Δ or *lys14*Δ background significantly disregulates lysine biosynthesis gene expression**. The effect of lysine on either double mutant is significantly stronger than the effect of lysine on the WT, or on the single mutant strains. This points to an important, but indirect, role for both Eds1p and Rgt1p in regulating lysine biosynthesis. In conjunction with the significant differential expression that results in the *eds1Δ* mutant, this adds to the evidence that Eds1p has a strong, though indirect, role on lysine biosynthesis.

**Eds1p regulates genes important to the TCA cycle**. Lysine biosynthesis branches from the TCA cycle at the 2-oxyglutarate step. In the *eds1Δ* mutant genes encoding enzymes which catalyze reactions leading up to the formation of 2-oxyglutarate, are differentially expressed. Though these appear to be mostly indirect effects, this points to a possible explanation of the relationship between Eds1 and lysine biosynthesis.

## Discussion

Currently, 547 paralogous genes, the result of the whole genome duplication event, remain functional in the *S.cerevisiae* genome[26]. A leading theory is that these genes are important to regulating the Crabtree effect, which describes the preference of *S.cerevisiae* for aerobic fermentation in the presence of high glucose[27]. It has been theorized that *S.cerevisiae* benefits from aerobic fermentation in high glucose by quickly filling its environment with ethanol, a fermentation byproduct, which may be toxic to its competitors. Another theory is that, while the environment provides a plentiful energy source which the cell can utilize quickly and cheaply, reducing TCA cycle activity allows *S.cerevisiae* to redirect its resources to reproduction through mitosis[28].

Rgt1p, as a regulator of the hexose transporters, is a target of pathways which respond to both external glucose concentration, and internal metabolic signals. Thus we expect to find evidence that

Rgt1p, and its paralog, Eds1p, interact with both the fermentative and respiratory metabolism pathways, which also affect growth.

Complementary lines of research support the hypothesis that Eds1p and Rgt1p interact with both metabolism and growth pathways. Genetic interaction profiling is a method which seeks to discover the relationship between one genomic locus on another[5]. *RGT1* has significant negative genetic interaction with genes associated with mitochondrial function. *EDS1*, conversely, has a slightly positive interaction with the mitochondrial genes (Appendix A2) [30]. A negative genetic interaction signifies that the gene in question is not in the direct functional pathway of the gene(s) with which it negatively interacts, while a positive interaction suggests that the genes are in the same functional pathway. This, therefore, suggests that *EDS1* and *RGT1* significantly differ in their effect on aerobic respiration.

Our data shows that Eds1p both binds and affects many of the same targets as Rgt1p. However, over time in glucose, even while *EDS1* expression drops substantially, the *eds1Δ* mutant generally has hundreds more differentially expressed genes than the *rgt1Δ* mutant. Furthermore, in limited glucose, and at each time point after adding glucose, the genes differentially expressed in the *eds1Δ* mutant are significantly enriched for TCA cycle genes, whereas the genes differentially expressed in the *rgt1Δ* mutant are not. In galactose, the genes which Eds1p both binds and induces are all functional in meiosis and DNA repair, which suggests a role in sporulation, a growth state which occurs in nutrient deprivation. Finally, we also note evidence (Appendix A3) that Eds1 is regulated by other targets of Snf1p which respond to both nutrient deprivation and nutrient deprivation induced growth cues, some of which affect sporulation and filamentous growth.

Overall, evidence shows that Eds1p is a significant regulator at the heart of the hexose transporter regulation network, though its role in this network is secondary to Rgt1p. *EDS1* expression reacts to external nutrient conditions and impacts the TCA cycle, amino acid biosynthesis, and the cell cycle differently than Rgt1p.

## References

1. Lawrence, M. & Morgan, M. Scalable Genomics with R and Bioconductor. *Stat. Sci.* **29**, 214–226 (2014).

2. DI Tommaso, P. *et al.* Nextflow enables reproducible computational workflows. *Nat. Biotechnol.* **35**, 316–319 (2017).

3. Reifenberger, E., Boles, E. & Ciriacy, M. Kinetic characterization of individual hexose transporters of Saccharomyces cerevisiae and their relation to the triggering mechanisms of glucose repression. *Eur. J. Biochem.* **245**, 324–333 (1997).

4. Horák, J. Regulations of sugar transporters: insights from yeast. *Curr. Genet. 2013 591* **59**, 1–31 (2013).

5. Brink, D. P., Borgström, C., Persson, V. C., Osiro, K. O. & Gorwa-Grauslund, M. F. D-Xylose Sensing in Saccharomyces cerevisiae: Insights from D-Glucose Signaling and Native D-Xylose Utilizers. *Int. J. Mol. Sci.* **22**, (2021).

6. Kuttykrishnan, S., Sabina, J., Langton, L. L., Johnston, M. & Brent, M. R. A quantitative model of glucose signaling in yeast reveals an incoherent feed forward loop leading to a specific, transient pulse of transcription. *Proc. Natl. Acad. Sci.* **107**, 16743–16748 (2010).

7. Dashko, S., Zhou, N., Compagno, C. & Piškur, J. Why, when, and how did yeast evolve alcoholic fermentation? *FEMS Yeast Res.* **14**, 826 (2014).

8. Gancedo, J. M., Flores, C. L. & Gancedo, C. The repressor Rgt1 and the cAMP-dependent protein kinases control the expression of the SUC2 gene in Saccharomyces cerevisiae. *Biochim. Biophys. Acta* **1850**, 1362–1367 (2015).

9. Palomino, A., Herrero, P. & Moreno, F. Tpk3 and Snf1 protein kinases regulate Rgt1 association with Saccharomyces cerevisiae HXK2 promoter. *Nucleic Acids Res.* **34**, 1427–1438 (2006).

10. Polish, J. A., Kim, J. H. & Johnston, M. How the Rgt1 Transcription Factor of Saccharomyces cerevisiae Is Regulated by Glucose. *Genetics* **169**, 583 (2005).

11. Kemmeren, P. *et al.* Large-scale genetic perturbations reveal regulatory networks and an abundance of gene-specific repressors. *Cell* **157**, 740–752 (2014).

12. Qian, W., Ma, D., Xiao, C., Wang, Z. & Zhang, J. The Genomic Landscape and Evolutionary Resolution of Antagonistic Pleiotropy in Yeast. *Cell Rep.* **2**, 1399–1410 (2012).

13. Westholm, J. O. *et al.* Combinatorial control of gene expression by the three yeast repressors Mig1, Mig2 and Mig3. *BMC Genomics* **9**, 1–15 (2008).

14. Harbison, C. T. *et al.* Transcriptional regulatory code of a eukaryotic genome. *Nature* **431**, 99–104 (2004).

15. Hu, Z., Killion, P. J. & Iyer, V. R. Genetic reconstruction of a functional transcriptional regulatory network. *Nat. Genet. 2007 395* **39**, 683–687 (2007).

16. Haynes, B. C. *et al.* Mapping functional transcription factor networks from gene expression data. *Genome Res.* **23**, 1319–1328 (2013).

17. Mayhew, D. & Mitra, R. D. Transposon Calling Cards. *Cold Spring Harb. Protoc.* **2016**, 101–103 (2016).

18. Ewels, P. A. *et al.* The nf-core framework for community-curated bioinformatics pipelines. *Nat. Biotechnol. 2020 383* **38**, 276–278 (2020).

19. Love, M. I., Huber, W. & Anders, S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol.* **15**, (2014).

20. Stephens, M. False discovery rates: a new deal. *Biostatistics* **18**, 275–294 (2017).

21. Polish, J. A., Kim, J.-H. & Johnston, M. How the Rgt1 Transcription Factor of Saccharomyces cerevisiae Is Regulated by Glucose. (2005) doi:10.1534/genetics.104.034512.

22. Roy, A., Shin, Y. J., Cho, K. H. & Kim, J. H. Mth1 regulates the interaction between the Rgt1 repressor and the Ssn6-Tup1 corepressor complex by modulating PKA-dependent phosphorylation of Rgt1. *Mol. Biol. Cell* **24**, 1493–1503 (2013).

23. Verwaal, R. *et al.* HXT5 expression is determined by growth rates in Saccharomyces cerevisiae. *Yeast* **19**, 1029–1038 (2002).

24. Haynes, B. C. *et al.* Mapping functional transcription factor networks from gene expression data. (2006) doi:10.1101/gr.150904.112.

25. A, F., E, D., F, R. & A, P. Repression of the genes for lysine biosynthesis in Saccharomyces cerevisiae is caused by limitation of Lys14-dependent transcriptional activation. *Mol. Cell. Biol.* **14**, 6411–6418 (1994).

26. Byrne, K. P. & Wolfe, K. H. The Yeast Gene Order Browser: combining curated homology and syntenic context reveals gene fate in polyploid species. *Genome Res.* **15**, 1456–1461 (2005).

27. Merico, A., Sulo, P., Piškur, J. & Compagno, C. Fermentative lifestyle in yeasts belonging to the Saccharomyces complex. *FEBS J.* **274**, 976–989 (2007).

28. Pfeiffer, T. & Morley, A. An evolutionary perspective on the Crabtree effect. *Front. Mol. Biosci.* **1**, (2014).

29. Kuzmin, E. *et al.* Exploring whole-genome duplicate gene retention with complex genetic interaction analysis. *Science (80-. ).* **368**, (2020).

30. Usaj, M. *et al.* TheCellMap.org: A web-accessible database for visualizing and mining the global yeast genetic interaction network. *G3 Genes, Genomes, Genet.* **7**, 1539–1549 (2017).

# Appendix A: Eds1

## A1: Eds1/Rgt1 DNA Binding Domain Alignment

>Eds1 DNA Binding Domain (pfam), position 54 - 95
HACDQCRRKRIKCRFDKHTGVCQGCLEVGEKCQFIRVPLKRG
>Rgt1 DNA Binding Domain (pfam), posiitons 47 - 76
CDQCRKKKIKCDYKDEKGVCSNCQRNGDRC

```
########################################
# Program: water
# Rundate: Sat 16 Apr 2022 18:09:59
# Commandline: water
#    -auto
#    -stdout
#    -asequence emboss_water-I20220416-180958-0806-87210608-p1m.asequence
#    -bsequence emboss_water-I20220416-180958-0806-87210608-p1m.bsequence
#    -datafile EBLOSUM62
#    -gapopen 10.0
#    -gapextend 0.5
#    -aformat3 pair
#    -sprotein1
#    -sprotein2
# Align_format: pair
# Report_file: stdout
########################################
#=======================================
#
# Aligned_sequences: 2
# 1: Rgt1
# 2: Eds1
# Matrix: EBLOSUM62
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 30
# Identity:      15/30 (50.0%)
# Similarity:    20/30 (66.7%)
# Gaps:           0/30 ( 0.0%)
# Score: 101.0
#
#
#=======================================

Rgt1               1 CDQCRKKKIKCDYKDEKGVCSNCQRNGDRC     30
                     |||||:|:|||.:....|||..|...|::|
Eds1               3 CDQCRRKRIKCRFDKHTGVCQGCLEVGEKC     32
```

A2: Eds1 and Rgt1 Genetic Interaction

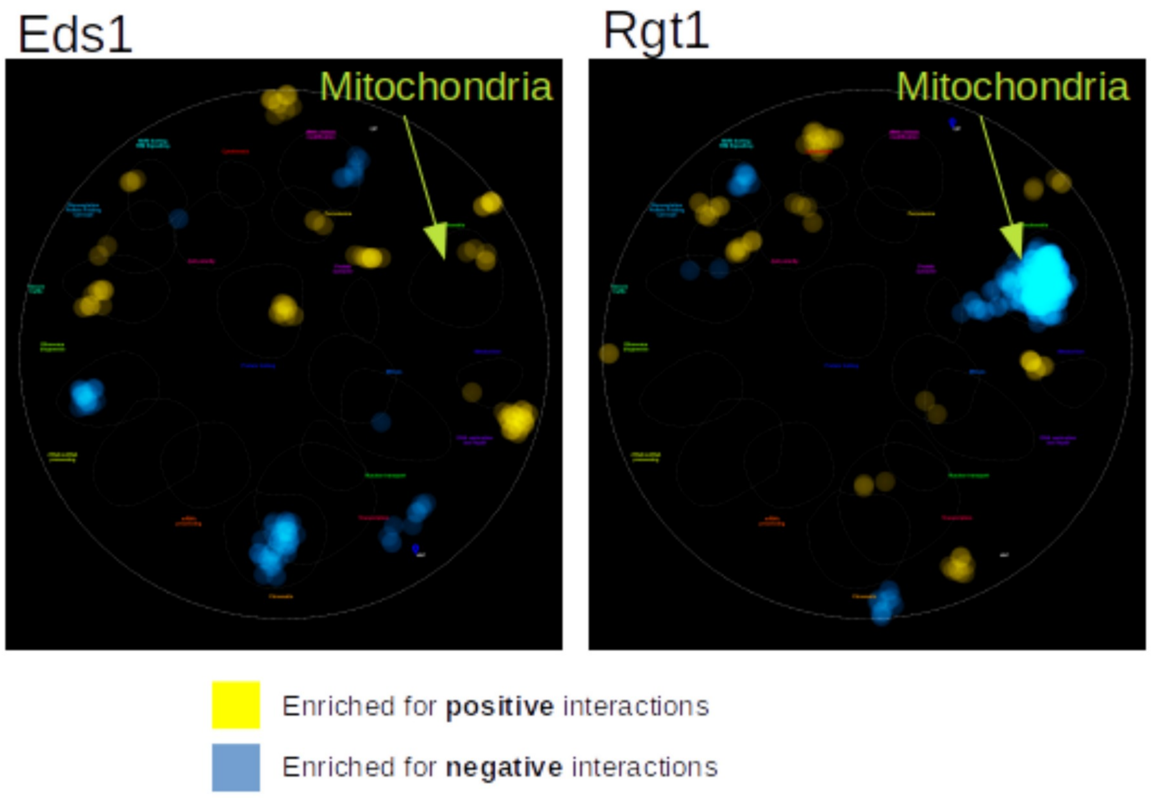

Figure 9: Eds1/Rgt1 Genetic Interaction Map

This shows interaction between Eds1 (left) and Rgt1 (right) and a large set of the genes in the S.cerevisiae genome. This results are mapped onto functional areas. This was generated using thecellmap.org[29]. Significantly, Rgt1 shows strong negative interaction with mitochondrial related genes, signifying that Rgt1 is not in the same pathway as these genes. Conversely, Eds1 has slight positive interaction with these genes, signifying that Eds1 is in the same pathway as some of these genes and suggesting that Eds1 has a greater role in regulating respiration than Rgt1.

A3: High Confidence Regulators of Eds1

| Kinase | Transcription Factor |
|---|---|
| Pkc1 | Gis1 |
| Sch9 | Sfp1 |
| Snf1 | Nrg2, Cat9 |
| Hog1 | Msn2 |
| Tpk3 | Msn2 |

Table 2: High confidence inferred transcription factor and inferred kinase interactors

Inferred transcriptional regulators of Eds1 (right column), derived from Netprophet3.0, which includes both over-expression and under expression data in the modeling[67]. The highest confidence regulators of EDS1 predicted by Netprophet3.0 were then intersected with a inferred kinase-target interaction map[8]. Then, we used StringDB to assign scores to each kinase-TF edge[9]. In all, filtering was performed to select high confidence EDS1 regulators from Netprophet3.0 and high confidence Kinase-TF relationships. Finally, those relationships were filtered for only those with a StringDB score > 900, which StringDB considers 'high confidence'. The result strongly suggest that EDS1 expression is regulated by the cell's stress response pathways, which are also known regulators of Rgt1 activity.

## Appendix B: brentlabRnaSeqTools

This section is a selection of tutorials, or vignettes in the R vernacular, which direct users in how to conduct certain analyses which are downstream of the RNA sequencing processing pipeline, using the brentlabRnaSeqTools. All of this may be found at https://brentlab.github.io/brentlabRnaSeqTools/index.html in a more pleasing format.

### B1: Getting Started with brentlabRnaSeqTools

**Setting Up Your R Environment**

**Requirements**

An up to date R (so whatever the current version is – check CRAN) and up to date Rstudio

**Install the following**

```
cran_pkgs = c('tidyverse', 'devtools')

# note: increase/decrease the number of CPUs. Any machine should be able to
# handle at least 5. You can probably bump this up to 10 without checking
# anything
install.packages(cran_pkgs, Ncpus = 5)
```

It is possible that there wil be install errors that come up during the installation of these two pages. If you are on the computational/analyst side of things, then this is a good opportunity to read the error messages and figure out what to install – both packages may have some c or maybe fortran dependencies which you will need to install onto your computer. Googling the error helps, and frequently R will say, “you need a gcc compiler” or something like that.

If you are not one of the computational/analyst people, then ask for help if there are errors in the installation of those two packages.

**Install the brentlabRnaSeqTools**

If there is a 'passing' badge in the github README, then this has been successfully built on up to date linux, mac and windows OS. It should also work for you. But, if it doesn't, copy the error and make an issue report. Please also include the output of `Sys.info()` in your issue report.

```
# as above, you can increase Ncpus
remotes::install.packages("BrentLab/brentlabRnaSeqTools", Ncpus = 5)
```

**Set up some environmental variables**

Environmental variables are variables that are read by R and loaded into your session when you launch R. You can set either 'user' level variables, which are loaded into any R session launched under your current user, and/or you can set project level environmental variables, which are only set if you launch into a project (see Using Rstudio Projects below). These are convenient to use, and are particularly good at avoiding any embarrassing leaks of login credentials onto github.

**User level environmental variables**

1. open Rstudio

```
usethis::edit_r_environ('user')
```

2. in the file that is opened, enter some variables that you'd like to have

```
db_username = "some_username"
db_password = "some_password"
```

3. restart your R session

4. see that you can access these variables like so:

```
Sys.getenv("db_username")
# output will be:
#> "some_password"
```

If you put the correct username/password into your environment file, then you can access the database like so:

```
library(brentlabRnaSeqTools)

meta = getMetadata(
  database_info$kn99$db_host,
  database_info$kn99$db_name,
  Sys.getenv("db_username"),
  Sys.getenv("db_password"))
```

Since the user level environment file is stored in `~/.Renviron`, there is no danger of pushing these credentials up to git if you are in a directory which you are tracking with git/github.

## Using an Rstudio project

### set your working directory

If you do not already have a directory in which you store the projects you work on, create one. I suggest a direcory called `projects` in your `$HOME`

```
project_dir = "~/projects"
if(!dir.exists(project_dir)){
  dir.create(project_dir)
}
setwd(project_dir)
```

### Create a new Rstudio project

A project is a subset of the infrastructure in an R software package. You can use a virtual environment, .Renviron files for environmental variables, etc. It is a tool for reproducibility

```
# if this is for the nineyMinuteInduction data, for example, you might call
# this project "ninetyMinInduction".
project_name = "my_new_project"

usethis::create_project(project_name)
```

### Using the project directory

A new Rstudio session will launch in your new project directory. Now, whenever you launch this project, all of your environment variables in .Renviron will be loaded. If you are using a virtual environment (a good idea for reproducibility. Use `renv`), then your virtual environment will be automatically launched, also.

The `/R` directory is for R scripts. You could make a `notebooks` directory, or just put the notebook in the project parent directory, for example.

See here for a project directory example

**Using project level environmental variables in an active R project**

Just as with the user level environmental variables, you can set environmental variables for a particular project. These are read in addition to the user level variables, and will overwrite them if there are two that are named the same thing. The first thing to do is to make sure that the project level `.Renviron` is in the `.gitignore` so that you don't accidently push up login credentials

```
# you can use this, or just click on the .gitignore file in the project
usethis::edit_git_ignore('project')
```

In the `.gitignore` file, add `.Renviron`. Next, open a project level `.Renviron`

```
usethis::edit_r_environ("project")
```

and edit as before. Remember to re-launch your R session after editing the `.Renviron` to have access to the environmental variables.

**Using data from the lts archive**

```
library(brentlabRnaSeqTools)
library(tidyverse)

# mount the cluster, or download the 20220208 kn99 db archive
# paths will need to be updated for your machine. These are examples
# of what it would look like if you mount.
archive_prefix = '/mnt/lts/sequence_data/rnaseq_data/kn99_database_archive'
coldata_df = read_csv(file.path(archive_prefix,
"20220208/combined_df_20220208.csv"))
count_df = read_csv(file.path(archive_prefix, "20220208/counts.csv"))

gene_ids = getGeneNames(database_info$kn99$db_host,
                        database_info$kn99$db_name,
                        Sys.getenv('db_username'),
                        Sys.getenv('db_password'))

gene_ids = gene_ids$gene_id[1:6967]

coldata_df_fltr = coldata_df %>%
  filter(purpose == "fullRNASeq",
         !is.na(fastqFileName))

count_df_fltr = count_df[, colnames(count_df) %in%
                           coldata_df_fltr$fastqFileName]

coldata_df_fltr = filter(coldata_df_fltr,
                         fastqFileName %in%
                           colnames(count_df_fltr))

# make sure the order of the count cols matches order of the metadata rows
count_df_fltr = count_df_fltr[order(match(colnames(count_df_fltr),
                                          coldata_df_fltr$fastqFileName))]

# if this checks false, stop and figure out why. useful functions would be
# setdiff(). Read the ?setdiff docs -- it is asymetric.
stopifnot(identical(colnames(count_df_fltr),
                    coldata_df_fltr$fastqFileName))

dds = DESeq2::DESeqDataSetFromMatrix(countData = count_df_fltr,
                                     colData = coldata_df_fltr,
                                     design = ~1)
rownames(dds) = gene_ids

blrs = brentlabRnaSeqSet(dds = dds)

# everything should work as usual from here
```

## B2: Process a New Run From MGI

After receiving a new run from MGI, align, count and QC brentlabRnaSeqTools package

```
library(brentlabRnaSeqTools)
library(tidyverse)
```

### Get the metadata from the database

```
meta = getMetadata(
  database_info$kn99$db_host,
  database_info$kn99$db_name,
  Sys.getenv("db_username"),
  Sys.getenv("db_password")
)
```

### Filter out the reads of interest

```
run_df = meta %>%
  filter(runNumber == 5500)
```

### Look at it. Make sure it is correct

```
View(run_df)
```

### Write out

If you have mounted your local to HTCF, you can write directly to HTCF. Otherwise, write to your computer and follow the directions below to move it to HTCF.

```
sample_sheet = createNovoalignPipelineSamplesheet(run_df,
"/scratch/mblab/chasem/rnaseq_pipeline/scratch_sequence")

write_csv(sample_sheet,
"/path/to/where/you/write_things/run_<some_identifier>.csv")
```

### Move a file to HTCF

Log into HTCF and make a directory that will store the input/output for this run. For example, if I were processing `run_1234`, I would log into HTCF and make a directory like so:

```
$ mkdir /scratch/mblab/chasem/rnaseq_pipeline/align_count_results/run_1234
```

Back on your local computer, send the file from your local to HTCF with `scp`

```
# copy the file from your computer to a directory in your personal subdirectory
# of the lab scratch space
$ scp /path/to/where/you/write_things/run_<some_identifier>.csv \
<your_username>@htcf.wustl.edu:/scratch/mblab/<your_username>/rnaseq_pipeline/
align_count_results/run_1234
```

Please note that there is no requirement that the path look like this: `<your_username>/rnaseq_pipeline/align_count_results/run_1234`. It is just an example of what it might look like.

**On HTCF, start the pipeline**

The first time you do this, navigate to your scratch space and do this:

```
$ git clone https://github.com/cmatKhan/brentlab_rnaseq_nf.git
```

If you have done this before, navigate into your brentlab_rnaseq_nf directory and do this to pull any possible updates:

```
$ git pull https://github.com/cmatKhan/brentlab_rnaseq_nf.git
```

If you get some sort of error that says something like, “this is not a git directory”, when you know it is, in fact, a git directory, then HTCF deleted some files. In that case, navigate out of `brentlab_rnaseq_nf`, delete it (`rm -rf brentlab_rnaseq_nf`), and use the `git clone` command described above.

**Copy the fastq files into scratch**

I suggest having a `rnaseq_pipeline` directory in your personal scratch space. If you don’t have one, make one, or otherwise navigate to where ever you are keeping rnaseq type data. You can use the script here for the job. Ask if you need help setting this up to use on HTCF. Here is an example, assuming that you have this scriptin your `$PWD`

```
$ ./fastqFilesToScratchFromSamplesheet.sh path/to/sample_sheet.csv
/lts/mblab/sequence_data/rnaseq_data/lts_sequence
```

**Run the pipeline**

Navigate into the directory into which you are going to store the input/output of the pipeline, eg:

```
$ cd rnaseq_pipeline/align_count_results/run_1234
```

**Make the params file**

You will need a file describing the experiment. This should go into the directory where the input/output is stored. It must look like this, and the paths must be correct. Save this as, eg, params_run1234.json. The example below is also shown here

```
{
"output_dir": ".",
"sample_sheet": "path/to/sample_sheet.csv",
"run_number": "1234",
"KN99_novoalign_index":
"/scratch/mblab/chasem/rnaseq_pipeline/genome_files/KN99/KN99_genome_fungidb.nix",
"KN99_fasta":
"/scratch/mblab/chasem/rnaseq_pipeline/genome_files/KN99/KN99_genome_fungidb.fasta"
,"KN99_stranded_annotation_file":
"/scratch/mblab/chasem/rnaseq_pipeline/genome_files/KN99/KN99_stranded_annotations_
fungidb_augment.gff",
"KN99_unstranded_annotation_file":
"/scratch/mblab/chasem/rnaseq_pipeline/genome_files/KN99/KN99_no_strand_annotations
_fungidb_augment.gff",
"htseq_count_feature": "exon"
}
```

**Run nextflow**

**NOTE:** both in the params file, and in the run script below, you *must* make sure that the paths are correct. They won't be, unless you change them to make them correct for you.

Next, make a script to run the pipeline. [An example may be found here] ((https://github.com/BrentLab/brentlabRnaSeqTools/blob/main/inst/bash/run_novo_nf_pipeline.sh), or you can copy/paste what is below into a file. Remember to update the paths.

```
#!/bin/bash

#SBATCH --time=15:00:00  # right now, 15 hours. change depending on time
expectation to run
#SBATCH --mem-per-cpu=10G
#SBATCH -J your_jobname.out
#SBATCH -o your_jobname.out

ml miniconda

# until HTCF updates and spack is available, this works. When HTCF updates and
# we have spack, ill update this...though at that point, hopefully we are no
# longer using this pipeline
source activate /scratch/mblab/chasem/rnaseq_pipeline/conda_envs/nextflow

mkdir tmp

nextflow run /path/to/brentlab_rnaseq_nf/main.nf \
             -params-file /path/to/your_params.json
```

You can check progress by looking at the squeue and the `<your_jobname>.out`. Right now, it is taking a very long time for HTCF to launch nextflow. When HTCF updates to the 'new' implementation, it starts much faster.

## B3: Environmental Perturbation Analysis Data Creation

Create a data object ready for analysis for the Environmental Perturbation experiment set.

brentlabRnaSeqTools package

**BrentlabRnaSeqSet**

The `BrentlabRnaSeqSet` is a child of the DESeqDataSet, which is a child of SummarizedExperiment.

Therefore, the `brentlabRnaSeqSet` has all of the DESeq functionality – eg, DESeq size factor normalization, or the `DESeq()` function – as well as all of the SummarizedExperiment methods, and some more 'custom' methods more suited to our purposes. It is easily extensible – for those more computationally minded users, it would be a good idea to learn and use the object oriented programming tools. There are many cases in which doing so will make your code less error prone, easier to test, more reproducible, and easier to maintain long term.

### Setup

```
library(brentlabRnaSeqTools)
library(rtracklayer)
library(tidyverse)

# set variables

KN99_GFF_RDS = Sys.getenv("kn99_stranded_gff_rds")
DB_USERNAME = Sys.getenv("db_username")
DB_PASSWORD = Sys.getenv("db_password")

# note: I mount to the cluster and output directly to it
DDS_OUTPUT_DIR = "."

# controls whether dds objects are written
WRITE_OUT = FALSE
```

### pull the database as a brentlabRnaSeqSet object

```
blrs = brentlabRnaSeqSetFromDatabase('kn99',DB_USERNAME, DB_PASSWORD)
```

### Add gene level data (optional)

this adds all of the data regarding each locus as a GRange object to the gene data slot of the brentlabRnaSeqSet object. Useful if you are going to use other Bioconductor packages.

```
kn99_gff = readRDS(KN99_GFF_RDS)

kn99_genes = kn99_gff[kn99_gff$ID %in% rownames(blrs)]

rowRanges(blrs) = kn99_genes[order(match(kn99_genes$ID,rownames(blrs)))]

rownames(blrs) = rowData(blrs)$ID
```

### Create Sets

If the experiment set you want is not already created, issue an ‘issue report’, and describe the **EXACT** filter that you want to use to create the set. What goes into your set depends on what you describe as your filter, so spend time with the database to figure out what is there.

```
ep_list = list(
  wt = createExperimentSet(blrs, 'envPert_epWT'),
  titr = createExperimentSet(blrs, 'envPert_titrationWT'),
  pert = createExperimentSet(blrs, 'envPert_perturbed')
)

# NOTE!! AS OF 20220201 CNAG_03894 forms a linear depdendence with both
# concat treatment and libraryDate columns. it is being removed here to solve
# that issue

ep_list$pert = ep_list$pert[,ep_list$pert$genotype1 != "CNAG_03894"]
```

## Quality Filter

```
ep_list_qc_passing = map(names(ep_list),
                         ~qaFilter(ep_list[[.]],1,paste0("ep_",., "_iqr")))
```

## Expression Filter

```
# How this is done is up to you, and obviously affects what genes are left in.
# Below is an example. You need to think about the thresholds and filter method
# that suits your purpose best.

expr_fltr_list = map(ep_list, ~rowSums(edgeR::cpm(counts(.))>4) >= 4)

ep_list_qc_passing_fltr = map2(ep_list_qc_passing, expr_fltr_list, ~.x[.y,])
```

## Set Design

```
setBaseConcatTreatmentBaseCond = function(ep_set, concat_base_cond){
  colData(ep_set)$concat_treatment =
    relevel(colData(ep_set)$concat_treatment, ref = concat_base_cond)

  colData(ep_set) = droplevels(DataFrame(colData(ep_set)))

  ep_set

}
```

```
setEpDesign = function(ep_set, design){
  design(ep_set) = design

  ep_set
}

concat_base_cond_list = list(

  wt = "YPD_noAtmosphere_30_noTreatment_noTreatmentConc_noPH_30",
  titr = "RPMI_noAtmosphere_30_noTreatment_noTreatmentConc_noPH_30",
  pert = 'PBS_noAtmosphere_30_noTreatment_noTreatmentConc_noPH_0'
)

ep_designs = list(
  wt = formula(~libraryDate + concat_treatment),
  titr = formula(~libraryDate + concat_treatment),
  pert = formula(~libraryDate + concat_treatment + genotype1)
)

ep_list_qc_passing_fltr = map2(ep_list_qc_passing_fltr,
                               concat_base_cond_list,
                               setBaseConcatTreatmentBaseCond)

ep_list_qc_passing_fltr = map2(ep_list_qc_passing_fltr,
                               ep_designs,
                               setEpDesign)
```

### Coerce back to DeseqDataObjects for proessing

```
ep_dds_list = map(ep_list_qc_passing_fltr, coerceToDds)
names(ep_dds_list) = names(ep_list)

DDS_OUTPUT_DIR = "/mnt/scratch/rnaseq_pipeline/experiments/epDatafreeze"

WRITE_OUT = TRUE

if(WRITE_OUT){
  today = format(lubridate::today(),"%Y%m%d")

  output_dir = file.path(DDS_OUTPUT_DIR, today)

  dir.create(output_dir, recursive=TRUE)

  map(names(ep_dds_list), ~write_rds(ep_dds_list[[.]],
                                     file.path(output_dir,
                                               paste0("ep_",.,".rds"))))
}
```

## B4: 90 Minute Induction Analysis Data Creation

Create a data object ready for analysis for the 90 Minute Induction experiment set.

brentlabRnaSeqTools package

**BrentlabRnaSeqSet**

The `BrentlabRnaSeqSet` is a child of the DESeqDataSet, which is a child of SummarizedExperiment.

Therefore, the `brentlabRnaSeqSet` has all of the DESeq functionality – eg, DESeq size factor normalization, or the `DESeq()` function – as well as all of the SummarizedExperiment methods, and some more ‘custom’ methods more suited to our purposes. It is easily extensible – for those more computationally minded users, it would be a good idea to learn and use the object oriented programming tools. There are many cases in which doing so will make your code less error prone, easier to test, more reproducible, and easier to maintain long term.

**Setup**

```
library(brentlabRnaSeqTools)
library(rtracklayer)
library(tidyverse)

# set variables

KN99_GFF_PATH = Sys.getenv("kn99_gff")
DB_USERNAME = Sys.getenv("db_username")
DB_PASSWORD = Sys.getenv("db_password")

# note: I mount to the cluster and output directly to it
DDS_OUTPUT_DIR =
"/mnt/htcf_scratch/chasem/rnaseq_pipeline/experiments/90minDataFreeze"

# controls whether dds objects are written
WRITE_OUT = FALSE

# Pull the database ----

blrs = brentlabRnaSeqSetFromDatabase('kn99',DB_USERNAME, DB_PASSWORD)
```

```
# filter down to just the protein coding genes. Note that this works because the
# nctr-rna annotations were all added to the end of the annotation file.
blrs = blrs[1:6967,]

# Add gene level data (optional) ----

# this adds all of the data regarding each locus as a GRange object to the
# gene data slot of the brentlabRnaSeqSet object. Useful if you are going to
# use other Bioconductor packages.

kn99_gff = rtracklayer::import(KN99_GFF_PATH)

kn99_genes = kn99_gff[kn99_gff$ID %in% rownames(blrs)]

rowRanges(blrs) = kn99_genes[order(match(kn99_genes$ID,rownames(blrs)))]

rownames(blrs) = rowData(blrs)$ID
```

If the experiment set you want is not already created, issue an ‘issue report’, and describe the **EXACT** filter that you want to use to create the set. What goes into your set depends on what you describe as your filter, so spend time with the database to figure out what is there.

**90 minute induction**

**2016 Grant set, singles**

```
blrs_90min = createExperimentSet(blrs, 'ninetyMin_2016Grant')

# Quality Filter ----

blrs_90min_qc_passing = qaFilter(blrs_90min,
                                 rle_iqr_threshold = .6125,
                                 iqr_colname = "ninetyMin_iqr")
```

**Set perturbed loci to zero**

```
blrs_90min_qc_passing = setPerturbedLociToZero(blrs_90min_qc_passing)
```

**Expression filtering**

Filter out low expression genes (and anything else you don't want in the analysis set). How you do this is up to you, the analyst. You might use the method proposed in this paper by, among others, Wolfgang Huber, and implemented in the bioconductor package `geneFilter` (as well as DESeq, by default), or you could filter based on some number of samples having greater than some threshold of expression, which is shown below.

```
# How this is done is up to you, and obviously affects what genes are left in.
# Below is an example. You need to think about the thresholds and filter method
# that suits your purpose best.

# mid_expression_filter <- rowSums(edgeR::cpm(counts(blrs_90min_qc_passing))>4) >=
4

high_disp_fltr = rownames(blrs_90min_qc_passing) %in% passing_genes_all$gene_id

blrs_90min_qc_passing = blrs_90min_qc_passing[high_disp_fltr,]
```

**Split the replicate groups**

```
# note, qaFilter returns those samples with less than 3 replicates, which have NA
# in the RLE stats. Filter those out, also

blrs_90min_qc_passing =
  blrs_90min_qc_passing[,!is.na(colData(blrs_90min_qc_passing)$ninetyMin_iqr)]

protocol_tallies = replicateByProtocolTally(blrs_90min_qc_passing)

# protocol_tallies$replicates_with_less_than_four_in_both_old_or_new. All of the
# genotypes should be in the list below

protocol_fltr =
  as_tibble(colData(blrs_90min_qc_passing)) %>%
  filter(!(genotype1 == "CNAG_00031" & libraryProtocol == "SolexaPrep"),
         !(genotype1 == "CNAG_00871" & libraryProtocol == "E7420L"),
         !(genotype1 == "CNAG_00883" & libraryProtocol == "E7420L"),
         !(genotype1 == "CNAG_01626" & libraryProtocol == "E7420L"),
         !(genotype1 == "CNAG_02774" & libraryProtocol == "E7420L"),
         !(genotype1 == "CNAG_03018" & libraryProtocol == "SolexaPrep"),
         !(genotype1 == "CNAG_03279" & libraryProtocol == "SolexaPrep"),
         !(genotype1 == "CNAG_03849" & libraryProtocol == "E7420L"),
         !(genotype1 == "CNAG_04353" & libraryProtocol == "E7420L"),
         !(genotype1 == "CNAG_05222" & libraryProtocol == "SolexaPrep")) %>%
  pull(fastqFileNumber)
```

```
blrs_90min_qc_passing_protocol_fltr =
  blrs_90min_qc_passing[,colData(blrs_90min_qc_passing)$fastqFileNumber %in%
                        protocol_fltr]

colData(blrs_90min_qc_passing_protocol_fltr)$libraryDate =
  droplevels(colData(blrs_90min_qc_passing_protocol_fltr)$libraryDate)
```

### Re-examine tallies after separating the protocol groups

```
# note that the column names may not reflect what is actually pasesd -- this
# function needs some updating due to hard coding colnames. Columns are in order of
the
# tables passed in
protocol_fltr_tally =
  createInductionSetTally(as_tibble(colData(blrs_90min)),
                        as_tibble(colData(blrs_90min_qc_passing)),
                        as_tibble(colData(blrs_90min_qc_passing_protocol_fltr)),
                        grant_df)
```

### Split

```
blrs_90min_qc_passing_protocol_fltr_split =
  splitProtocolGroups(blrs_90min_qc_passing_protocol_fltr)

# libraryprotocol date date have their levels dropped. genotype1 still needs it
# this needs to be handled internally somehow

blrs_90min_qc_passing_protocol_fltr_split$SolexaPrep$genotype1 =
  droplevels(blrs_90min_qc_passing_protocol_fltr_split$SolexaPrep$genotype1)

blrs_90min_qc_passing_protocol_fltr_split$E7420L$genotype1 =
  droplevels(blrs_90min_qc_passing_protocol_fltr_split$E7420L$genotype1)
```

### Examine replicate tallies by protocol group

```
# filter out those samples with less than 3 replicates in either libraryDate or
# genotype1

# helper function to create model matricies by protocol
createFullSetModelMatricies = function(full_set_split){

  full_set_mm_list = list(
    E7420L = model.matrix(~libraryDate+genotype1,
                        as_tibble(colData(full_set_split$E7420L))),
    SolexaPrep = model.matrix(~libraryDate+genotype1,
                            as_tibble(colData(full_set_split$SolexaPrep))))
  full_set_mm_list
}

# return those replicate sets which have less than 2 replicates
lowReplicateParams = function(model_matrix){

  mm_summary_df = tibble(model_params = colnames(model_matrix),
                        replicate_tally=colSums(model_matrix))

  low_rep_parameters = mm_summary_df %>%
    filter(replicate_tally < 2) %>%
    pull(model_params)

  low_rep_parameters = str_remove(low_rep_parameters, "libraryDate")
  low_rep_parameters = str_remove(low_rep_parameters, "genotype1")

  return(low_rep_parameters)
}

# create model matricies from the blrs_90min_qc_passing_protocol_fltr_split
full_set_mm_list =
  createFullSetModelMatricies(blrs_90min_qc_passing_protocol_fltr_split)

# find low rep parameters
low_rep_parameters = lapply(full_set_mm_list ,lowReplicateParams)

# all dates, no genotypes
low_rep_parameters = lapply(low_rep_parameters, as.factor)

# create filters for each protocol set
e7420l_fltr =
  !colData(blrs_90min_qc_passing_protocol_fltr_split$E7420L)$libraryDate %in%
    low_rep_parameters$E7420L

solexaprep_fltr =
  !colData(blrs_90min_qc_passing_protocol_fltr_split$SolexaPrep)$libraryDate %in%
    low_rep_parameters$SolexaPrep

# create the full set list
fltr_full_set_90min = list(
  E7420L = blrs_90min_qc_passing_protocol_fltr_split$E7420L[, e7420l_fltr],
  SolexaPrep = blrs_90min_qc_passing_protocol_fltr_split$SolexaPrep[,
solexaprep_fltr]
)

# drop the factor levels which no longer exist from the filtered  libraryDate
column
colData(fltr_full_set_90min$E7420L)$libraryDate =
  droplevels(colData(fltr_full_set_90min$E7420L)$libraryDate)

colData(fltr_full_set_90min$SolexaPrep)$libraryDate =
  droplevels(colData(fltr_full_set_90min$SolexaPrep)$libraryDate)

# no more low rep sets left
fltr_full_set_mm = createFullSetModelMatricies(fltr_full_set_90min)

low_rep_parameters_2 = lapply(fltr_full_set_mm, lowReplicateParams)
```

### Recheck tallies again

```
protocol_fltr_tally =
  createInductionSetTally(as_tibble(colData(blrs_90min)),
                          as_tibble(colData(blrs_90min_qc_passing)),
                          rbind(as_tibble(colData(fltr_full_set_90min$E7420L)),

as_tibble(colData(fltr_full_set_90min$SolexaPrep))),
                          grant_df)
```

### Create hold out set

```
min_set_size = 1

hold_out_set = list(
  SolexaPrep = createTestTrainSet(fltr_full_set_90min$SolexaPrep, min_set_size),
  E7420L = createTestTrainSet(fltr_full_set_90min$E7420L, min_set_size)
)
```

### Set the experiment designs

```
setNinetyMinDesign = function(obj){
  obj$genotype1 = as.factor(obj$genotype1)
  relevel(obj$genotype1, ref = "CNAG_00000")

  obj$libraryDate = as.factor(obj$libraryDate)
  relevel(obj$libraryDate, ref = min(as.character(obj$libraryDate)))

  design(obj) = formula(~libraryDate + genotype1)

  obj
}

fltr_full_set_90min = map(fltr_full_set_90min, setNinetyMinDesign)

hold_out_set$SolexaPrep$train = setNinetyMinDesign(hold_out_set$SolexaPrep$train)
hold_out_set$E7420L$train = setNinetyMinDesign(hold_out_set$E7420L$train)
```

### convert everything back to DESeqDataObjects for the time being

```
hold_out_set$SolexaPrep = map(hold_out_set$SolexaPrep, coerceToDds)
hold_out_set$E7420L = map(hold_out_set$E7420L, coerceToDds)

fltr_full_set_90min = map(fltr_full_set_90min, coerceToDds)
```

### combine protocol groups

```
refactorDesign = function(dds){

dds$libraryProtocol =
  dds$libraryProtocol %>%
  as.character() %>%
  as.factor() %>%
  droplevels()

dds$libraryDate =
  dds$libraryDate %>%
  as.Date() %>%
  as.factor() %>%
  droplevels()

dds$genotype1 =
  dds$genotype1 %>%
  as.character() %>%
  as.factor() %>%
  droplevels()

mm = model.matrix(~libraryProtocol + libraryDate + genotype1,
                  as_tibble(colData(dds)))

min_date = colData(dds) %>%
  as_tibble() %>%
  filter(libraryProtocol == "E7420L") %>%
  pull(libraryDate) %>%
  as.Date() %>%
  min()

mm_redux = mm[,-which(colnames(mm) == paste0("libraryDate", min_date)), drop=FALSE]

design(dds) = mm_redux

dds
}

full_set_90min_both_protocols = cbind(fltr_full_set_90min$SolexaPrep,
                                       fltr_full_set_90min$E7420L)

sizeFactors(full_set_90min_both_protocols) =
  c(estimateSizeFactors(fltr_full_set_90min$SolexaPrep)$sizeFactor,
    estimateSizeFactors(fltr_full_set_90min$E7420L)$sizeFactor)

full_set_90min_both_protocols = refactorDesign(full_set_90min_both_protocols)

train_set_90min_both_protocols = cbind(hold_out_set$SolexaPrep$train,
                                        hold_out_set$E7420L$train)

sizeFactors(train_set_90min_both_protocols) =
  c(estimateSizeFactors(hold_out_set$SolexaPrep$train)$sizeFactor,
    estimateSizeFactors(hold_out_set$E7420L$train)$sizeFactor)

train_set_90min_both_protocols = refactorDesign(train_set_90min_both_protocols)

test_data = list(
  SolexaPrep = hold_out_set$SolexaPrep$test,
  E7420L = hold_out_set$E7420L$test
)
```

### Write out

```
if(WRITE_OUT){
  today = format(lubridate::today(),"%Y%m%d")

  output_dir = file.path(DDS_OUTPUT_DIR, today)

  dir.create(output_dir, recursive=TRUE)

  # note: this is probably better done with a function and map() as the list gets
  #       longer
  write_rds(fltr_full_set_90min$E7420L,
            file.path(output_dir,"full_set_new_protocol_input.dds"))

  write_rds(fltr_full_set_90min$SolexaPrep,
            file.path(output_dir, "full_set_old_protocol_input.dds"))

  write_rds(test_data,
            file.path(output_dir, "test_data.rds"))

  write_rds(hold_out_set$E7420L$train,
            file.path(output_dir, "train_set_new_protocol_input.rds"))

  write_rds(hold_out_set$SolexaPrep$train,
            file.path(output_dir, "train_set_old_protocol_input.rds"))

    write_rds(full_set_90min_both_protocols,
          file.path(output_dir, "full_set_both_protocol_input.rds"))

  write_rds(train_set_90min_both_protocols,
          file.path(output_dir, "train_set_both_protocol_input.rds"))

}
```

**Copy the HTCF DESeq scripts to the working directory**

These scripts will run DESeq in parallel on HTCF. The only items you'l need to edit is the path to the lookup file (if you are running more than one model), and to the dds_input in deseq_mpi.sh. If you do not keep deseq_de.R in the same directory as deseq_mpi.sh, then you'll need to update the path to the deseq_de.R script, also.

```
if(WRITE_OUT){
  htcf_deseq_scripts = c(system.file('bash',
                              'htcf_parallel_deseq.sh',
                              package = "brentlabRnaSeqTools"),
                         system.file('R_executable',
                                 'deseq_de.R',
                                  package = "brentlabRnaSeqTools"))

lapply(htcf_deseq_scripts, file.copy, to = DDS_OUTPUT_DIR)
}
```

## B5: QC – Library Quality

Details of library quality metrics and thresholds

**Cryptococcus**

| Metric | Threshold | Status |
|---|---|---|
| Protein Coding Total | 1e6 | 1 |
| Not Aligned Total Percent | .07 | 2 |
| Perturbed Coverage | .25 | 4 |
| NAT coverage: expected | .5 | 8 |
| NAT log2cpm: expected | 5 | 8 |
| NAT coverage: unexpected | .5 | 16 |
| NAT log2cpm: unexpected | 2.5 | 16 |

| Metric | Threshold | Status |
|---|---|---|
| G418 log2cpm: expected | 5.688 | 32 |
| G418 log2cpm: unexpected | 5.688 | 64 |
| overexpression FOW | 2 | 128 |
| missing marker in metadata | | 256 |

Note that libraryComplexity is included in the qc metrics now, but there is no threshold currently set. The default setting for libraryComplexity is to calculate the portion of the total counts made up by the top 25 expressed genes

```
library(brentlabRnaSeqTools)
library(tidyverse)
```

**Get the metadata from the database**

```
meta = getMetadata(
  database_info$kn99$db_host,
  database_info$kn99$db_name,
  Sys.getenv("db_username"),
  Sys.getenv("db_password")
)
```

**Filter out the reads of interest**

```
run_df = meta %>%
  filter(runNumber == 5500)
```

**Create QC table**

You will need to have the cluster mounted to your local for this to work. Note that you may get warnings about the index file being older than the bam. That isn't critical and can generally be ignored, though something I am aware of and trying to figure out.

```
pipeline_out =
"/mnt/scratch/rnaseq_pipeline/align_count_results/run_5500/rnaseq_pipeline_results/
run_5500_samples"

run_qc = novoalignPipelineQC(run_df, pipeline_out,
Sys.getenv("kn99_stranded_gff_rds"))
```

### parse this out into a QC sheet

```
qc_table = addQcColsToMeta(run_df, run_qc)


# write_csv(dplyr::select(qc_df, -c(genotype1, genotype2, marker1, marker2)),
#             "~/Desktop/tmp/run_5500_qc.csv")
```

### auto and manual audit

```
audited_qc_df = autoAuditQcTable(qc_table) %>%
  # add manual audit columns
  mutate(manualAudit = NA,
         manualStatus = NA)
```

### update manual audit

```
audited_qc_df = edit(audited_qc_df)

audited_qc_df %>%
  dplyr::select(-c(genotype1, genotype2, marker1, marker2)) %>%
write_csv("~/Desktop/tmp/run_5500_qc.csv")
```

### post Counts to database

```
count_files = Sys.glob(file.path(pipeline_out, "count", "*_read_count.tsv"))
names(count_files) = str_remove(basename(count_files), "_read_count.tsv")

compiled_counts = map(names(count_files), ~readHTSeqFile(count_files[[.]], .)) %>%
  plyr::join_all() %>%
  filter(!startsWith(feature, "__")) %>%
  dplyr::select(-feature)

fastq_df = read_csv("/mnt/lts/sequence_data/rnaseq_data/kn99_database_archive/
20220131/fastqFiles.csv")

# note: commented out to prevent me from accidently running this
# res = postCounts(database_info$kn99$urls$counts,
#                  5500,
#                  Sys.getenv("kn99_db_token"),
#                  compiled_counts,
#                  fastq_df)
#
# res
```

## Send QC to database

```
# note: commented out in the vignette to prevent me from running accidently
# res = postQcSheet_test(database_info$kn99$urls$qualityAssess,
#             Sys.getenv("kn99_db_token"),
#             5500,
#             "~/Desktop/tmp/run_5500_qc.csv",
#             "/mnt/lts/sequence_data/rnaseq_data/kn99_database_archive/20220131/
fastqFiles.csv")

# a code of 200 or 201 means it worked. anything else means failure
# res
```

## IGV browser shot

```
kn99_gff = readRDS(Sys.getenv("kn99_stranded_gff_rds"))

unique_loci = str_replace(unique(run_qc$locus), "CNAG", "CKF44")

unique_loci = c(unique_loci, c("CNAG_NAT", "CNAG_G418"))

bam_prefix = file.path(pipeline_out, "align")
bam_suffix = "_sorted_aligned_reads_with_annote.bam"

bam_list_df = run_df %>%
  distinct(fastqFileNumber, .keep_all = TRUE)

bam_list = unlist(map(pull(bam_list_df, fastqFileName), ~file.path(bam_prefix,
paste0(., bam_suffix))))

bam_list = map(bam_list, ~c(.,
"/mnt/scratch/rnaseq_pipeline/align_count_results/run_5500/rnaseq_pipeline_results/
run_5500_samples/align/
Brent_3235_GTAC_1_SIC_Index2_08_TGAGGTTATC_AAGCACGT_S2_R1_001_sorted_aligned_reads_
with_annote.bam"))
names(bam_list) = bam_list_df$fastqFileNumber

igvScriptAll = function(ffn){
  for(locus in unique_loci){
      granges = kn99_gff[kn99_gff$ID == locus & kn99_gff$type == 'gene']
      print(granges)
      basename = paste(ffn, locus, sep = "_")
      createIgvBatchscript(
        bam_list = bam_list[[ffn]],
        granges = granges,
        igv_genome = Sys.getenv("kn99_stranded_igv_genome"),
        output_dir = "/home/oguzkhan/Desktop/tmp/igv/",
        output_file_basename = basename)
  }

}

igvScriptAll(names(bam_list)[1])
```

**Running the IGV batch scripts**

It is likely easiest just to cd into the place where you output the scripts

```
cd /home/oguzkhan/Desktop/tmp/igv/scripts

for batch_script in $(ls .); do
  xvfb-run --auto-servernum igv.sh -b batch_script
done
```

I then scp these into the run directory before moving the run directory over to `lts`

```
scp -r /home/oguzkhan/Desktop/tmp/igv
chasem@htcf.wustl.edu:/mnt/scratch/rnaseq_pipeline/align_count_results/run_5500/
rnaseq_pipeline_results/run_5500_samples/

# then log into htcf, go to the pipeline output and move the whole run directory to
lts

rsync -aHv run_5500_samples /lts/mblab/sequence_data/rnaseq_data/lts_align_count/
```

## B6: QC – **Replicate Agreement**

Calculate RLE to evaluate replicate agreement

**Introduction**

This illustrates how to construct the data object for the Sample Agreement QC step – RLE – on each of the data sets. In each case, once you have added the design (the last step for each set), you need to write the object to file and send it to the cluster. I suggest mounting to the cluster and writing to the mounted directory. See the vignette *Running DESeq on HTCF* for instructions on running DESeq on

HTCF. Note that at the bottom of this script, there are instructions on how to copy the necessary scripts to run the DESeq model using MPI on htcf.

You can see all defined sets by entering `?createExperimentSet`. If an experiment set you are interested in does not exist as an option in `set_names`, then you'll need to use the `extractColData(blrs)` of the `blrs` object below to play around and come up with a set of filters to define your new set. See the github repository, "R/ExperimentSetFunctions.R" for examples

**Setup the environment**

```
library(brentlabRnaSeqTools)
library(rtracklayer)
library(tidyverse)
library(caret)

# set variables

KN99_GFF_RDS = Sys.getenv("kn99_stranded_gff_rds")
DB_USERNAME = Sys.getenv("db_username")
DB_PASSWORD = Sys.getenv("db_password")

# note: I mount to the cluster and output directly to it
DDS_OUTPUT_DIR = Sys.getenv("OUTPUT_DIR")
TODAY = format(lubridate::today(),"%Y%m%d")

# controls whether dds objects are written
WRITE_OUT = TRUE
```

**Pull the database**

```
blrs = brentlabRnaSeqSetFromDatabase('kn99',DB_USERNAME, DB_PASSWORD)
```

**Add gene level data (optional)**

this adds all of the data regarding each locus as a GRange object to the gene data slot of the brentlabRnaSeqSet object. Useful if you are going to use other Bioconductor packages, or brentlabRnaSeqTools::createIgvBrowserShot()

```
kn99_gff = readRDS(KN99_GFF_RDS)

kn99_genes = kn99_gff[kn99_gff$ID %in% rownames(blrs)]

rowRanges(blrs) = kn99_genes[order(match(kn99_genes$ID,rownames(blrs)))]
```

### 90minuteInduction

#### 2016 grant set

This is the data set defined by the single locus KOs in 90minuteInduction conditions for genotypes in the `grant_df` (this is a data object which is loaded as part of `brentlabRnaSeqTools` – take a look if you are interested)

##### Create the set and quality filter

```
blrs_90min_grant = createExperimentSet(blrs, 'ninetyMin_2016Grant')

# note that this filters out those samples which failed QC1,
# but but does not filter on RLE unless the argument rle_iqr_threshold
# is set to a numeric value
blrs_90min_grant_fltr = qaFilter(blrs_90min_grant)

# remove WT which fall on dates with no perturbed samples
blrs_90min_grant_fltr =
  filterWtByExperimentalLibdate_90min(blrs_90min_grant_fltr)
```

##### Add the design

```
blrs_90min_grant_fltr = estimateSizeFactorsByProtocol(blrs_90min_grant_fltr)

min_libdate = min(as.Date(colData(blrs_90min_grant_fltr)$libraryDate))

colData(blrs_90min_grant_fltr)$libraryDate =
  colData(blrs_90min_grant_fltr)$libraryDate %>%
  relevel(ref = as.character(min_libdate)) %>%
  droplevels()

colData(blrs_90min_grant_fltr)$libraryProtocol =
  colData(blrs_90min_grant_fltr)$libraryProtocol %>%
  factor() %>%
  relevel(ref = "SolexaPrep") %>%
  droplevels()

colData(blrs_90min_grant_fltr)$genotype1 =
  colData(blrs_90min_grant_fltr)$genotype1 %>%
  relevel(ref = "CNAG_00000") %>%
  droplevels()

mm = model.matrix(~libraryProtocol+libraryDate+genotype1,
                  extractColData(blrs_90min_grant_fltr))

min_new_protocol = min(as.Date((unique(pull(filter(
  extractColData(blrs_90min_grant_fltr),
  libraryProtocol == 'E7420L'),
  libraryDate)))))

mm = mm[,-which(colnames(mm) == paste0('libraryDate',min_new_protocol))]
```

### inspect the model matrix for linear dependencies

```
lin_dep_report = caret::findLinearCombos(mm)
```

### add the design to the object

```
design(blrs_90min_grant_fltr) = mm
rm(mm)
```

### write out

```
if(WRITE_OUT){

  outpath = file.path(DDS_OUTPUT_DIR,
                      "90minDataFreeze",
                      TODAY,
                      "grant_only_input.rds")

  dds = DESeq2::DESeqDataSetFromMatrix(
    colData = extractColData(blrs_90min_grant_fltr),
    countData = counts(blrs_90min_grant_fltr),
    design = design(blrs_90min_grant_fltr))

  sizeFactors(dds) = sizeFactors(blrs_90min_grant_fltr)

  write_rds(dds, outpath)
}
```

### 2016 Grant Set with Doubles

This has all double KO samples in which either of the KO loci are in the `grant_df`, plus single KO samples corresponding to one of the double KO perturbed loci and WT.

**Create set and quality filter**

```
blrs_grant_with_doubles = createExperimentSet(blrs,
'ninetyMin_2016GrantWithDoubles')

# note that this filters out those samples which failed QC1,
# but but does not filter on RLE unless the argument rle_iqr_threshold
# is set to a numeric value
blrs_grant_with_doubles_fltr = qaFilter(blrs_grant_with_doubles)

# remove WT which fall on dates with no perturbed samples
blrs_grant_with_doubles_fltr =
  filterWtByExperimentalLibdate_90min(blrs_grant_with_doubles_fltr)
```

**Add the design**

```
blrs_grant_with_doubles_fltr =
estimateSizeFactorsByProtocol(blrs_grant_with_doubles_fltr)

min_libdate = min(as.Date(colData(blrs_grant_with_doubles_fltr)$libraryDate))

colData(blrs_grant_with_doubles_fltr)$libraryDate =
  colData(blrs_grant_with_doubles_fltr)$libraryDate %>%
  relevel(ref = as.character(min_libdate)) %>%
  droplevels()

# add a 'genotype' column which is a concatenation of genotype1 and genotype2
colData(blrs_grant_with_doubles_fltr)$genotype =
  paste(colData(blrs_grant_with_doubles_fltr)$genotype1,
        colData(blrs_grant_with_doubles_fltr)$genotype2,
        sep = "_") %>%
  str_remove('_$') %>%
  factor() %>%
  relevel(ref = "CNAG_00000") %>%
  droplevels()

colData(blrs_grant_with_doubles_fltr)$libraryProtocol =
  colData(blrs_grant_with_doubles_fltr)$libraryProtocol %>%
  factor() %>%
  relevel(ref = "SolexaPrep") %>%
  droplevels()

mm = model.matrix(~libraryProtocol+libraryDate+genotype,
                    extractColData(blrs_grant_with_doubles_fltr))

min_new_protocol = min(
  as.Date(
    (unique(
      pull(
        filter(extractColData(blrs_grant_with_doubles_fltr),
               libraryProtocol == 'E7420L'), libraryDate)))))

mm = mm[,-which(colnames(mm) == paste0('libraryDate',min_new_protocol))]
```

**inspect the model matrix for linear dependencies**

```
lin_dep_report = caret::findLinearCombos(mm)
lin_dep_report
```

**add the design to the object**

```
design(blrs_grant_with_doubles_fltr) = mm
rm(mm)
```

**Write out**

```
if(WRITE_OUT){

  outpath = file.path(DDS_OUTPUT_DIR,
                      "90minDataFreeze",
                      TODAY,
                      "grant_doubles_input.rds")

  dds = DESeq2::DESeqDataSetFromMatrix(
    colData = extractColData(blrs_grant_with_doubles_fltr),
    countData = counts(blrs_grant_with_doubles_fltr),
    design = design(blrs_grant_with_doubles_fltr))

  sizeFactors(dds) = sizeFactors(blrs_grant_with_doubles_fltr)

  write_rds(dds, outpath)
}
```

**All**

This set is all samples – singles and doubles – which are in the 90minuteInduction conditions.

**Create set and quality filter**

```
blrs_90min_all = createExperimentSet(blrs, 'ninetyMin_all')

# note that this filters out those samples which failed QC1,
# but but does not filter on RLE unless the argument rle_iqr_threshold
# is set to a numeric value
blrs_90min_all_fltr = qaFilter(blrs_90min_all)

# remove WT which fall on dates with no perturbed samples
blrs_90min_all_fltr =
  filterWtByExperimentalLibdate_90min(blrs_90min_all_fltr)
```

### Add the design

```
blrs_90min_all_fltr = estimateSizeFactorsByProtocol(blrs_90min_all_fltr)

min_libdate = min(as.Date(colData(blrs_90min_all_fltr)$libraryDate))

colData(blrs_90min_all_fltr)$libraryDate =
  colData(blrs_90min_all_fltr)$libraryDate %>%
  relevel(ref = as.character(min_libdate)) %>%
  droplevels()

# add a 'genotype' column which is a concatenation of genotype1 and genotype2
colData(blrs_90min_all_fltr)$genotype =
  paste(colData(blrs_90min_all_fltr)$genotype1,
        colData(blrs_90min_all_fltr)$genotype2,
        sep = "_") %>%
  str_remove('_$') %>%
  factor() %>%
  relevel(ref = "CNAG_00000") %>%
  droplevels()

colData(blrs_90min_all_fltr)$libraryProtocol =
  colData(blrs_90min_all_fltr)$libraryProtocol %>%
  factor() %>%
  relevel(ref = "SolexaPrep") %>%
  droplevels()

mm = model.matrix(~libraryProtocol+libraryDate+genotype,
                  extractColData(blrs_90min_all_fltr))

min_new_protocol = min(
  as.Date(
    (unique(
      pull(
        filter(extractColData(blrs_90min_all_fltr),
               libraryProtocol == 'E7420L'), libraryDate)))))

mm = mm[,-which(colnames(mm) == paste0('libraryDate',min_new_protocol))]
```

#### inspect the model matrix for linear dependencies

```
lin_dep_report = caret::findLinearCombos(mm)
lin_dep_report
```

#### add the design to the object

```
design(blrs_90min_all_fltr) = mm
rm(mm)
```

#### Write out

```
if(WRITE_OUT){

  outpath = file.path(DDS_OUTPUT_DIR,
                      "90minDataFreeze",
                      TODAY,
                      "90min_all_input.rds")

  dds = DESeq2::DESeqDataSetFromMatrix(
    colData = extractColData(blrs_90min_all_fltr),
    countData = counts(blrs_90min_all_fltr),
    design = design(blrs_90min_all_fltr))

  sizeFactors(dds) = sizeFactors(blrs_90min_all_fltr)

  write_rds(dds, outpath)
}
```

### Environmental Perturbation

#### Create Sets

```
ep_list = list(
  wt = createExperimentSet(blrs, 'envPert_epWT'),
  titration = createExperimentSet(blrs, 'envPert_titrationWT'),
  perturbed = createExperimentSet(blrs, 'envPert_perturbed')
)

# NOTE!! AS OF 20220201 CNAG_03894 forms a linear depdendence with both
# concat treatment and libraryDate columns. it is being removed here to solve
# that issue

ep_list$perturbed = ep_list$perturbed[,ep_list$perturbed$genotype1 != "CNAG_03894"]
```

#### Quality Filter

```
ep_list_qc_passing = map(ep_list, qaFilter)
```

### Expression Filter

```
# How this is done is up to you, and obviously affects what genes are left in.
# Below is an example. You need to think about the thresholds and filter method
# that suits your purpose best.

expr_fltr_list = map(ep_list, ~rowSums(edgeR::cpm(counts(.))>4) >= 4)

ep_list_qc_passing_fltr = map2(ep_list_qc_passing, expr_fltr_list, ~.x[.y,])
```

### Set Design

```
setBaseConcatTreatmentBaseCond = function(ep_set, concat_base_cond){
  colData(ep_set)$concat_treatment =
    relevel(colData(ep_set)$concat_treatment, ref = concat_base_cond)

  colData(ep_set) = droplevels(DataFrame(colData(ep_set)))

  ep_set

}

setEpDesign = function(ep_set, design){
  design(ep_set) = design

  ep_set
}

concat_base_cond_list = list(

  wt = "YPD_noAtmosphere_30_noTreatment_noTreatmentConc_noPH_30",
  titration = "RPMI_noAtmosphere_30_noTreatment_noTreatmentConc_noPH_30",
  perturbed = 'PBS_noAtmosphere_30_noTreatment_noTreatmentConc_noPH_0'
)

ep_designs = list(
  wt = formula(~libraryDate + concat_treatment),
  titration = formula(~libraryDate + concat_treatment),
  perturbed = formula(~libraryDate + concat_treatment + genotype1)
)

ep_list_qc_passing_fltr = map2(ep_list_qc_passing_fltr,
                               concat_base_cond_list,
                               setBaseConcatTreatmentBaseCond)

ep_list_qc_passing_fltr = map2(ep_list_qc_passing_fltr,
                               ep_designs,
                               setEpDesign)
```

### Coerce back to DeseqDataObjects for proessing

```
ep_dds_list = map(ep_list_qc_passing_fltr, coerceToDds)

DDS_OUTPUT_DIR = "/mnt/scratch/rnaseq_pipeline/experiments/epTally"

if(WRITE_OUT){
  today = format(lubridate::today(),"%Y%m%d")

  output_dir = file.path(DDS_OUTPUT_DIR, today)

  dir.create(output_dir, recursive=TRUE)

  map(names(ep_dds_list), ~write_rds(ep_dds_list[[.]],
                                     file.path(output_dir,
                                               paste0("ep_",.,".rds"))))
}
```

### Copy the HTCF DESeq scripts to the working directory

These scripts will run DESeq in parallel on HTCF. The only items you'l need to edit is the path to the lookup file (if you are running more than one model), and to the dds_input in deseq_mpi.sh. If you do not keep deseq_de.R in the same directory as deseq_mpi.sh, then you'll need to update the path to the deseq_de.R script, also.

```
if(WRITE_OUT){
  htcf_deseq_scripts = c(system.file('bash',
                              'htcf_parallel_deseq.sh',
                              package = "brentlabRnaSeqTools"),
                         system.file('R_executable',
                                 'deseq_de.R',
                                  package = "brentlabRnaSeqTools"))

lapply(htcf_deseq_scripts, file.copy, to = DDS_OUTPUT_DIR)
}
```

### After DESeq(), Calculate RLE

### Environmental Perturbation

### Load in the post DESeq() data sets

Note: this entirely depends on what set you want to look at – in this example, I am doing the sample agreement and tallies for the EP sets

```
DESEQ_OUTPUT_LIST = list(
  ep_wt =
"/mnt/scratch/rnaseq_pipeline/experiments/epTally/20220201/ep_wt_20220201_output.rd
s",
  ep_titration =
"/mnt/scratch/rnaseq_pipeline/experiments/epTally/20220201/ep_titration_20220201_ou
tput.rds",
  ep_perturbed =
"/mnt/scratch/rnaseq_pipeline/experiments/epTally/20220201/ep_perturbed_20220201_ou
tput.rds"
)
```

### Environmental Perturbation

### Create the set

```
# get deseq output object
ep_dds_list = list(
  ep_wt = readRDS(DESEQ_OUTPUT_LIST$ep_wt),
  ep_titr = readRDS(DESEQ_OUTPUT_LIST$ep_titration),
  ep_pert = readRDS(DESEQ_OUTPUT_LIST$ep_perturbed)
)
```

### add a column describing replicate groups if necessary

```
ep_dds_list$ep_pert$rep_col =
  paste(ep_dds_list$ep_pert$genotype1,
        ep_dds_list$ep_pert$concat_treatment, sep = "_")
```

### calculate RLE

```
ep_rle_list = list(
  ep_wt = removeLibdateByReplicate(ep_dds_list$ep_wt, "concat_treatment"),
  ep_titr = removeLibdateByReplicate(ep_dds_list$ep_titr, "concat_treatment"),
  ep_pert = removeLibdateByReplicate(ep_dds_list$ep_pert, "rep_col")
)
```

### Update the database

```
if(UPDATE_DB){

 updateRleTable = function(set_name, set_rle_summary){
   iqr_col = paste0(set_name, "_iqr")
   med_dev_col = paste0(set_name, "_median_dev")

   update_df = set_rle_summary %>%
   dplyr::select(replicateAgreementNumber, rle_iqr, rle_median_deviation) %>%
   dplyr::rename(!!quo_name(iqr_col) := rle_iqr,
                 !!quo_name(med_dev_col) := rle_median_deviation)


 res = patchTable(
  database_info$kn99$urls$replicateAgreement,
  Sys.getenv("kn99_db_token"),
  update_df,
  "replicateAgreementNumber")

 res
 }

res_list = map(names(ep_rle_list), ~updateRleTable(., ep_rle_list[[.]]
$without_libdate_effect$summary))

# check status ----
## failures should only be those without enough replicates to calculate RLE
extract_status = map(res_list, ~map(., ~as.numeric(.$status_code)))

extract_status[[1]][unlist(extract_status[[1]]) != 200]
extract_status[[2]][unlist(extract_status[[2]]) != 200]
extract_status[[3]][unlist(extract_status[[3]]) != 200]
}
```

## B7: Tally Experiment Sets

### Introduction

The purpose of tallying the experiment sets is to track the development of a given set. These are meant as examples – if you are the analyst consuming a given data set, it is a good idea to figure out how to tally your set to see how close it is to being done.

For each set (eg, the Environmental Perturbation), if you extract the code and paste it into a clean notebook, a somewhat formatted html document will be created. You can publish this to Rpubs, a free server for rendered Rmd notebooks, which is a nice way of sharing the set progress as well as a method of tracking progress over time, as the rendered results are saved on the Rpubs server.

### Setup

```
library(brentlabRnaSeqTools)
library(rtracklayer)
library(tidyverse)

# set variables

KN99_GFF_RDS = Sys.getenv("kn99_stranded_gff_rds")
DB_USERNAME = Sys.getenv("db_username")
DB_PASSWORD = Sys.getenv("db_password")

# note: I mount to the cluster and output directly to it
DDS_OUTPUT_DIR = "."

# controls whether dds objects are written
WRITE_OUT = FALSE
```

### pull the database as a brentlabRnaSeqSet object

Note: you really only need the metadata for this task. You could use `getMetadata`.

```
blrs = brentlabRnaSeqSetFromDatabase('kn99',DB_USERNAME, DB_PASSWORD)
```

### Add gene level data (optional)

this adds all of the data regarding each locus as a GRange object to the gene data slot of the brentlabRnaSeqSet object. Useful if you are going to use other Bioconductor packages.

```
kn99_gff = readRDS(KN99_GFF_RDS)

kn99_genes = kn99_gff[kn99_gff$ID %in% rownames(blrs)]

rowRanges(blrs) = kn99_genes[order(match(kn99_genes$ID,rownames(blrs)))]

rownames(blrs) = rowData(blrs)$ID
```

### Environmental Perturbation

### Create Sets

```
ep_list = list(
  wt = createExperimentSet(blrs, 'envPert_epWT'),
  titr = createExperimentSet(blrs, 'envPert_titrationWT'),
  pert = createExperimentSet(blrs, 'envPert_perturbed')
)

# NOTE!! AS OF 20220201 CNAG_03894 forms a linear depdendence with both
# concat treatment and libraryDate columns. it is being removed here to solve
# that issue

ep_list$pert = ep_list$pert[,ep_list$pert$genotype1 != "CNAG_03894"]


ep_list_qc_passing = map(ep_list,qaFilter)

ep_list_qc_passing_with_iqr = map(names(ep_list),
                         ~qaFilter(ep_list[[.]],1,paste0("ep_",., "_iqr")))
names(ep_list_qc_passing_with_iqr) = names(ep_list)




condition_lists = list(
  wt = alist(medium,
             atmosphere,
             temperature,
             timePoint,
             treatment,
             treatmentConc,
             pH),
  titr = alist(medium,
               atmosphere,
               temperature,
               timePoint,
               treatment,
               treatmentConc,
               pH),
  pert = alist(genotype1,
               medium,
               atmosphere,
               temperature,
               timePoint,
               treatment,
               treatmentConc,
               pH))

ep_tally_list = map2(names(ep_list), condition_lists,
                     ~createEPTally(ep_list[[.x]],
                                    ep_list_qc_passing[[.x]],
                                    ep_list_qc_passing_with_iqr[[.x]],
                                    .y))
names(ep_tally_list) = names(ep_list)


names(ep_list_qc_passing_with_iqr) = names(ep_list)

# write_csv(env_pert_tally,
#           '../../../datafreeze_202111/data/env_pert_tally_20211122.csv')
#
# # add those samples with less than 2 replicates
# iqr_fltr_rle_summary_mod = removed_effect_rle_summary %>%
#   filter(INTERQUARTILE_RANGE<iqr_threshold | is.na(INTERQUARTILE_RANGE))
#
# write_csv(iqr_fltr_rle_summary_mod,
#           '../../../datafreeze_202111/data/iqr_fltr_rle_summary_20211122.csv')
```

### ep_wt tallies

```
tally_type_list = c('unfiltered_tally',
                    'qc1_passing_tally',
                    'iqr_filter_qc1_passing_tally')

ep_tally = map(ep_tally_list, ~map(tally_type_list, reshapeEnvPertTallies, .))

ep_tally = map(ep_tally, setNames, tally_type_list)
```

### RLE results

### Norm Counts Iqr

```
# note that this function requires that ep_list and ep_list_qc_passing_with_iqr
# be in the namespace already (see the setup section)
# TODO: save the norm count rle from the sample agreement step for this function
cumDistIqr = function(set_name){
  plot(ggplot() +
       stat_ecdf(data = norm_count_rle),
                 aes(!!rlang::sym(paste0("ep_",set_name, "_iqr"))),
                 color = 'orange') +
       stat_ecdf(data = extractColData(ep_list_qc_passing_with_iqr[[set_name]]),
                 aes(!!rlang::sym(paste0("ep_", set_name, "_iqr"))),
                 color = "blue") +
       ggtitle("orange = normalized counts; blue = libdate_model_removed_libdate"))
+
  scale_x_continuous(limits = c(0,1), breaks = seq(0,1,.05))+
  scale_y_continuous(limits = c(0,1), breaks = seq(0,1,.05))
}
```

### Environmental Perturbation – WT

#### unfiltered tallies

```
ep_tally$wt$unfiltered_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

#### qc1 passing

```
ep_tally$wt$qc1_passing_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

#### iqr filtered

```
ep_tally$wt$iqr_filter_qc1_passing_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

### Environmental Perturbation – Perturbed

#### unfiltered tallies

```
ep_tally$pert$unfiltered_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

### qc1 passing

```
ep_tally$pert$qc1_passing_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

### iqr filtered

```
ep_tally$pert$iqr_filter_qc1_passing_tally  %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

### Titration Tallies – WT

### unfiltered tallies

```
ep_tally$titr$unfiltered_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

### qc1 passing

```
ep_tally$titr$qc1_passing_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```

### iqr filter

```
ep_tally$titr$iqr_filter_qc1_passing_tally %>%
  DT::datatable(
    options = list(
    pageLength=50,
    scrollX='400px',
    fixedColumns = list(leftColumns = 2),
    autoWidth = TRUE)
)
```