Title
Something new under the sun: transposon sequencing and phylogenomics shed light on unstudied photosynthetic genes

Author
Johnson, Jeffrey

Publication Date
2019

Peer reviewed|Thesis/dissertation

Something new under the sun: transposon sequencing and phylogenomics shed light on unstudied photosynthetic genes

By

Jeffrey Johnson

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Microbiology

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Krishna Niyogi, Chair

Professor Arash Komeili

Professor David Savage

Professor Bob Buchanan

Summer 2019

Abstract

Something new under the sun: transposon sequencing and phylogenomics shed light on unstudied photosynthetic genes

by

Jeffrey Johnson

Doctor of Philosophy in Microbiology

University of California, Berkeley

Professor Krishna Niyogi, Chair

The core machinery required for oxygenic photosynthesis is well conserved from cyanobacteria to land plants and in various algal clades, and has been studied intensely for decades. Much remains to be learned, however, both about the regulation and assembly of the core machinery, and about a host of optional factors associated with it only in particular lineages or under particular conditions.

I sought to identify unstudied genes required for long-term acclimation to high light intensity in the cyanobacterium *Synechococcus elongatus* PCC 7942 by simultaneously comparing the growth rates of mutants in every nonessential gene in the genome under low, normal, and high light via RB-TnSeq (growth of a pooled transposon library, followed by sequencing and quantification of DNA barcodes). This was the first study to investigate low and high light conditions with high enough resolution to assign fitness values to nearly every gene. It confirmed that high light mainly damages the Photosystem II reaction center, and that synthesis and recycling of chlorophyll from damaged PSII subunits is essential for acclimation. It also identified several unexpected genes with large fitness effects. The include a mysterious intrinsically disordered protein with homology to phasins, an unusual transcriptional regulator that may physically protect DNA from oxidative damage, and an ABC transporter that takes up amino acids.

Next I grew the transposon library under a range of fluctuating light conditions to determine which genes are important for fitness only when fluctuations include higher highs or lower lows, alternate quickly or slowly, include gradual or sudden low-to-high transitions, or mimic the patterns found in dense bioreactor cultures. Many genes were beneficial only under certain conditions but not others, and some were also found to be detrimental (knocking them out increased fitness) under certain conditions. The genes identified are implicated in diverse cellular processes relevant to photosynthesis, including: the stringent response to dark periods, alternative electron sinks (plastoquinone reduction, glycogen synthesis) needed during sudden shifts to high light, and regulation of transcription.

As part of a separate line of inquiry into genes conserved across photosynthetic species but missing from other branches of life, I also developed a scripting language that partially automates comparisons between large numbers of genomes using a variety of popular sequence search and ortholog-finding programs: BLAST, BLAST reciprocal best hits, HMMER, DIAMOND, MMSeqs2, OrthoFinder, SonicParanoid, and others. It reproducibly installs all the programs and runs them as needed to perform searches as described in a concise, human-readable domain-specific language designed to facilitate sharing and incremental improvement of the search algorithms.

To Grandma Dorothy, who didn't quite get to see me graduate.

Contents

Listing of figures

Acknowledgments

I'm grateful to my committee members: Bob Buchanan, Arash Komeili, and Dave Savage. You've been extremely supportive and flexible. And especially to my mentor Kris Niyogi! You stuck with me all this time despite a couple really bizarre plot twists along the way, and were always helpful. I'm a much better scientist now than I would be without your influence.

Thanks to everyone in the Niyogi lab, past and present, for your help over the years. From way back in my rotation days with Jose and Patrick all the way up to the last-minute revisions of this dissertation! Especially fresh in my mind now are Chris Baker, Chris Gee, Robbie, Patrick, Daniel, Tim, ... and of course Marilyn.

I also want to thank everyone who helped me figure out how to science during undergrad: Michal Galdzicki, Herbert Sauro, Chris Amemiya. Even Mr. Rapin, all the way back in high school. You made biology seem like a pretty cool field to go into and that was when I started to think about doing the grad school thing.

Thanks Mom and Dad, Brian, and Hillary! For understanding when I had to work during visits home, or sometimes wouldn't answer texts, or forgot about dates. Thanks to Michael Gomez who's been my friend and roommate throughout most of grad school. I hope we can still keep in touch when you move back to L.A. and I go wherever I end up going.

And finally, special thanks to Rocio for helping out with what must have been a maddening number of little delays and administrative snafus.

I couldn't have done it without all of you, and imagine I might still need more help after graduating. Thanks for being wonderful people! I'm sorry if I didn't get your name down. It's not that I don't appreciate you, just that I'm sleep deprived and still need to do one very important last-minute task: file my dissertation an hour before the deadline. I'll get you a beer or something in person!

Chapter 1

RB-TnSeq to identify genes required under high light

1.1 Summary

I grew a randomly barcoded library of cyanobacterial transposon mutants under low, medium, and high light to determine which genes are required for normal acclimation to high light stress. It revealed that the most critical genes are related to repair of damaged photosystem II. Specifically, defects in the pathway responsible for synthesizing (or recycling) chlorophyll and inserting it into newly made D1 reaction center subunits have large fitness costs. Several other genes with large effects on fitness were also identified, whose roles are not obvious. These include an intrinsically disordered phasin-like protein, an ABC transporter operon, an unusual transcription factor, and a GAF histidine kinase which help orchestrate the regulatory response to high light.

1.2 Methods

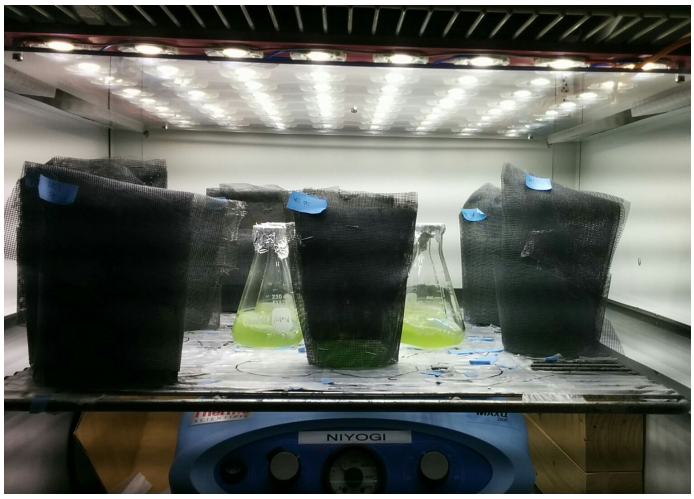
My first attempt failed during the final analysis due to low barcode diversity—the library had experienced a selection bottleneck during a previous outgrowth in another lab.

1.2.1 Experimental design

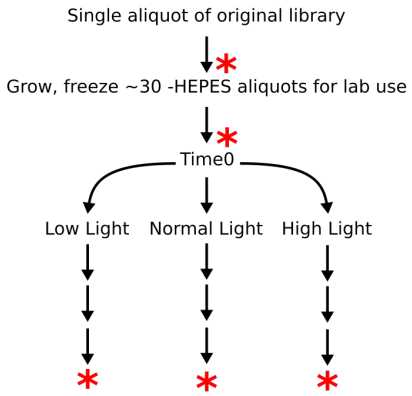
I repeated this experiment twice. The first run used light levels ranging from "low low light" (LLL, ~10 μmol photons m^{-2} sec^{-1}) to "high high light" (HHL, ~450 μmol photons m^{-2} sec^{-1}), with three replicates each. FEBA quality controls (Section 1.2.5) flagged the Time 0 data as having low barcode diversity, so I repeated it starting from one of the aliquots from the original study[48]. The second run used a simpler design with more replicates: four flasks each at low, medium, and high light (7, 70, and 700 μmol photons m^{-2} sec^{-1}). All the results presented are from the second run, with the exception of Figure 1.15.

1.2.2 Time 0 Outgrowth

After receiving my second copy of the TnSeq library from the Susan Golden lab at UC San Diego, I did an initial outgrowth under standard conditions as described in the Rubin paper[48]. Briefly: the inital aliquot was thawed, diluted 1.5mL into 300mL BG11, split between 6 250mL flasks, allowed them to recover for 24 hours under 70



(a) Growth setup



(b) Outgrowths and sampling

Figure 1.1: Design of the TnSeq high light experiment. **a.** Four replicate flasks per treatment were exposed to 700 µmol photons m^{-2} sec^{-1} from a custom LED panel (JBeamBio) with cool white LEDs (BXRA-56C1100-B-00, Farnell). HL flasks were left uncovered; NL and LL were covered in several layers of shading mesh until they received approximately 70 µmol photons m^{-2} sec^{-1} and 7 µmol photons m^{-2} sec^{-1} respectively. **b.** Arrows represent one outgrowth, followed by dilution and freezing samples for later DNA extraction. Red asterisks represent samples that were sequenced.

µmol photons m^{-2} sec^{-1} with no shaking, grown with shaking for 4 days, and finally pooled into one Time 0 sample. All cultures were grown in air at 30°C with shaking at 100rpm. The only change in outgrowth conditions was use of pH buffered media (+HEPES) for the first few days. Then the library was split into +/- HEPES treatments and both were sequenced on a MiSeq sequencer (Illumina). -HEPES resulted in slightly improved barcode diversity (data not shown here, but included in the pipeline), so it was used for all the TnSeq experiments. The Time 0 sample was spun down at 4000rcf and split into about 30 aliquots, which were frozen for later library prep and sequencing. The rest was diluted to OD_{750} 0.025 and used immediately for the first experimental outgrowth. The recovery process was repeated for each later Time 0 outgrowth in Chapter 2.

1.2.3 Experimental Outgrowths

Each day I took a 1mL sample from each flask and measured their OD_{750} on an M-1000 infinite plate reader (Tecan). I also took full absorbance scans from 400-800nm in order to monitor the bulk state of the cultures. Because the plate reader has difficulty reading low-density samples, I also measured them manually using a spectrophotometer at 750nm. The manual readings are used in Figure 1.2.

I spun down samples and diluted cultures every few days when they reached an OD_{750} of around 0.25-0.3 to minimize self-shading, stopping the final outgrowth of each treatment when it had doubled roughly 7 times (Figure 1.2). The exact number does not seem to be important, but having them consistent simplifies the data analysis because the proportion of counts represented by each strain does not need to be corrected for number of generations grown.

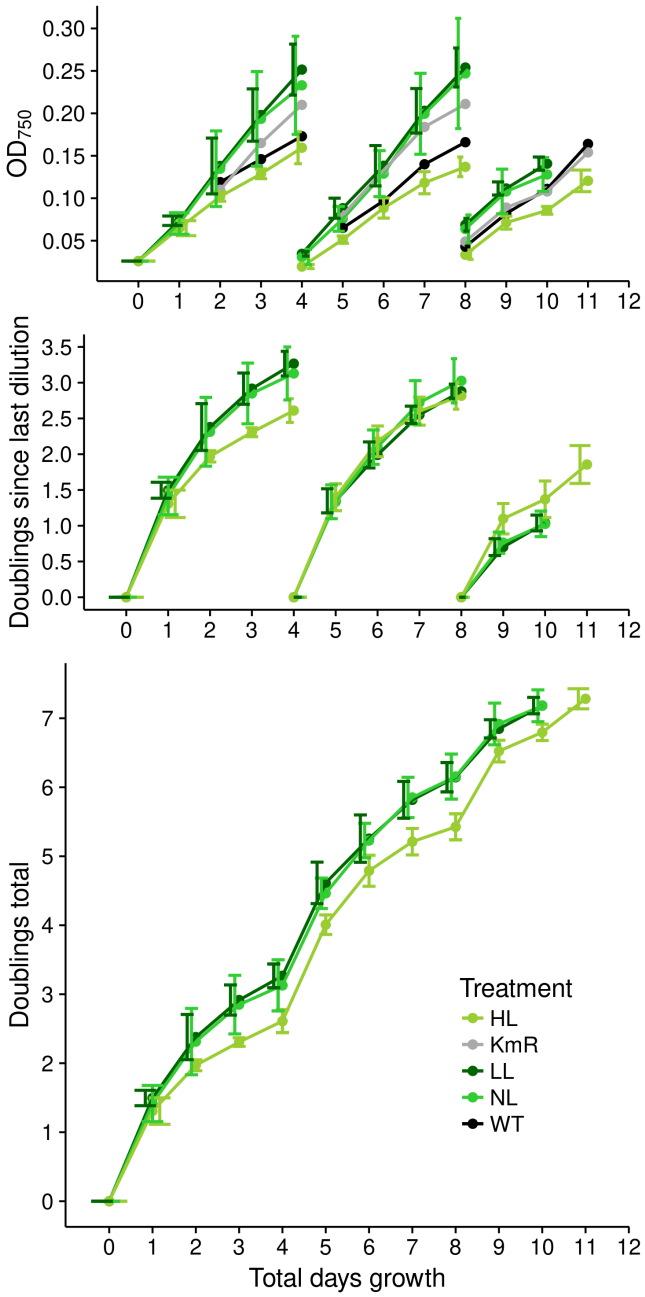


Figure 1.2: OD_{750} of TnSeq cultures during the high light outgrowth. Optical density (top) was measured each day at 750nm to avoid overlap with chlorophyll fluorescence. Cultures were diluted when they reached an OD_{750} of around 0.25-0.3 to minimize self-shading. Number of doublings of the bulk culture since each dilution (middle) was used to calculate total doublings (bottom) and stop each treatment after 7 generations. A flask of *S. elongatus* and one of the kanamycin resistance control strain were also grown under the same conditions (top graphs only) to verify that the TnSeq cultures behaved normally .

1.2.4 Library Prep

The standard RB-TnSeq library prep[62] consists of DNA extraction followed by one PCR with a set of primers that amplify the barcodes and add indexed Illumina adapters for sequencing. I used a 2-PCR variant that makes use of universal adapter stubs for compatibility with multiple Illumina index sets (Figure 1.3). After optimizing the new prep (Figure 1.4), both preps were sequenced to compare data quality (data not shown, but included in the pipeline), and the new prep was used for all further experiments.

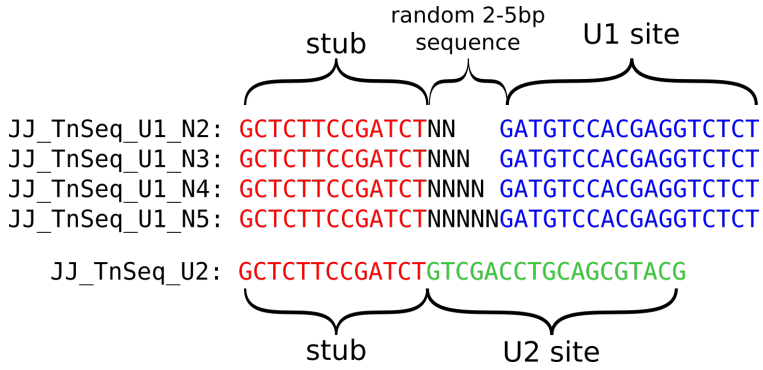


Figure 1.3: Primers for the PCR prep, designed by Shana McDevitt at the Vincent J. Coates sequencing core facility. The U1 and U2 sites are standard for BarSeq experiments, and the stubs are standard for Illumina adapters. They improve on the standard BarSeq primers in two ways: first, the random sequence helps distinguish clusters during the first step of Illumina sequencing, and second, the universal stubs can be extended by a second PCR into any full-length Illumina adapters to fit in with different sequencing schemes. U2 is a single primer, while U1 is an equimolar mix of the 4 shown.

1.2.5 Data analysis

After sequencing, my analysis began with a pipeline based on FEBA, a set of scripts first developed for the original BarSeq paper[62], and now maintained (as of 2019) by Morgan Price in the Arkin lab.

The initial pipeline (FASTQ files to differential fitness plots) is fully reproducible. An archive of all the necessary software is provided on[25] and[26]. These commands will download and run it using the Nix package manager[14] on a Linux or MacOS system:

```
git clone https://bitbucket.com/jefdaj/tnseq7942-hl.git
cd tnseq7942-hl
nix-build --verbose -j$(nproc) --option build-use-chroot false --keep-going
```

Note that my second set of TnSeq experiments extended this same pipeline with fluctuating light data. See the code in Section 2.2.5 for both.

The pipeline begins with fastqfix.py (Figure 1.5) discarding any reads that do not contain the expected flanking sequences, flipping the ones that ended up backward relative to how FEBA expects them to be aligned, and trimming them to the start of the forward barcode.

Next, the pipeline runs three FEBA scripts: MultiCodes.pl, CombineBarseq.pl, and BarSeqR.pl. The scripts themselves are unmodified, but I run them in a reproducible Nix environment rather than the standard method of installing ad-hoc Perl and R packages, and modify the input metadata slightly: the scripts expect a spreadsheet defining experimental treatments in a format tailored to the Arkin and Deutschbauer labs' standard workflow involving large numbers of bacterial strains and small molecule stressors. I adapted it to work instead with multiple

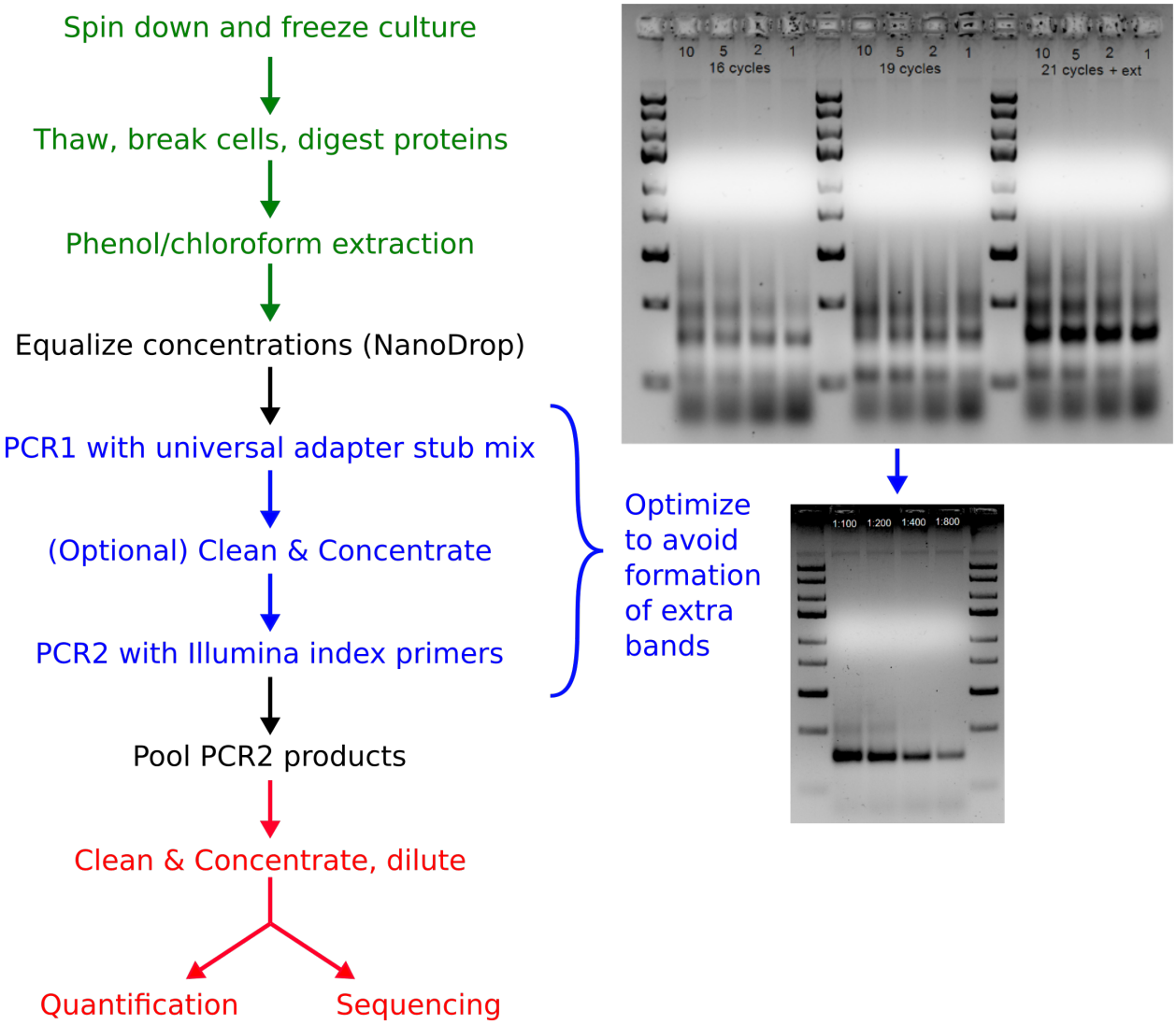


Figure 1.4: The complete PCR library prep. The green section is a standard DNA extraction from *S. elongatus*[12], black steps would need to be done for a standard BarSeq prep, and the blue section is new. Red is the standard sequencing protocol and depends on the core facility involved. Quantification was done by Bioanalzyer at the QB3 Functional Genomics facility, and sequencing by HiSeq4000 at the QB3 Vincent J. Coates sequencing facility. Gels show representative PCR2 quality before and after optimization (top and bottom respectively). The third lane in the bottom labeled 1:400 was the version sequenced.

outgrowths of the same bacteria starting from various timepoints, needed for the fluctuating light experiments in Chapter 2.

BarSeqR.pl generates a large number of quality control reports with an HTML index explaining them. Most importantly it breaks experiments into those that succeeded and those that failed quality control, and generates a file `fit_logratios_good.tab` with normalized fitness values per gene for each successful experiment/treatment.

Per-strain fitness is defined as the log_2 fold-change in abundance of that strain's tag between the start and the

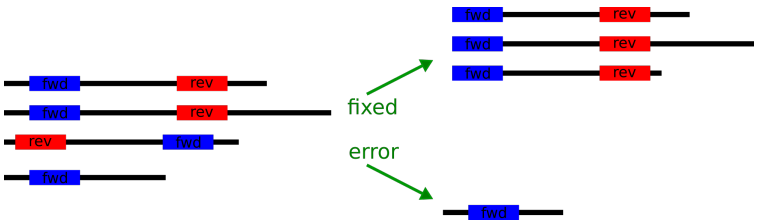


Figure 1.5: The fastqfix script formats PCR data for FEBA. Because the same universal adapter stub is used on both ends of my primers, roughly half the sequenced reads are backwards. This script fixes that. It also trims the leading sequences and removes reads missing the forward and reverse flanking sequences.

end of the outgrowth, with small constants e_{start} and e_{end} added to prevent the values becoming zero or infinite:

$$fitness = \log_2 \left(\frac{n_{end} + e_{end}}{n_{start} + e_{start}} \right) + C \tag{1.1}$$

Strain-level fitness values are averaged to create gene-level fitness values, then C is chosen to shift the peak of the gene fitness distribution to zero. The final value can intuitively be thought of as how many doublings ahead or behind the group mutants in a given gene were by the end of the growth competition. Quality controls include several common-sense checks of these values. For example, the first and second half of most genes should have similar fitness values when considered separately, because an insertion in either one will probably disrupt the same function. Most *S. elongatus* genes have roughly 30 to 50 insertions along their length, so the values are quite robust. Only the middle 80% of each gene is used because even essential genes may permit insertions near the ends[48], and very short genes without enough insertions are not considered further.

The last step in the pipeline is to generate my own reports, mostly by comparing the final fitness values in an R script. The pipeline is reproducible, but not fully automated; each new experiment requires a Nix file grouping FASTQ reads into treatments, an updated metadata spreadsheet to pair each treatment with its Time 0 control, and a new analysis script to make useful comparisons from the FEBA data.

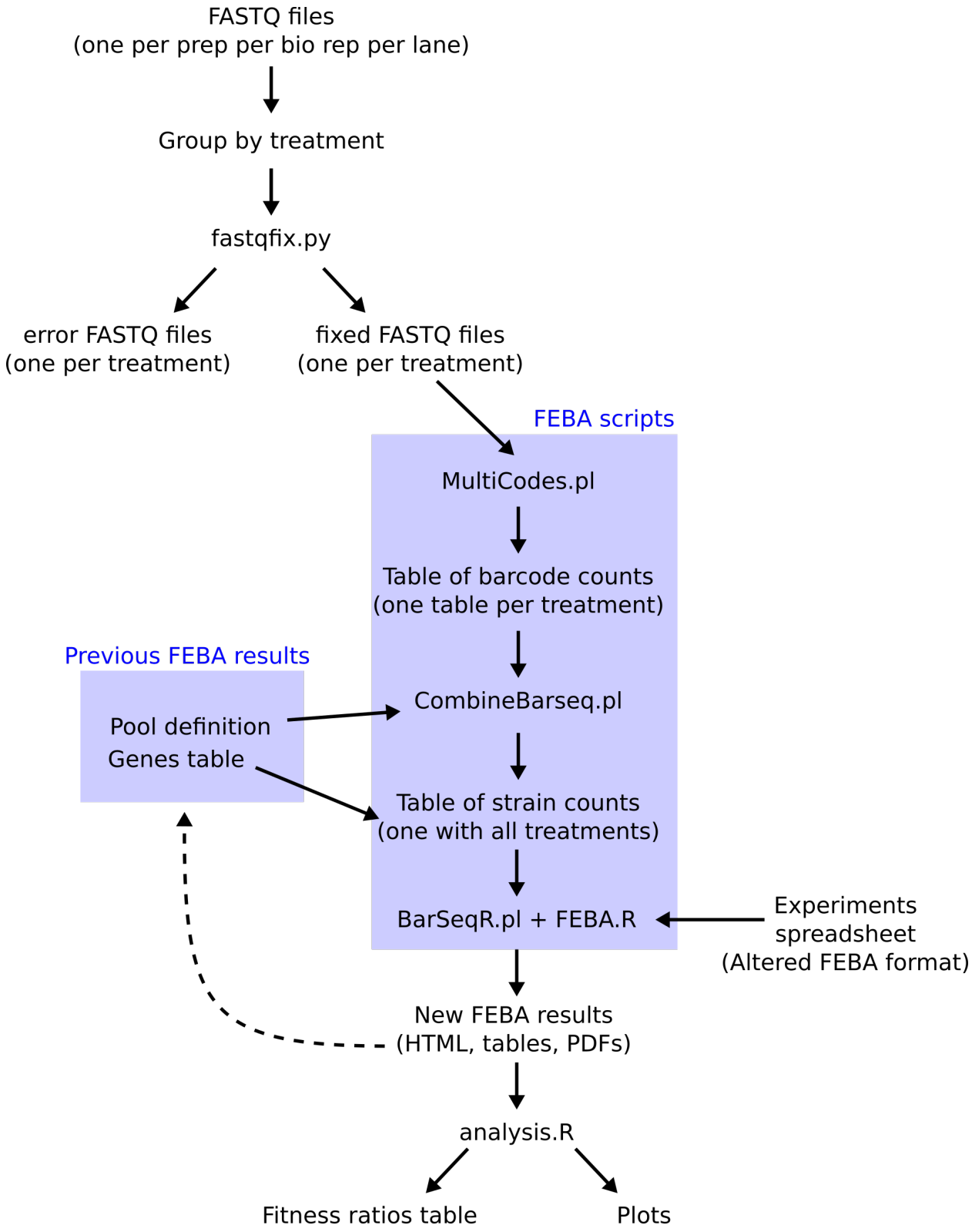


Figure 1.6: TnSeq analysis pipeline based on FEBA and adapted to the PCR prep. Available at `https://bitbucket.org/jefdaj/tnseq-hl`.

1.3 Results

1.3.1 Overview of top hits

Few fitness values differed significantly between the low and normal light treatments. However, comparing high to low (or normal) light yielded a large number of strong hits. Genes discussed here were picked by first considering all genes with a difference of at least 2 between HL and LL fitness ratios, then grouping them by annotation and/or known functions of their homologs in other organisms. An overview is given in Figure 1.7, and each labeled set of genes is discussed in the following sections. A spreadsheet of top hits is also included in Appendix A.

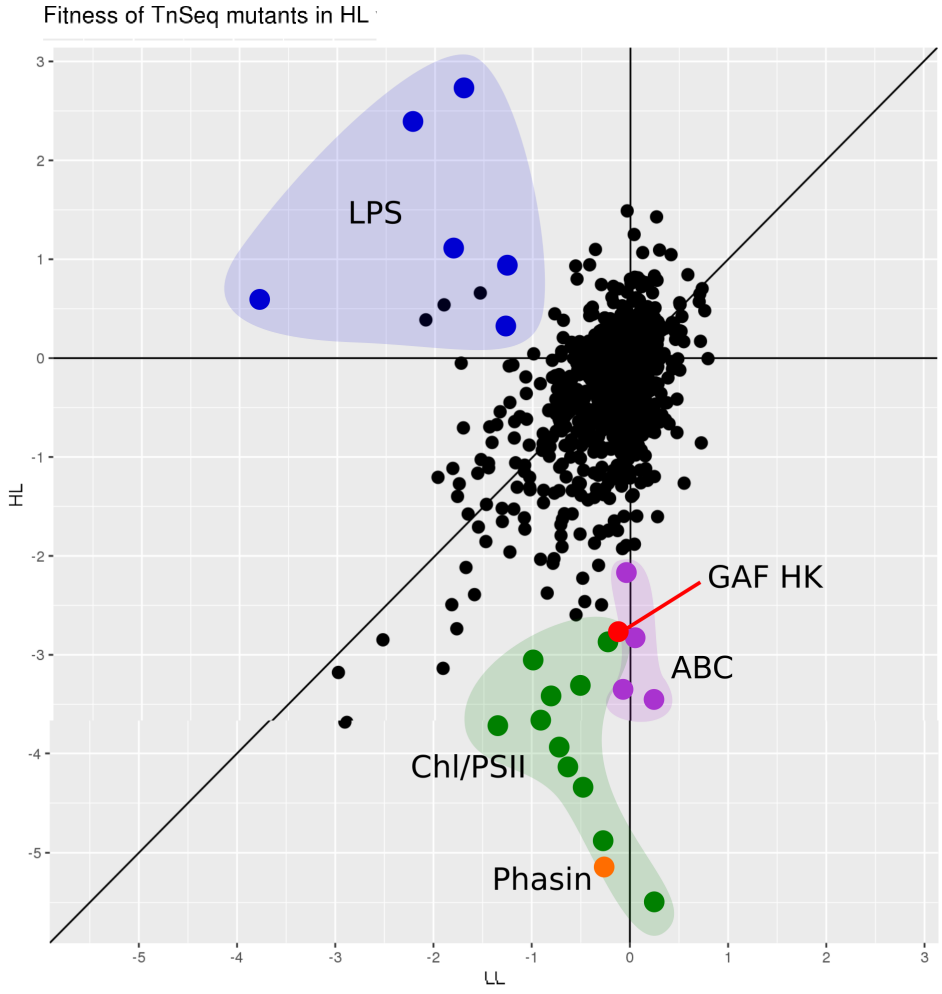


Figure 1.7: Overview of differential fitness under high vs low light. Shown are normalized fitness values (see Section 1.2.5 for details) of mutants in each gene for which a reliable value could be calculated. Genes toward the bottom are more important to fitness (more costly to interrupt) under high light, while those toward the left are more costly to interrupt under low light.

1.3.2 Chlorophyll biogenesis/recycling and photosystem stability

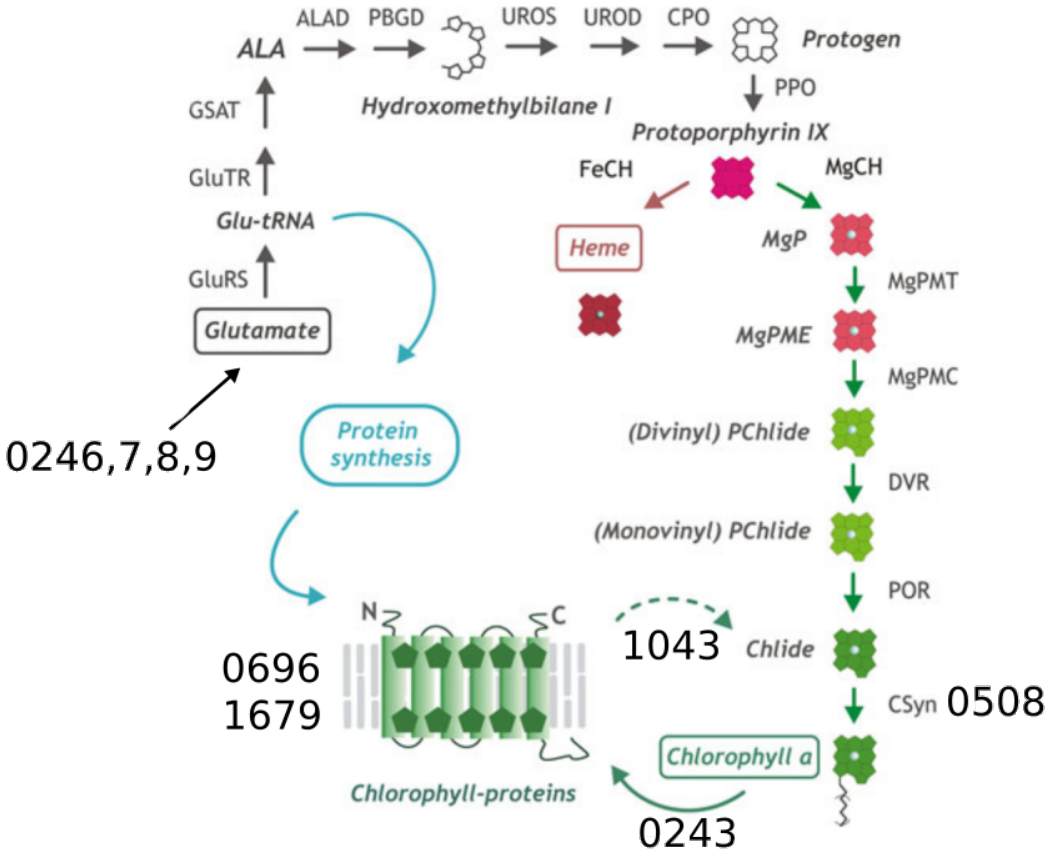


Figure 1.8: Hits in the tetrapyrrole biosynthetic pathway. Reproduced with permission from[56], overlayed with SynPCC7942_ locus IDs of TnSeq hits likely to function in the early or late steps of the pathway, or in stabilizing PSII while newly synthesized chl proteins are inserted. The Synpcc7942_0246,7,8,9 operon is discussed in Section 1.3.7.

Insertions in HliC (Synpcc7942_0243) caused the most severe defects; strains bearing them grew normally at low light but lagged behind by more than 5 generations on average at high light. HliC is one of the four classic cyanobacterial high-light inducible proteins (HLIPs). It has recently been determined to bind 4 chlorophyll and 2 β-carotene molecules, and is hypothesized to protect chlorophyll protein intermediates during synthesis[53]. As expected, other genes related to chlorophyll and PSII biogenesis affect fitness at high light as well. These genes are summarized in Table 1.1.

PsbT (Synpcc7942_0696) has been shown[39,38,40] to be involved in repair of photodamaged PSII in *C. reinhardtii*. It is associated with the Q_A binding pocket and hypothesized to help stabilize that region during PSII repair, speeding recovery[38] of electron transfer.

Locus ID	Annotation	Known or Predicted Phenotype
SynPCC7942_0243	HliC	Perturbed chlorophyll recycling, D1 synthesis
SynPCC7942_0508	Geranylgeranyl reductase	Buildup of geranylgeranylated chlorophyll
SynPCC7942_0696	PsbT	Inefficient repair of photodamaged PSII
SynPCC7942_1043	Phytol kinase	Buildup of geranylgeranylated chlorophyll
SynPCC7942_1679	PsbW	Defect in PSII dimerization
SynPCC7942_1793	Thioredoxin (trxA)	Impaired oxidative stress and redox regulation

Table 1.1: Hits related to chlorophyll biosynthesis and PSII repair.

PsbW (Synpcc7942_1679, also known as Psb28) is involved in PSII stability too. An Arabidopsis knock-down was found to have decreased PSII dimer stability at all light levels, which caused a growth phenotype only at high light[52]. It is degraded similarly to D1 but unlike D1 its degradation is phosphorylation-independent, leading to the hypothesis that it degrades when the PSII complex is destabilized during D1 repair. A knockout in *Synechocystis* has also been studied. It was found[49] to associate with the CP43-less PSII monomer, and the knockout caused a high-light specific growth defect. Most recently, it has also been found in another study using this same TnSeq library[47] to be important for protection against oxidative stress in *S. elongatus* during darkness.

Vavilin & Vermaas[60] used ^{13}C labeling to show that there is continuous turnover of chlorophyll in *Synechocystis*, especially from damaged PSII during high light, and that the recycled porphyrin rings, rather than *de novo* synthesis, are the major source of chlorophyll for PSII repair. This makes sense, as chlorophyll would typically survive much longer than the D1 protein before requiring degradation.

Although high light is mainly thought to damage PSII, there is some recent evidence that enzymes in the chlorophyll biosynthesis pathway are also required to prevent PSI damage. Interrupting the phytyl-phosphate kinase VTE6 (a homolog of Synpcc7942_1043) in Arabidopsis caused destabilization of the PSI complex in high light[61]. Phytol-phosphate kinase provides phytyl moieties for phylloquinone, part of the PSI reaction center, as well as for the antioxidant tocopherol. Since *S. elongatus* lacks tocopherol, PSI is the more likely target here. In *Synechocystis*, knocking out geranylgeranyl reductase (homolog of Synpcc7942_0508) caused instability and degradation of both photosystems, perhaps due to increased rigidity of the geranylgeranyl tails vs phytyl tails[61].

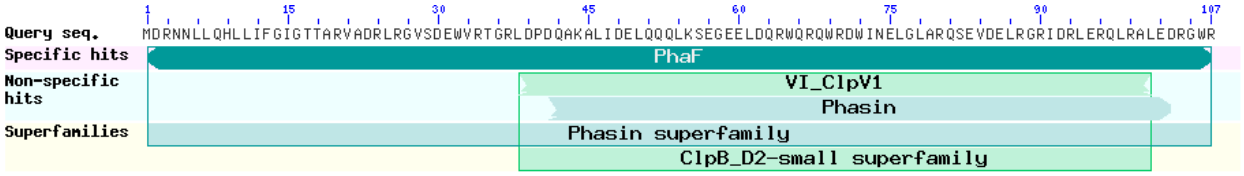
1.3.3 A disordered, phasin-like protein

Insertions in Synpcc7942_2355 show the second largest fitness defects after those in HliC. It is an uncharacterized protein with homology to PhaP, known[21] to coat hydrophobic polyhydroxyalkanoate (PHA) granules to prevent them merging or interacting with other cellular components in *Synechocystis*.

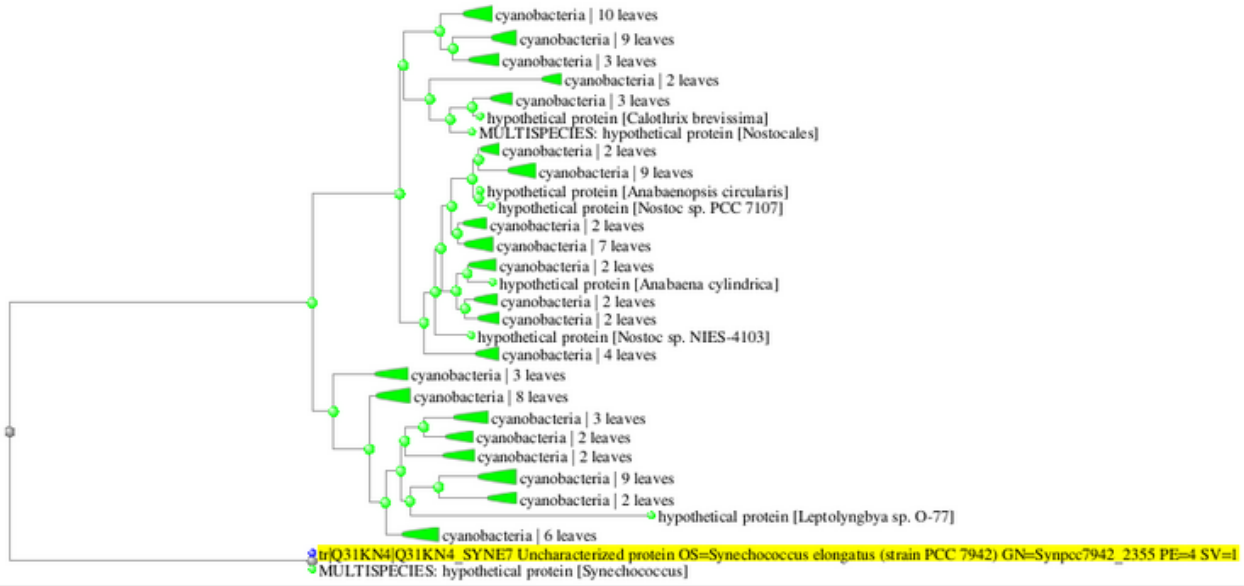
Many bacteria, including cyanobacteria, produce PHA granules. However *S. elongatus* has no predicted PHA synthase, and was found[7] not to produce PHA upon nitrogen starvation. Staining of wild type *S. elongatus* cells with Nile Blue A (Figure 1.10) did reveal foci similar in appearance to PHA granules in other species, but there are no obvious differences between high and low light.

One possibility is that *S. elongatus* makes PHA granules via a nonstandard synthase that has been misannotated or has multiple functionalities. Another, which would explain the lack of close homology to other cyanobacterial phasins (Figure 1.9) is that this is a different type of intrinsically disordered protein, perhaps also involved in chlorophyll biogenesis and PSII repair. It could be serving as a scaffold, increasing stability of an inter-

mediate complex, or even interacting with lipids in the thylakoid membrane. Biochemical studies will be required to elucidate its interaction partners and mechanism of action.



(a) Domain prediction (Phyre2)



(b) Conservation (NCBI BLAST)

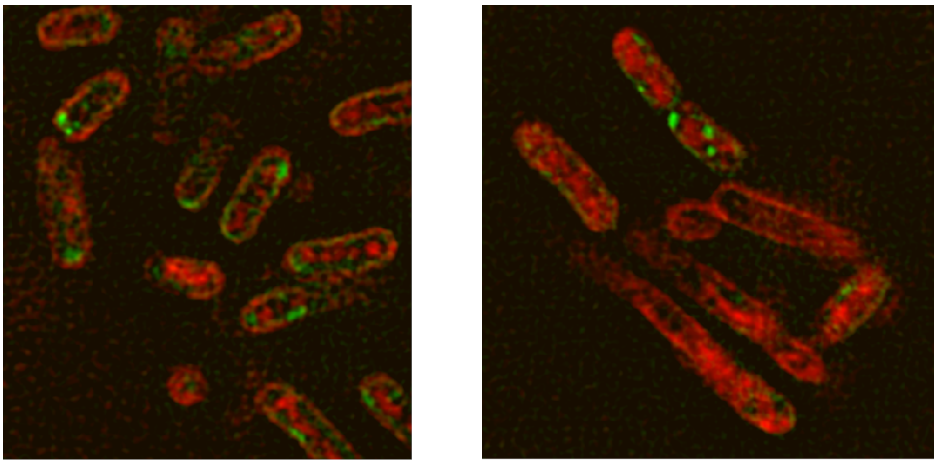
Figure 1.9: Weak structural and evolutionary evidence that Synpcc7942_2355 is a phasin.

1.3.4 Thiamine phosphate synthesis and the Calvin/Benson cycle

Synpcc7942_1057 is a thiamine-phosphate synthase. It is a required to make thiamine pyrophosphate, a cofactor of transketolase in both the oxidative pentose phosphate pathway and the Calvin/Benson cycle. Henkes *et al.* [22] found that a small decrease (20-40%) in transketolase activity almost completely inhibited photosynthesis at high light. Therefore a bottleneck in carbon fixation is the most likely cause of the high-light fitness defect in strains with insertions in Synpcc7942_1057.

1.3.5 An unusual TF Protects DNA from Oxidative Damage?

Synpcc7942_0817 is a ferric uptake regulator (FUR) family transcriptional regulator with homologs across cyanobacteria. Interestingly, the closest one that has been studied (all2473 in Anabeana) has been found [31] to function in the protection of DNA from oxidative damage rather than in transcriptional regulation. Overexpression in *E. coli* increased survival rates when challenged with H_2O_2 or methyl viologen. The authors suggest that high expression leading to nonspecific binding could be physically blocking damage to the DNA.



(a) WT LL **(b)** WT HL

Figure 1.10: Staining for PHA granules with Blue Nile A. Structured illumination microscopy (with Masakazu Iwai) revealed foci similar in appearance to the PHA granules known from other cyanobacteria. Green: Blue Nile A. Red: chlorophyll autoflourescence.

It also happens to be one of the genes I worked on as part of my first high light project on transcription factor knockouts (See Appendix A). Unfortunately I was never able to get the mutant to segregate, and did not pursue it further. The only relevant data I have is that it was slow growing (Figure A.1, second from the left). Based on the fitness phenotype here it should be easier to segregate at low light, or over-expressing it might prove a more fruitful approach.

1.3.6 A new regulator of the HL stress response?

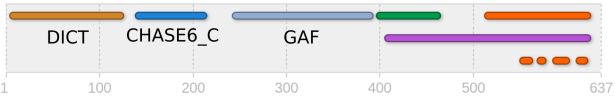


Figure 1.11: Domain predictions for the GAF HK Synpcc7942_2282 (Interpro).

Synpcc7942_2282 is a two-component histidine kinase likely involved in the high light acclimation response. It has several interesting sensor domains: CHASE6_C is the C-terminal part of a two-domain periplasmic sensor found across cyanobacteria; the N-terminal part was reclassified as a separate DICT domain. DICT is found in photosynthetic bacteria and halophilic archaea, leading to the hypothesis that it could be involved in light sensing[8]. GAF domains are often involved in light sensing as well, among other functions. It has been predicted[4] to be part of the indirect nblA regulon, suggesting that it could be related to bleaching (degradation of phycobilisomes) in response to high light and other stresses. Based on these data, Synpcc7942_2282 could be sensing nutrients, light, and/or participating in extracellular communication via an unknown signal. The most promising route to discover more would probably be to profile transcription in knockout and overexpressor lines by RNAseq.

1.3.7 N-II Amino Acid ABC Transporter

The final group of beneficial genes is interesting because they cover an entire operon, Synpcc7942_0246-0249. Interruption of any one of them leads to a similar fitness defect.

Escudero, Mariscal & Flores named the operon N-II by analogy to its homologs in the filamentous cyanobacterium Anabaena (PCC 7120), and showed that it is the major system in *S. elongatus* for import of acidic amino acids, neutral amino acids with polar sidechains, and glycine. Knocking it out reduced glutamate uptake most strongly, to 1-2% of wild type levels [16].

Professor Flores kindly provided me with 3 plasmids used in that paper to make deletions in portions of the operon, which they had also found to behave similarly. I was unable to revive two strains, but grew the third and used it to transform *S. elongatus* , resulting in the Δ0247-8 strain (labeled CSLE15b in the paper). It is missing a region covering most of the two permease proteins, Synpcc7942_-0248. Interrupting them should reliably prevent the ABC transporter from working, but I did not verify that.

I did verify that the transformants lack Synpcc7942_0247, and grew them in plates to confirm the expected high light defect (Figure 1.12). I also discovered that they form an unusual biofilm (Figure 1.13) when left in test tubes without shaking.

This data suggests two distinct hypotheses: first, the fitness defect could be caused by inability to take up glutamate. Second, it could caused by inability to receive an extracellular signal required to inhibit biofilm formation. *S. elongatus* has an unusual "reverse quorum sensing" biofilm logic: it releases small GG-motif-containing peptides into the media to suppress biofilm formation at high culture densities [50,43]. Mutants in export of the signal have been isolated [42], but to my knowledge this is the first indication of a gene required to respond to it.

I favor the first hypothesis though, because it would be consistent with the importance of chlorophyll biosynthesis and D1 repair. It would also immediately explain why the defect is more severe at high light: glutamate is required both as one of the amino acids for D1 synthesis, and as the starting point for making chlorophyll (Figure 1.8).

But why would *S. elongatus* need to take up amino acids? Cyanobacteria, as well as diverse heterotrophic bacteria, have been found to leak them. This may be due to an inherent property of bacterial membranes and/or multiple active and passive transport processes depending on the group in question [36]. Therefore, it is likely that the TnSeq culture media contains a large amount of free amino acids. Taking them up would be an important advantage in competitive growth conditions.

This could be tested by growing wild type and Δ0247-8 strains in media supplemented with glutamate, or by radioactive glutamate uptake assays. The uptake assays could also confirm whether the glutamate is incorporated into D1 and/or chlorophyll at high light, or perhaps used in photorespiration.

The second hypothesis could be supported by starting wild type and Δ0247-8 cultures from a very low OD which would normally trigger "biofilm mode", but in conditioned media from a wild type culture. They would take up the signal and grow planktonically, whereas Δ0247-8 would not.

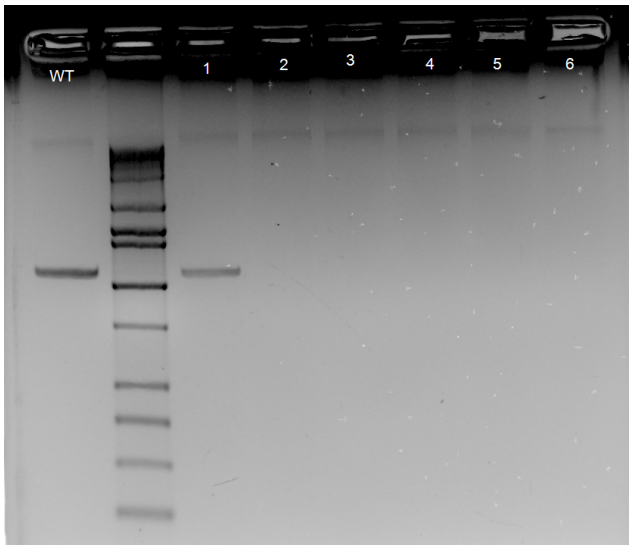
1.3.8 Selection for impaired LPS Export

Despite failing the FEBA quality controls, the first TnSeq run did predict some of the same results as the second. Most obvious was a small set of detrimental genes—that is, interrupting them drastically increased fitness at normal to high light. They are shown in Figure 1.15. Genes in the same pathway appeared again in the second run (Figure 1.7, upper left), although they were less prominent.

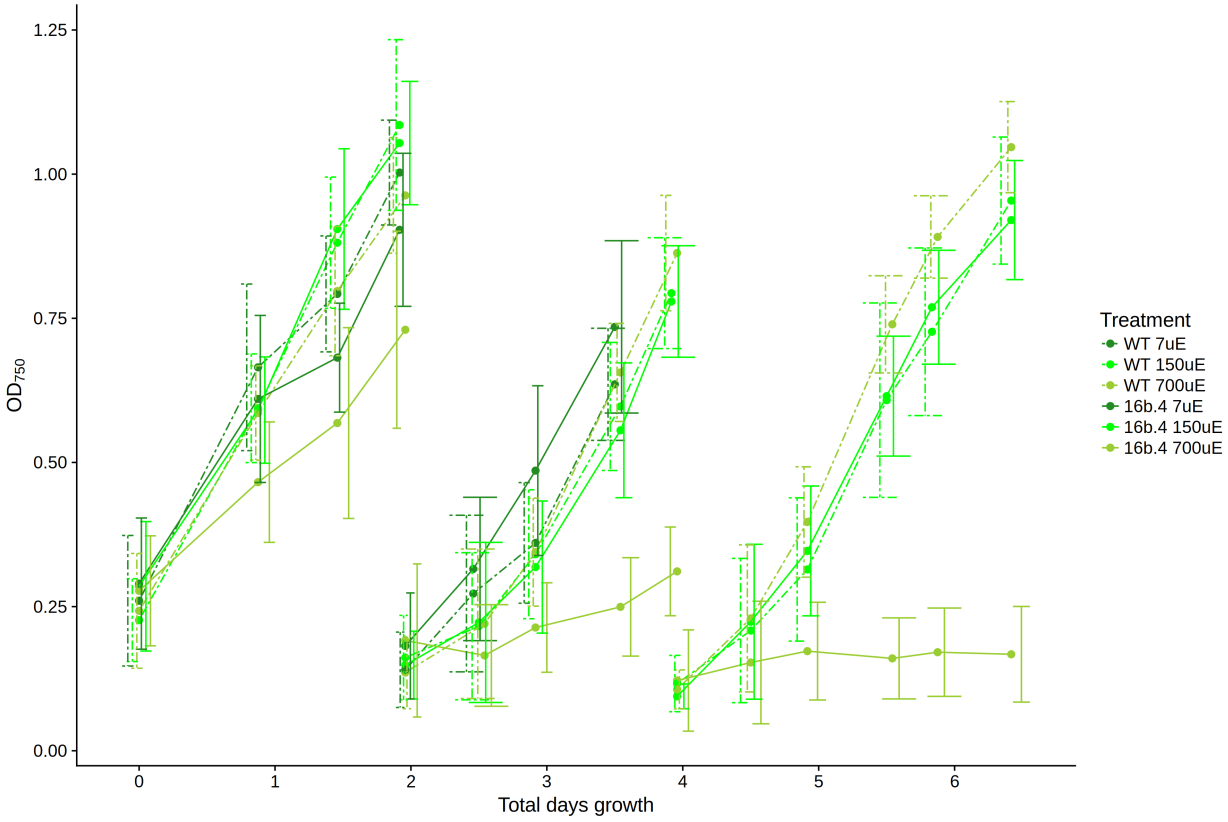
They seem to be related to export and attachment of O-antigen (lipopolysaccharide) to the outer surface of the cell, and were separately discovered in a screen for *S. elongatus* mutants which resist predation by amoebae [54]. One explanation is cryptic contamination of the TnSeq culture with grazing eukaryotes—especially in light of similar

issues discussed in Section 2.3.12. However, none of the TnSeq cultures grown in flasks collapsed in a similar manner to the PBRs, even if left growing for months after the experiment (data not shown).

An alternate hypothesis is that the "rough" O-antigen mutants, as they are called due to their appearance under the microscope, are also better suited to growth in flasks at medium to high light intensity. Simkovsky *et al.* did find that they have a high light growth advantage, and speculated that knocking out the pathway might be useful for biofuels since it leads to a grazer-resistant strain that also grows well at high light and auto-flocculates for easy collection. That would be a disadvantage in many natural systems. Cyanobacterial exopolysaccharides are used partly for bouyancy, which is important for staying close to the surface. Perhaps when grown with constant shaking they are a waste of materials. These same genes also appear in the fluctuating light data in Chapter 2, and their dominance seems to increase the longer the library is grown.



(a) Segregation of Δ0247-8 mutants



(b) Δ0247-8 growth curves

Figure 1.12: Confirmation of HL growth defect in Δ0247-8strain. **a.** 5 out of transformants lack the wild-type Synpcc7942_0247 gene. **b.** Wild type *S. elongatus* and one of the segregated Δ0247-8 strains were each grown in triplicate in 24-well plates at low (7 µmol photons m^{-2} sec^{-1}), medium (150 µmol photons m^{-2} sec^{-1}), or high (700 µmol photons m^{-2} sec^{-1}) light. Similar to the TnSeq outgrowths, cultures were diluted every 2-3 days. Dashed lines indicate wild type and solid lines Δ0247-8. Colors indicate light level.

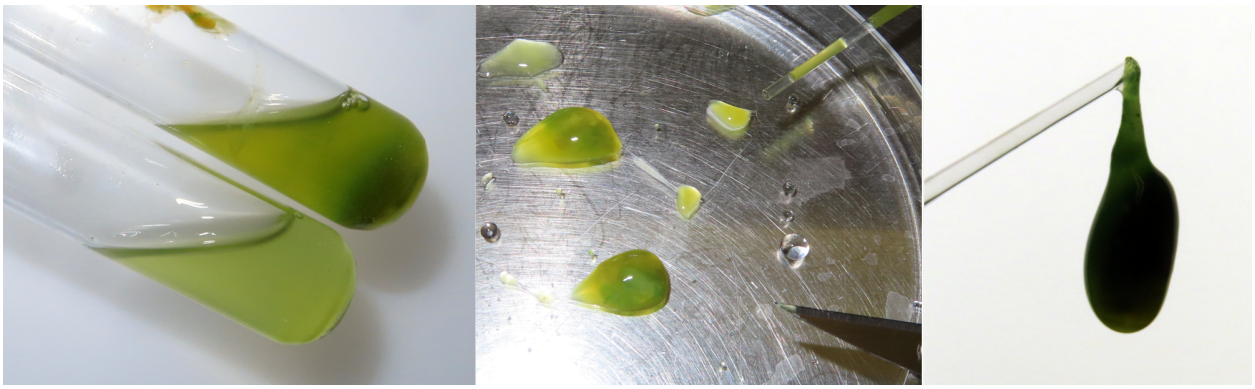


Figure 1.13: ABC transporter KO Δ0247-8 exhibits "gooey" biofilm. **Left:** Test tubes of Δ0247-8 (top) and kanamycin resistance control strain (bottom) were left out on the benchtop without shaking for several weeks. **Middle, Right.** The KO developed a strongly self-adhesive, gelatinous biofilm.

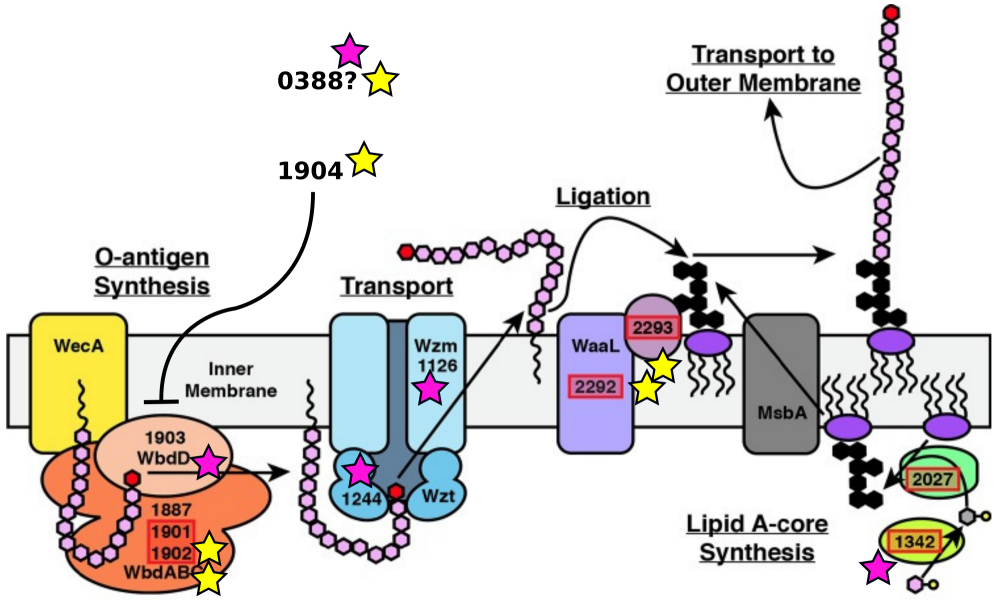


Figure 1.14: O-antigen export pathway hits. Modified from Simkovsky *et al.*[55]. Numbers are Synpcc7942 locus IDs. Purple stars indicate genes identified in the first TnSeq run; yellow stars indicate genes identified in the second run. Synpcc7942_1904 was proposed to regulate activity of Synpcc7942_1903. Synpcc7942_0388 had high differential fitness in both runs, and was also identified by Simkovsky *et al.* Its function remains unknown.

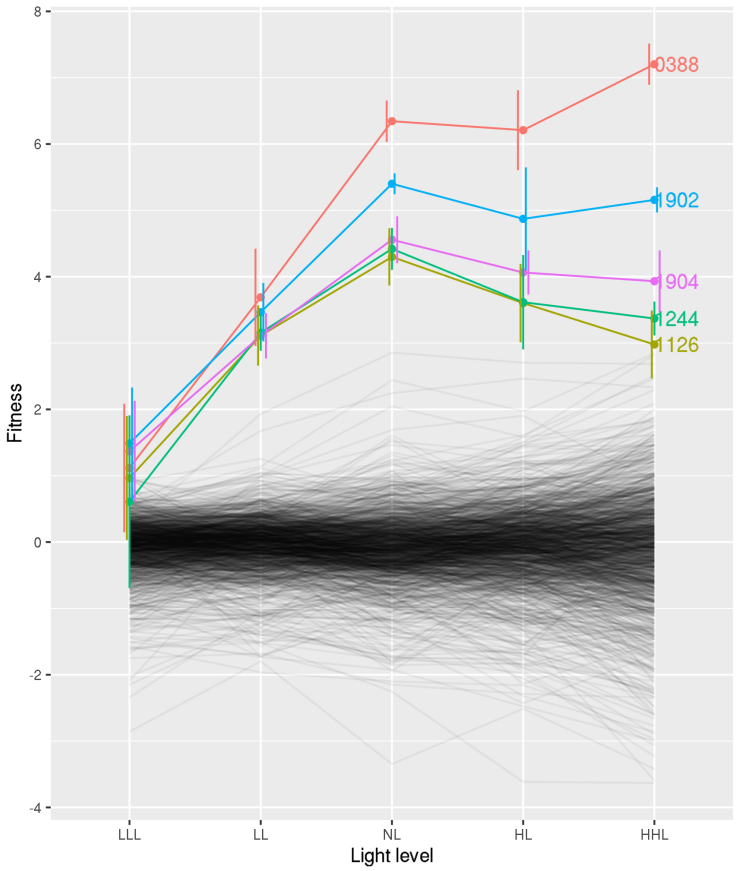


Figure 1.15: Selection for impaired O-antigen at high (and normal) light. A small set of genes related to export of O-antigen stand out from the rest of the first TnSeq run. Mean plus and minus standard deviation of 3 flasks at each light level shown.

1.4 Future Directions

Although my custom metadata format was useful for this study, it should be replaced (or at least reconciled) with the standard one so the data can be added to the Fitness Browser (`fit.genomics.lbl.gov`), which compares co-fitness across genes, organisms, and stress conditions for thousands of BarSeq experiments. The *S. elongatus* library has lagged behind heterotrophic species in terms of data sharing, despite being the most-requested library from the original paper (Adam Deutschbauer, personal communication), and I would like to see the website updated to work for the kinds of comparisons relevant to photosynthetic studies.

More importantly, some of the genes discovered here should investigated further. TnSeq libraries do not allow easily isolating individual mutants from the population, but they do indicate which conditions to segregate the transformants under, making separate knockout and complementation strains relatively easy to generate. All the genes described here should segregate under low light and have phenotypes at high light. I have the knockout constructs for a few test genes designed and partially constructed already, as well as in-progress script that generates all the relevant primer sequences (Appendix A).

1.5 Conclusion

Based on the top hits in this experiment, recycling of chlorophyll from damaged D1 via the phytylase/dephytylase cycle may be the most crucial HL-specific adaptation. This makes sense, because a defect in the pathway would either cause a buildup of highly reactive intermediates, or force cells to shut down photosynthesis entirely to avoid that buildup. I also found several interesting, unknown genes which will require more thorough characterization.

Most of the proteins needed for PSII assembly in plants are deeply conserved and have cyanobacterial homologs[44], so this TnSeq library could be a viable route to more identifying previously unknown components—as long as they are not absolutely required under standard conditions.

Chapter 2

RB-TnSeq to identify genes required under fluctuating light

2.1 Summary

I was also interested in photosynthetic regulation under dynamic conditions, so I also grew the TnSeq library in multiple fluctuating light conditions and compared the genes required for growth under each. With high vs low light there was only one clear comparison to make, but here many different comparisons are possible. So in addition to looking at which genes are beneficial or detrimental in which fluctuating light regimes, I also globally clustered the data to search for patterns. Many genes and interesting patterns have begun to emerge, but will require further analysis. Unexpectedly, the flavodiiron proteins flv1 and 3 were not among the strongest hits, though they were mildly beneficial. Many other genes that appear relevant for growth under fluctuating light were identified, including alternative electron sinks and an inhibitor of cell division with a role in the stringent response.

2.2 Methods

Most of the basic methods remain the same as in the constant light experiments in Chapter 1; only the changes are described here.

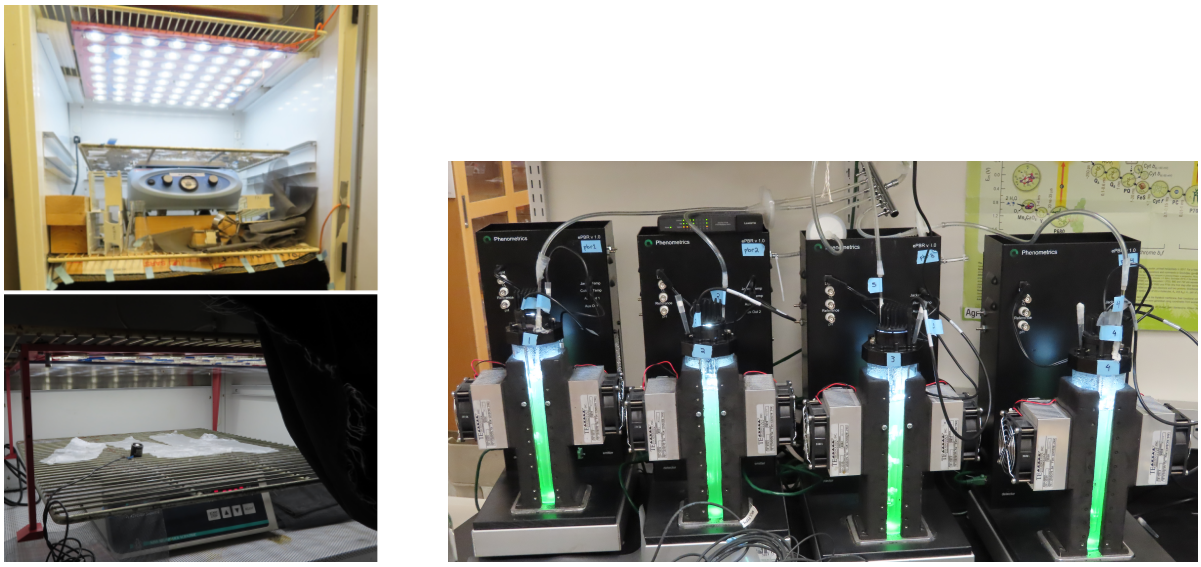
2.2.1 Experimental design

6 patterns of fluctuating light (Figure 2.2) and two types of growth vessels (Figure 2.18) were investigated. One baseline pattern (FL2) was chosen to match conditions previously shown [2] to inhibit growth of Δ*flv1* and Δ*flv3* strains, and 4 others were chosen to facilitate comparisons:

- FL1, 2, and 3 cover a range of time scales: seconds, minutes, hours
- FL2 and 4 compare high/low with low/dark (0 μmol photons m^{-2} sec^{-1}) fluctuations
- FL2 and 5 have the same period, intensities, overall photon flux, and timescale but FL5 changes intensity gradually rather than suddenly

The final pattern was designed to mimic the light regime found in Phenometrics photobioreactors (ePBR v1.1) based on a simple empirical model explained next in Section 2.2.2. It combines characteristics of several others: rapid fluctuations similar to FL1, low intensity periods similar to FL2, and a mix of rapid and gradual shifts between light levels.

To investigate possible differences in the regulatory mechanisms triggered by complete darkness as opposed to very low light, the lower growth chamber was covered with a blackout curtain (Figure 2.18a) and PAR was confirmed as 0 on the light meter. FL6 was done in the same chamber, but without complete darkness. The LED arrays have a minimum level of about 12 µmol photons m^{-2} sec^{-1}, so a dim supplemental fluorescent light (~0.5-0.7 µmol photons m^{-2} sec^{-1}) was added to approximate the levels in dense PBR culture. The other treatments were done in the photobioreactors themselves.



(a) Growth chambers **(b)** Photobioreactors

Figure 2.1: Growth setup for the TnSeq FL experiments. **a.** Two treatments at a time were exposed to FL, isolated from each other by a cardboard light barrier (middle), and an additional blackout curtain for low light treatments (bottom half). **b.** The PBR FL treatments were grown in 4 photobioreactors (Phenometrics ePBR v1.1), bubbled with air from an aquarium pump.

2.2.2 Model for fluctuating light in photobioreactors

Based on measured intensity of 512 µmol photons m^{-2} sec^{-1} at the top and 0.33 µmol photons m^{-2} sec^{-1} at the bottom (See Figure 2.3), PAR in the 20cm-tall dense *S. elongatus* culture was estimated to be:

$$I = I_s * 0.6926^d$$

Where I is the intensity at a given depth in µmol photons m^{-2} sec^{-1} , I_s is intensity at the surface, and d is the depth in cm. As shown in Figure 2.3, the resulting curve was combined with video tracking of a small marker object of roughly neutral buoyancy (piece of sponge) to approximate the light levels experienced by a cell over 10 minutes. It is only a rough estimate due to several factors: changes in density during the outgrowths, differences in motion of a sponge vs a bacterial cell, etc. However, the overall pattern that a given cell experiences mostly

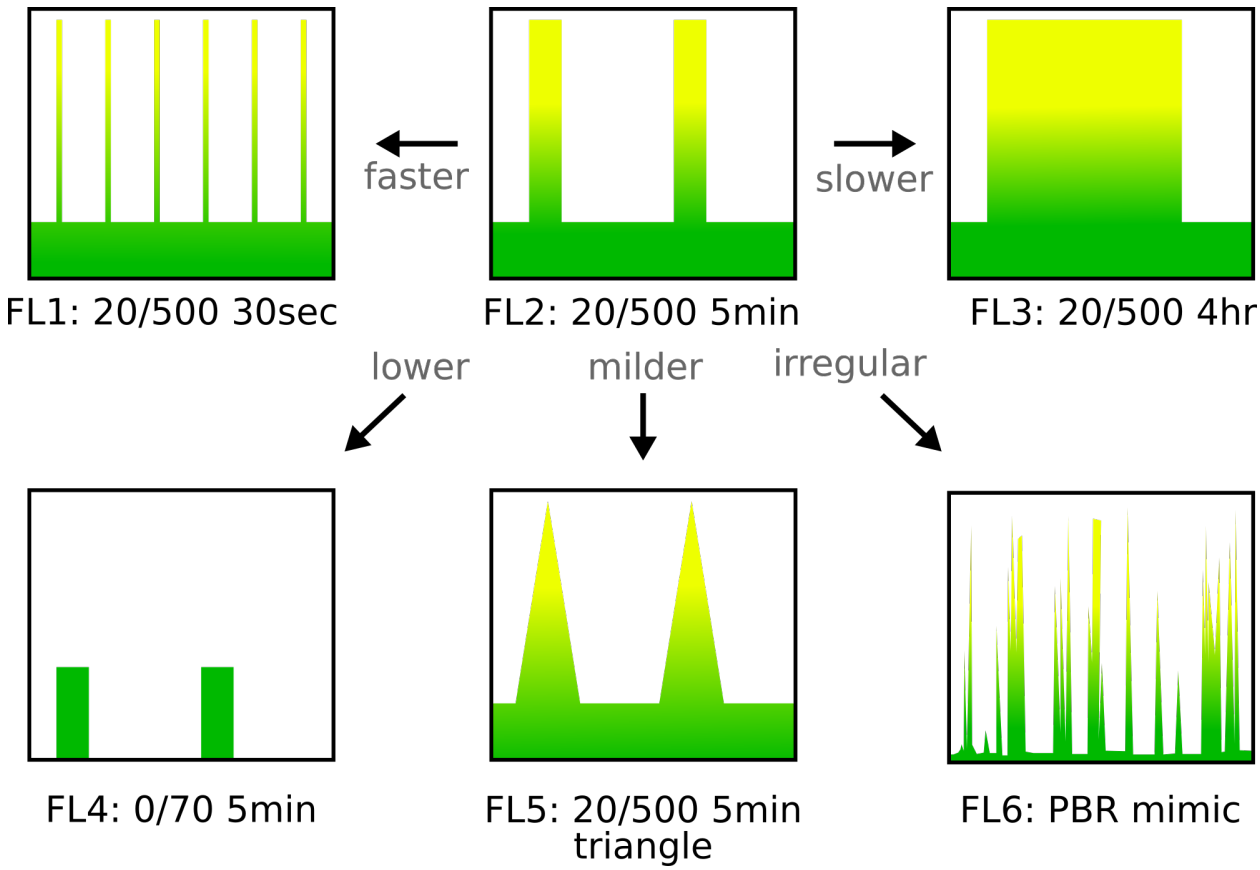


Figure 2.2: Fluctuating light LED patterns. **Patterns not to scale.** Custom LED panels (JBeamBio) with cool white LEDs (BXRA-56C1100-B-00, Farnell) were programmed with fluctuating light patterns. FL1-3 consist of 90% low light (20 µmol photons m^{-2} sec^{-1}) interrupted by high light (500 µmol photons m^{-2} sec^{-1}) at regular intervals on increasing timescales. FL2 is similar to the condition shown[2] to inhibit growth of Δ*flv1* and Δ*flv3* strains. FL4-6 change other parameters: FL4 alternates dark with normal light (70 µmol photons m^{-2} sec^{-1}), FL5 makes shifts between low and high light more gradual, and FL6 is an irregular pattern designed to mimic the light regime found in the PBRs (See Figure 2.3 for details).

low light punctuated by short random spikes up to about 500 µmol photons m^{-2} sec^{-1} should hold, and is in agreeement with a forthcoming study[6] that constructs and validates a fluid dynamic model of the light regime in these same photobioreactors.

An LED array (Figure 2.18, bottom) was programmed to follow the same pattern, with the limitations that the LEDs could only change intensity once per second, and could not go below about 12 µmol photons m^{-2} sec^{-1}. Since complete darkness might have different physiological effects from low light, a dim fluorescent bulb was also used to supplement with about 0.5-0.75 µmol photons m^{-2} sec^{-1}.

2.2.3 Outgrowths and sampling

PBR and LED outgrowths were done in parallel starting from the same Time 0 sample as the first LED outgrowths (FL1 and 2). Only one PBR treatment was planned. However, due to contamination by grazers (Section 2.3.12), they had to be repeated twice more. Only two replicates started from the FL3+4 Time 0 sample were

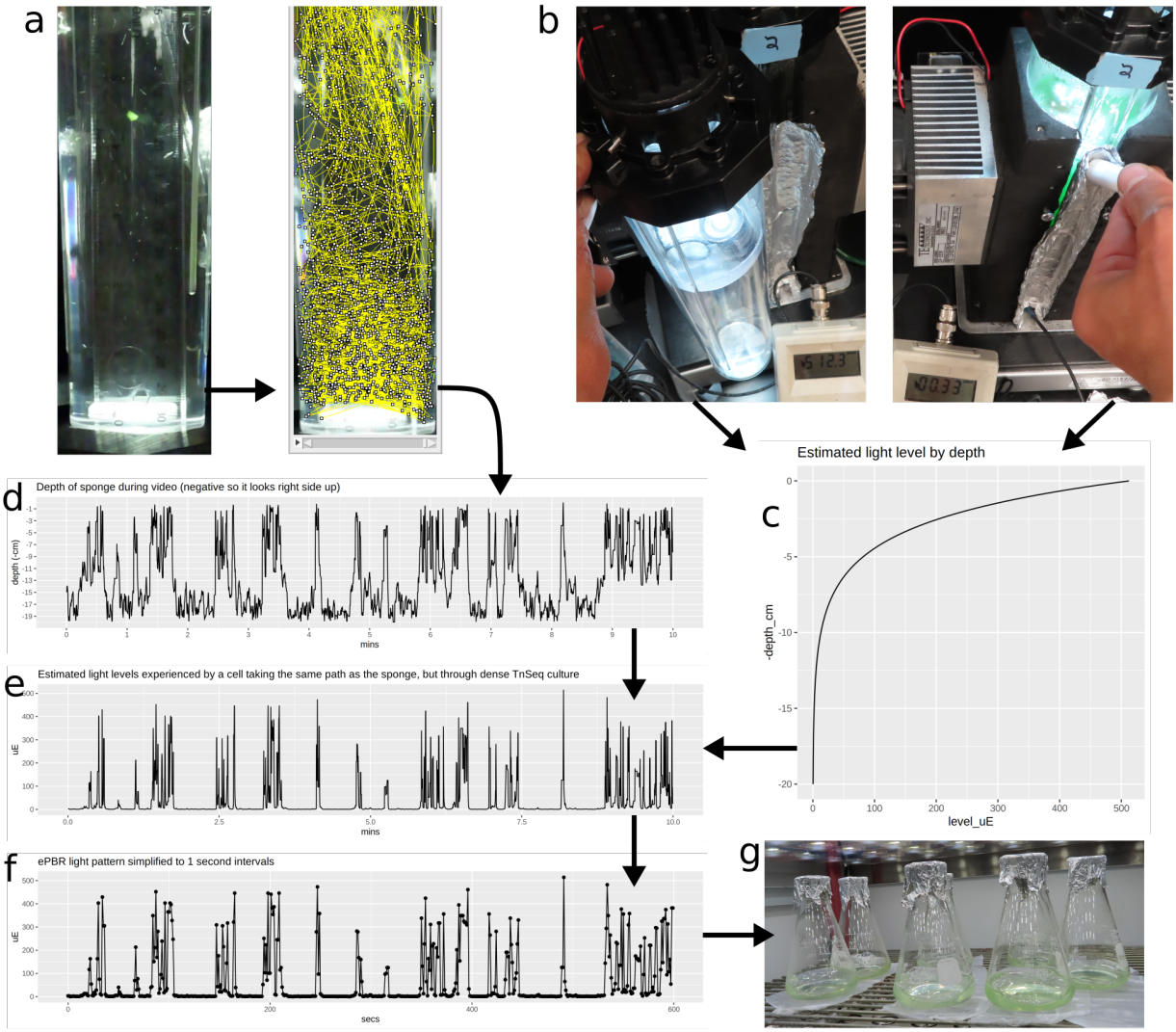


Figure 2.3: Mimicking the PBR light regime in flasks. **a.** Particle tracking: the motion of a small piece of sponge (yellow, top center) in the PBR culture vessel was recorded for a period of 10 minutes, and particle tracking was performed in Fiji. **b.** PAR measurement: the intensity of photosynthetically active radiation (PAR) reaching the culture was estimated by measuring at the same height in an empty vessel (left), and the intensity passing through the entire culture was estimated from beneath the vessel (right). (The light meter is inserted at the bottom; the white device at the top is a magnet being used to hold the stir bar out of the light path.) **c.** Intensity was estimated as a function of depth in the culture (see text for details). **d.** Plot of depth of the sponge over time as determined by particle tracking. **e.** Depth data and the extinction curve were used to estimate the light intensity experienced by a hypothetical cell taking the same path through dense *S. elongatus* culture. **f.** The light intensity plot was simplified by taking the maximum intensity in each 1-second interval. **g.** An LED array was programmed to match the simplified pattern in flasks.

sequenced. Before discovering that grazers were the culprit, a final PBR-only outgrowth was also done on the theory that the collapse could be density-dependent feature. It was growth an extra 5 days in Time 0 conditions and

started at a high density, and is included with as PBR treatment F (for "final").

The extra outgrowths also presented the opportunity to investigate long-term growth and grazer resistance. Two contaminated and one uncontaminated PBR were left running past the planned end of their outgrowths. The contaminated cultures were sampled after turning yellow (Figure 2.17), then back to green 4-7 days later. The long-term uncontaminated culture was sampled twice over the next several weeks, and an earlier sample was added to make a 4-part time series spanning about a month. To avoid cross-contamination with grazers, no OD_{750} measurements were taken during these outgrowths. See Figure 2.5 for an overview of all the TnSeq outgrowths and sampling points, including the constant light experiments from Chapter 1.

2.2.4 Library preparation and sequencing

The optimized DNA extraction and PCR protocol from the first set of experiments resulted in variable quality during the FL library preps, even after experimenting with various PCR conditions. Some samples were high quality (one solid band ~180bp), while others had extra bands and/or smearing. This may be due to differences in the number and composition of cells that could be sampled from different treatments. Final PCR conditions were chosen to prioritize evenness between samples over clean bands (Figure 2.6).

2.2.5 Updated analysis

This set of experiments uses an extended version of the previous pipeline (Section 1.2.5). It can be downloaded and run the same way by changing "hl" to "fl":

```
git clone https://bitbucket.com/jefdaj/tnseq7942-fl.git
cd tnseq7942-fl
nix-build --verbose -j$(nproc) --option build-use-chroot false --keep-going
```

Since it also includes the previous results, this version will be the one maintained and referenced in the paper.

Clustering counts and fitness ratios

Barcode counts aggregated at the gene level were clustered using two separate software packages. DESeq2 results were normalized with a regularized log transform as recommended in the documentation, then scaled by quantiles for visualization.

The Clust Python package[1] was used to automatically normalize and cluster gene-level barcode counts (not fitness values) across all 17 treatments and 68 samples sequenced, including the Time 0 conditions. 8 robust clusters emerged (Figure 2.8). They are designated C0-7 and contain between 12 and 121 genes each (380 total).

FEBA also has built-in cofitness analysis, which was used to search for genes correlating with the core photosynthetic machinery (Figure 2.8).

For the principal component analysis, a distance matrix was constructed using the number of genes with fitness effect > 1 in both conditions, and distances were clustered with k-means (k=9).

2.2.6 Confirmation of grazer-dependent biofilms

Collapsed PBR culture was spun down at 4000rcf and washed with fresh BG11 media. *S. elongatus* and biofilm mutants (isolated from the previous outgrowth) were challenged with either supernatant, which would be enriched in the toxin, or resuspended pellet, which would be enriched in large cells and their cysts. Cultures were imaged by Differential interference contrast (DIC) microscopy the next day.

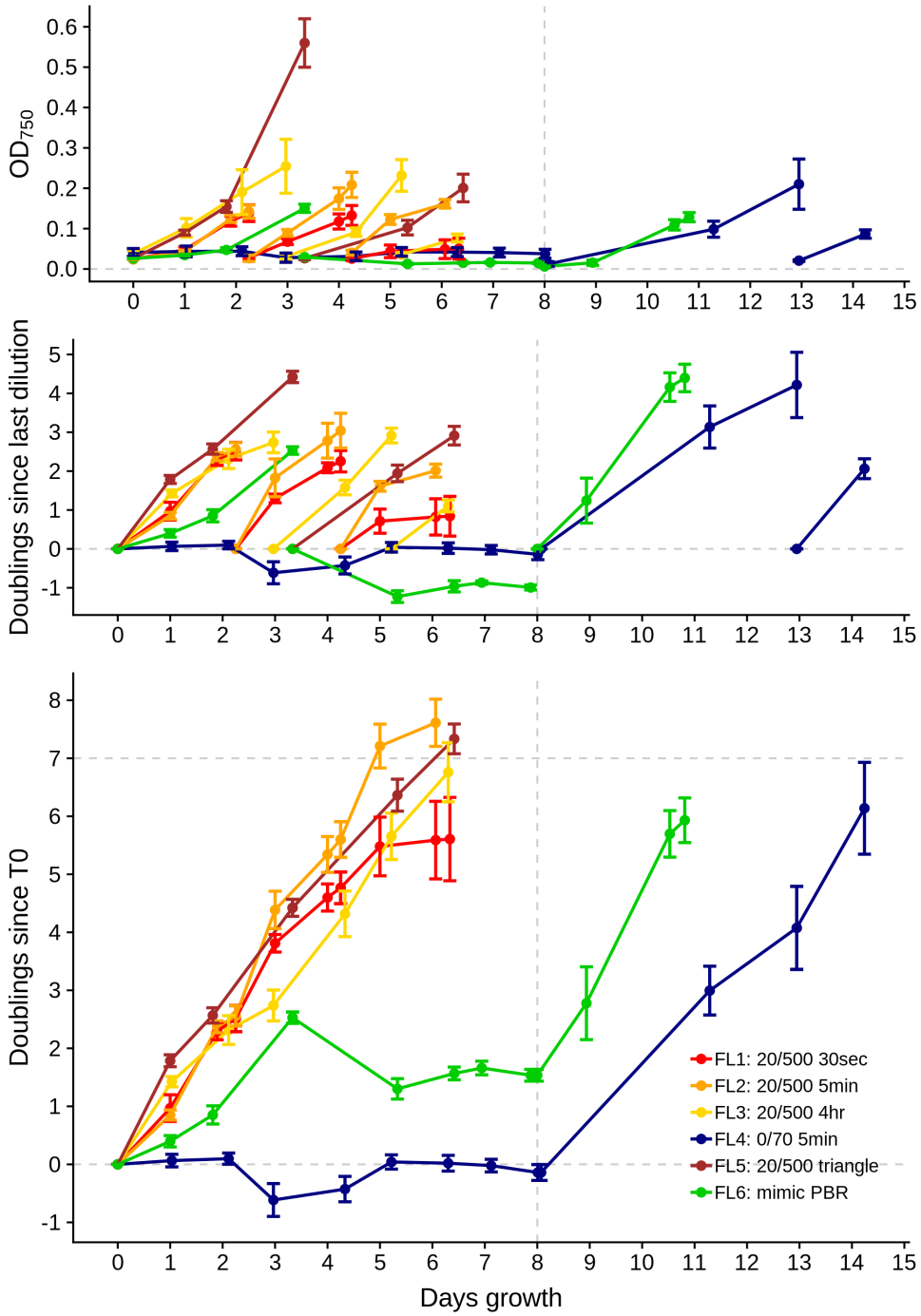


Figure 2.4: TnSeq fluctuating light growth curves. OD_{750} of 1ml samples of TnSeq cultures measure in 24-well plates on the Tecan Infinite M1000 plate reader. Error bars represent mean +/- standard deviation of 6 replicate flasks each. Raw OD_{750} (top) was used to calculate doublings since the last dilution and sampling timepoint (middle), and total doublings since the Time 0 sampling (bottom). Top dashed line marks the target of 7 generations total growth. Vertical dashed line marks the end of experimental outgrowths, when FL4 and FL6 were transferred to recovery conditions (70 µmol photons m^{-2} sec^{-1}).

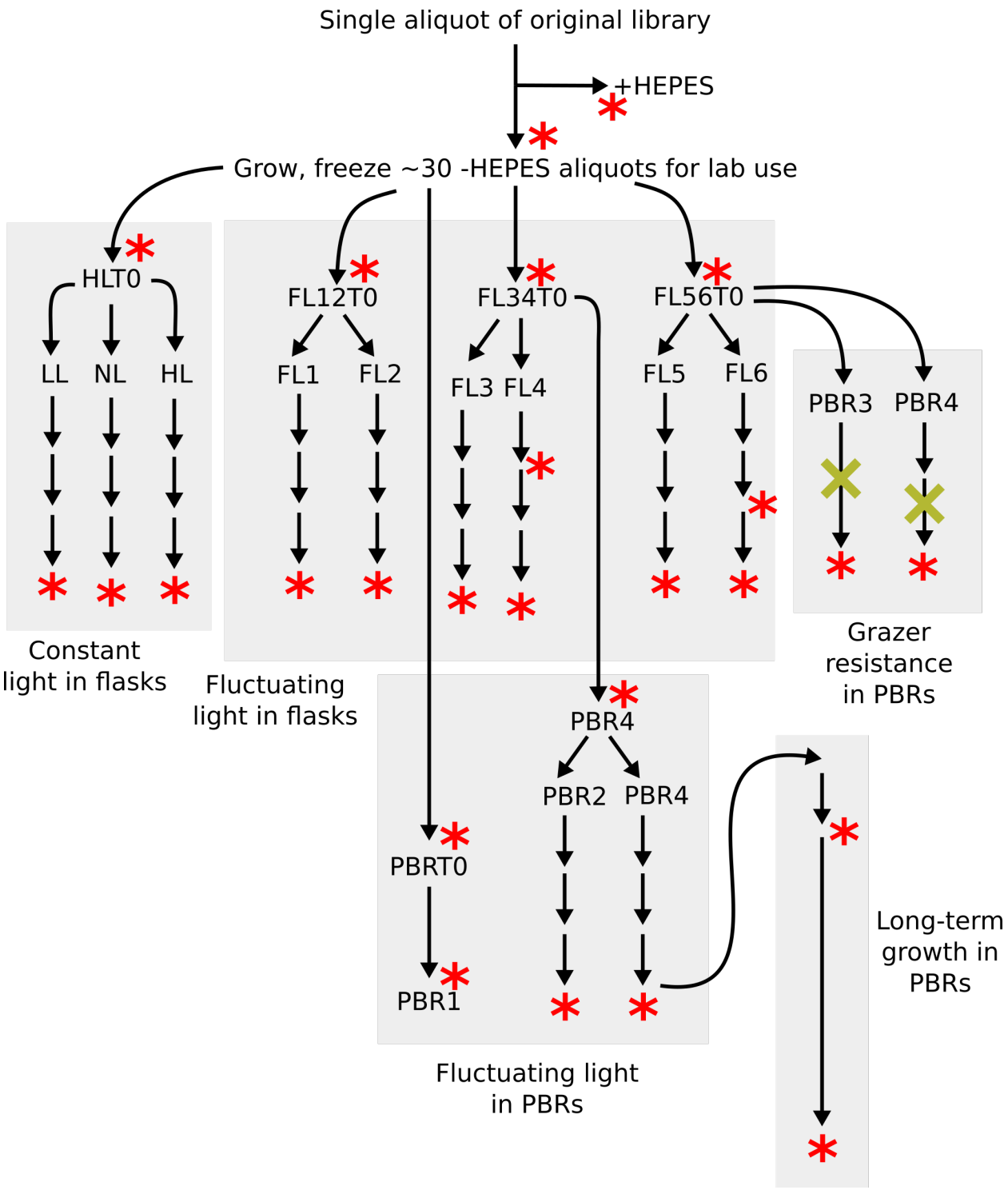


Figure 2.5: Overview of all TnSeq outgrowths (high light and fluctuating light), starting with the aliquot received as a gift from the Susan Golden lab. Constant light samples are the ones from the first set of experiments (Chapter 1), also shown in Figure 1.1. Arrows represent one outgrowth, followed by dilution and freezing samples for later DNA extraction. Red asterisks represent samples that were sequenced. Gold crosses represent cultures that collapsed, then regrew several days to a week later. Not shown: more PBR outgrowths that also collapsed and regrew, but were not sequenced.

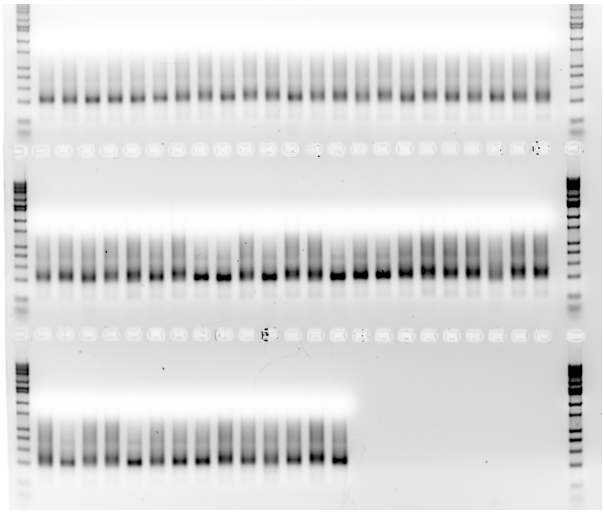


Figure 2.6: Unavoidable smears during the PCR library prep for fluctuating light samples.

2.3 Results

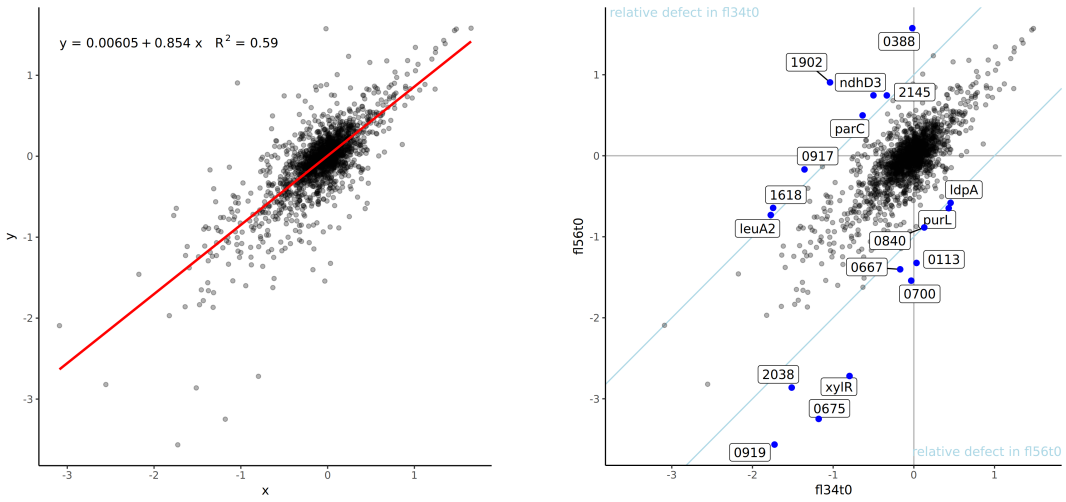
2.3.1 Quality control and global patterns

Barcode counts from the FL experiments were noisier than those from the HL experiment in Chapter 1, but the larger number of samples also presents the opportunity to compare and filter them more stringently. Pairwise comparisons of Time 0 samples with a lax fitness score cutoff of +/- 1 (Figure 2.7) led to 50 false positive hits. They were removed from most fitness comparisons, and are represented by red dots in pairwise comparison plots. A more stringent cutoff was used to define beneficial genes in the FL treatments:

$$abs(mean(fitness)) - abs(sd(fitness))) >= 2$$

There are many possible ways to group the FL treatments, so clustering was applied to determine which variables correlate most strongly with fitness. Gene-level counts were clustered separately using DESeq2[5] and Clust[1], two packages designed for differential expression data but which can take any count data as input. TnSeq barcode counts are expected to have different statistical properties from transcripts—for example their up- and down-"regulation" are limited by cell growth and death rates—so only the most conservative clusters identified by both packages, using different normalization and clustering algorithms, are presented here.

As shown in the DESeq2 heatmap (Figure 2.9), no single variable is sufficient to explain the clustering of gene-level counts between samples. PBR samples cluster together, as do each of the constant-light treatments, but the FL treatments largely overlap. The post-recovery FL4 and FL6 samples mostly cluster together. FL2 and FL5 are very similar, as expected. The grazed PBR treatment was removed from both analyses because it contains outliers in a majority of genes, which dominated differences between the other samples.



(a) Correlation of gene fitness in FL34 and FL56 Time 0 samples

(b) Spurious hits in comparison of FL34 and FL56 Time 0 samples

Figure 2.7: Significant variation between Time 0 outgrowths. Low correlation between Time 0 samples and the resulting false positives when using a lax fitness cutoff of 1 for pairwise comparisons.

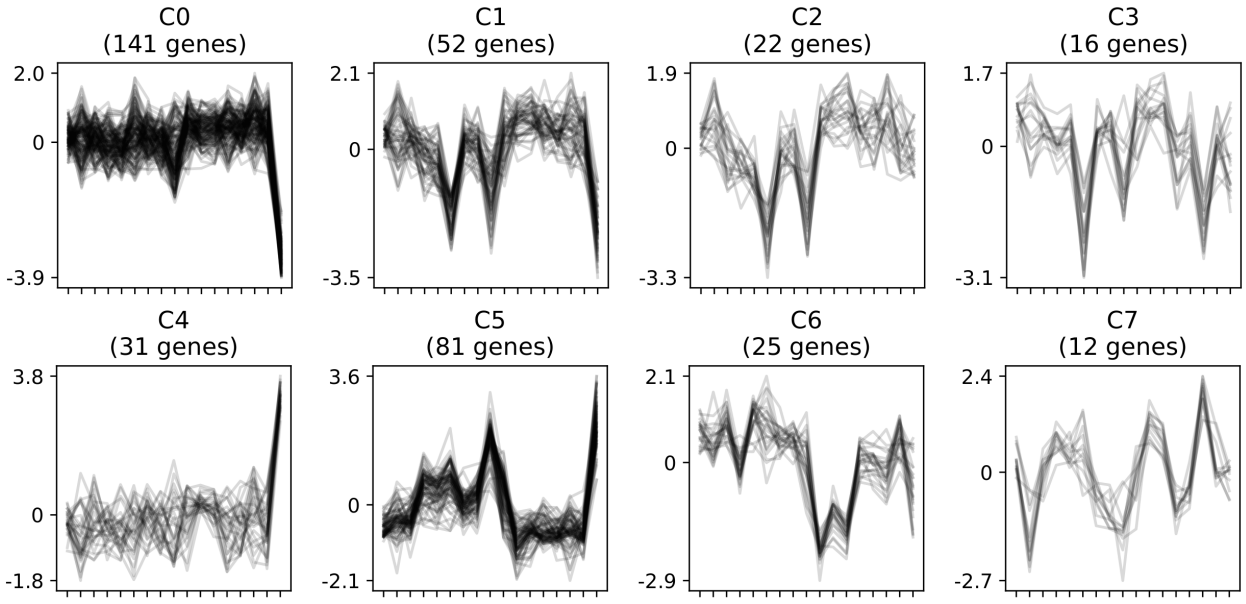


Figure 2.8: Genes clustering by barcode counts across all treatments and samples according to Clust. Automatic normalization and clustering was performed using Clust[1] with the the default "tightness" parameter '-t 1'. Other '-t' settings produced nearly identical clusters C3, C5, and C6 (data not shown). Axes are arbitrary.

Treatments were also clustered using principal component analysis of the number of FEBA hits they have in common (Figure 2.10), to identify patterns that remain after raw counts are normalized and compared. A permissive cutoff of +/- 1 was used because some treatments have no stronger hits. Time 0 and constant light samples each cluster together as expected. The PBRf Time 0 treatment diverged from the others, which is also expected since it was grown longer. The main PBR treatment clusters with FL5 and the pre-recovery FL6 sample as well as the first long-term PBR timepoint derived from it, while FL1,2,3, and 4 (pre-recovery) all cluster together closely. This is interesting because they represent mostly the "PBR-like" and "un-PBR-like" FL treatments respectively. The post-recovery FL4 and FL6, as well as the longer-grown PBR treatments, are all different from each other. Along with the larger overall differences between post-recovery gene counts in Figure 2.9, this supports the idea that FL4 and FL6 cultures underwent roughly equivalent selection to the other FL samples during the first stage of their outgrowths (Figure 2.4), despite little to no increase in OD_{750}.

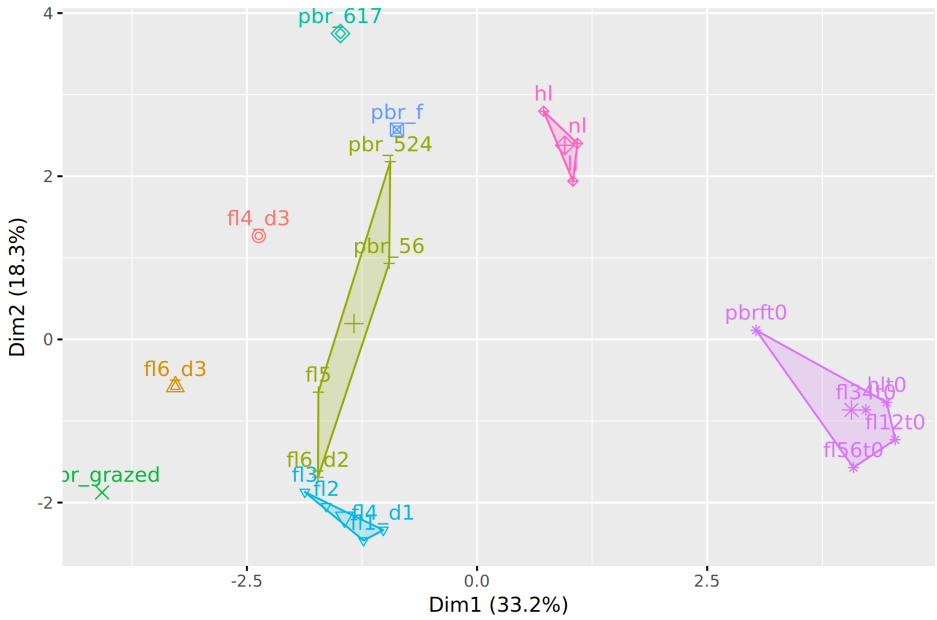


Figure 2.10: Principal component analysis of shared hits. Distances were computed from the number of genes with fitness effect > 1 shared by each pair of treatments.

2.3.2 Co-fitness clusters

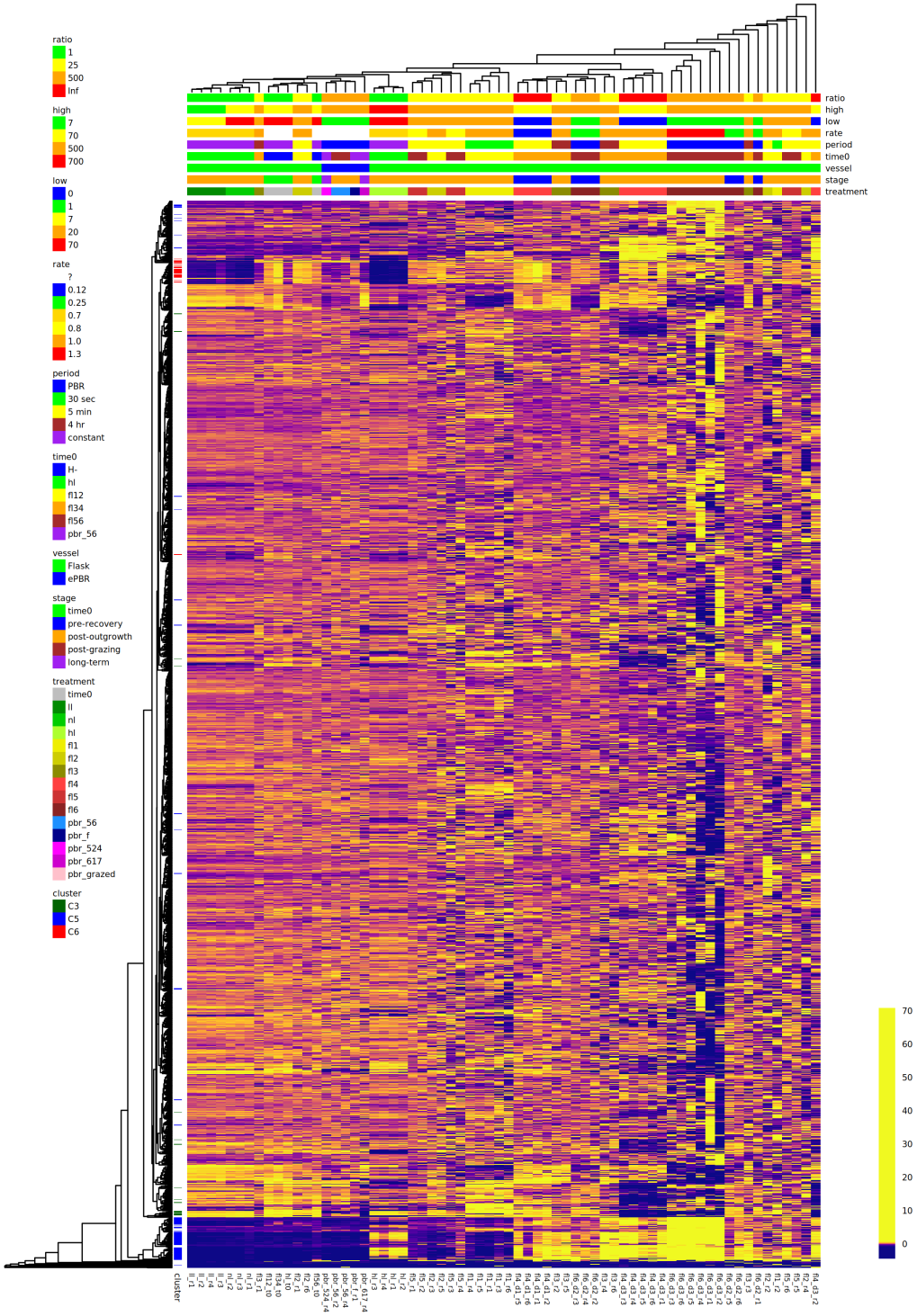
Three high-confidence gene clusters were identified by both the DESeq2 and Clust algorithms 2.11. Cluster C3 (Table 2.3), contains photosynthetic genes, including Synpcc7942_1043 and Synpcc7942_2355, the phytyl-phosphate kinase and putative phasin identified as beneficial under HL in Chapter 1. Phytyl-phosphate kinase was also beneficial in the FL4, FL6, PBR, and grazed PBR treatments of the current experiment; Synpcc7942_2355 was beneficial in FL4 only. Other genes in the cluster include ndhD1 (Synpcc7942_1976), GGAT (Synpcc7942_2160) and ycf52 (Synpcc7942_0773), discussed in Section 2.3.5 below, and the unstudied Synpcc7942_0006-0008 operon. The genes in this cluster are all beneficial, mostly in the long-grown treatments (FL4 and FL6 post-recovery, PBR grazed) and HL.

Treatment	Beneficial genes	Deleterious genes
hl	20	1
fl4_d3	16	5
pbr_grazed	12	32
fl6_d3	7	8
pbr_56	6	0
ll	3	0
nl	3	0
fl5	1	0
fl1	0	2
fl4_d1	0	1

Table 2.1: Numbers of beneficial and detrimental genes discovered (at a cutoff of 2) per treatment.

Figure 2.9 ***(following page)*****:** Genes clustering by barcode counts across all treatments and samples according to DESeq2. Rows (genes) and columns (samples sequenced) are clustered by patterns in their normalized rlog-transformed gene-level barcode counts. Column labels encode sample metadata and row labels gene sets. Sample metadata: Time 0, which Time 0 sample the outgrowth started from; vessel, whether the outgrowth was done in a flask under an LED array or in a photobioreactor; growth stage, when the sample was taken; growth rate, bulk growth rate from OD_{750} (Figure 1.2), where white is unknown; low light, the lowest light intensity used; high light, the highest light intensity used; ratio, ratio of highest/lowest intensity; fluctuation period, period of one high/low cycle, treatment, groups in which all factors match except perhaps the growth stage sampled. Gene sets: Clust, genes clustering in the Clust analysis (Figure 2.8); Removed, genes removed from fitness plots because their fitness varied by at least 1 between Time 0 samples; PS Term, select photosynthesis-related gene ontology annotations.

Figure 2.9: (continued)



Cluster C5 is much larger and contains some of the LPS export pathway genes identified earlier (Section 1.3.8): Synpcc7942_0388, 1126, 1244, 1902, 1904, 2292, and 2293. They are deleterious, mostly in the post-recovery FL4 and FL6 and PBR grazed treatments. Synpcc7942_1902 is also beneficial in the LL and main PBR treatments. *S. elongatus* uses at least one pathway homologous to the Wzm/Wzt system in *E. coli* to synthesize exopolysaccharides[28], and it is likely that many of these genes participate in that process. It is still unclear whether they contribute to grazer resistance directly or simply to faster growth in the absence of competition during the recovery from stress conditions.

Cluster C6 is interesting because it contains the only gene (Synpcc7942_0109) beneficial only under the constant light conditions, discussed in Section 2.3.10. It also contains over half the genes found on the larger of two *S. elongatus* plasmids, pANL.

FEBA also includes its own co-fitness analysis, based on fitness values rather than raw counts. The strongest co-fitness values across all genes were observed between PsbW and the genes in Table 2.8. Several were identified in Section 1.3.2 as specific to chlorophyll biosynthesis and PSII repair: Synpcc7942_0508, Synpcc7942_2282, Synpcc7942_0817, Synpcc7942_0247, Synpcc7942_2355. Most are new though, and correlation with that pathway suggests they are also involved in maintaining photosystem stability under multiple FL conditions.

2.3.3 Genes with fitness effects in multiple conditions

Several LPS export pathway components (Section 1.3.8) have large fitness effects across many of the fluctuating light treatments as well, but some were removed from the analysis because they also varied between the Time 0 samples. They appear to be important contributors to fitness under any condition involving selection for rapid growth, or perhaps rapid protein synthesis, and to have some effect under most conditions.

Synpcc7942_2160, annotated as alanine-glyoxylate aminotransferase with a possible role in photorespiration, has fitness defects of -2 or more in 9 treatments with no obvious pattern uniting them. The strongest defects were in the FL4 and FL6 recovery outgrowths, FL5, and HL. An alanine-glyoxylate aminotransferase homolog in *Arabidopsis* was found[30] to be a photorespiratory Glutamate-glyoxylate aminotransferase (GGAT or GGT).

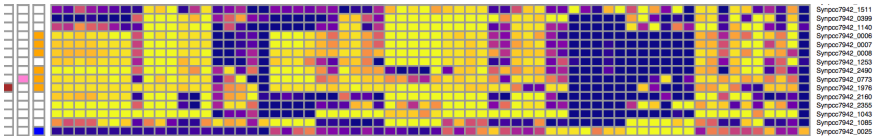
Synpcc7942_1043, the phytyl kinase identified in Section 1.3.2, was also in beneficial (defect > 2) in the FL4 and FL6 recovery outgrowths and in each PBR treatment.

2.3.4 Photosynthetic genes without differential fitness

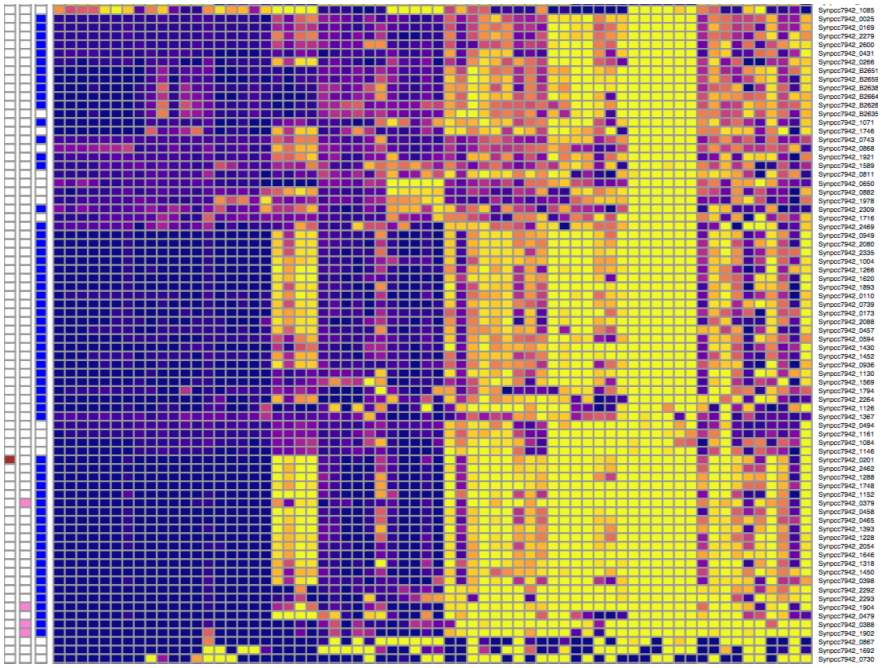
As noted in the Rubin *et al.* paper[48], FEBA is unable to uniquely map barcodes to the PSII reaction center due to paralogs with very similar sequences: psbA (Synpcc7942_0424, 0893, 1389, and 0655), and psbD (Synpcc7942_0655 and 1637). They also classified much of the rest of the core machinery as essential under standard conditions: cytochrome b559, the internal PSII antenna proteins, core subunits of the cytochrome b6f complex and PSI, and 2 of 3 PSI subunits responsible for docking with ferredoxin.

Neither of the two flavodiiron proteins present in *S. elongatus* , flv1 and 3 (Synpcc7942_1810 and 1809 respectively) had large fitness effects under any of the FL conditions studied, except for flv3 in the PBR grazed treatment. They did both have effects > 1 in all the PBR conditions, and flv3 also in FL4 post-recovery. Previous work[2] showed dramatic phenotypes for knockouts of either gene in conditions similar to the FL2 treatment. It is possible, though unlikely, that their phenotypes depend on growth as a monoculture rather than in a population with other genotypes, or on minor differences in the growth conditions. More likely, the noise in this dataset simply prevents resolving them from the background without also picking many false positives.

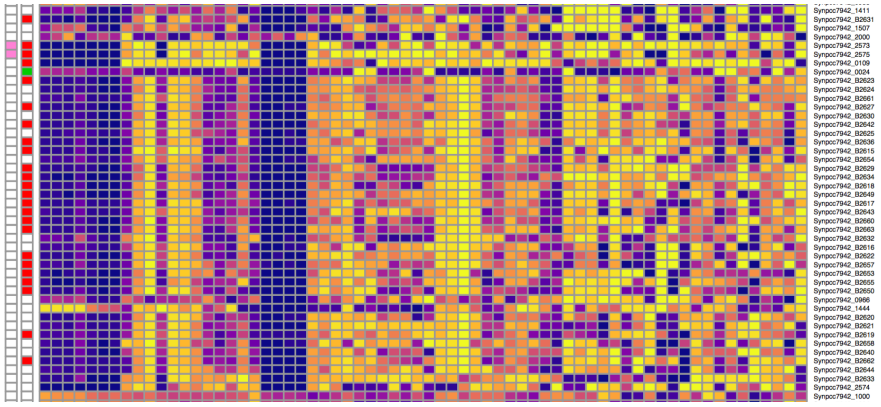
Plastocyanin (Synpcc7942_1088) and cytochrome c553 (Synpcc7942_1630) are not individually required under any of the conditions studied; the worst fitness defect due to inturrupting either of them is plastocyanin



(a) Cluster C3



(b) Cluster C5



(c) Cluster C6

Figure 2.11: Merging DESeq2 and Clust gene count clusters. Three large clusters substantially predicted by both the DESeq2 and Clust algorithms were merged to create the C3, C5, and C6 clusters discussed in the text. Genes predicted by Clust but falling outside the main DESeq2 cluster are included in the analysis but not shown here.

Locus ID	Annotation	Treatment	Mean	SD	Essentiality	Cyanos	GreenCut
0006		fl6_d3	-2.6	0.6	non-essential	no	yes
		pbr_grazed	-2.8	0.5			
0007		fl6_d3	-2.9	0.6	non-essential	no	no
0008		fl6_d3	-3.0	0.8	non-essential	no	no
		pbr_grazed	-2.5	0.4	non-essential	no	no
0072							
0185							
0773	conserved hypothetical protein YCF52	fl4_d3	-3.8	0.5	non-essential	no	yes
0991							
1043		fl4_d3	-5.9	0.6	non-essential	yes	no
		fl6_d3	-4.2	1.3			
		hl	-4.4	0.2			
		pbr_56	-2.0	0.0			
		pbr_grazed	-3.1	0.1			
1201							
1253							
1300							
1760							
1865							
1976	NAD(P)H-quinone oxidoreductase subunit D	fl4_d3	-4.1	0.4	non-essential	yes	no
2155							
2160	alanine-glyoxylate aminotransferase	fl4_d3	-5.4	1.0	beneficial	no	no
		fl5	-3.3	1.2			
		fl6_d3	-3.1	0.8			
		hl	-3.3	0.7			
2334							
2355		fl4_d3	-4.3	0.5	non-essential	no	no
		hl	-5.1	0.4			
2380		fl4_d3	-3.0	0.5	non-essential	no	no
2490							

Table 2.3: Genes in cluster C3, supported by both DESeq2 and Clust, along with fitness values for beneficial and detrimental genes.

Locus ID	Annotation	Treatment	Mean	SD	Essentiality	Cyanos	GreenCut
0379	hypothetical protein	fl6_d3	3.2	0.8	non-essential	yes	no
0388	probable glycosyltransferase	fl4_d1	2.6	0.5	beneficial	no	no
		fl4_d3	3.1	0.9			
		fl6_d3	5.6	1.2			
		pbr_grazed	5.3	0.1			
0479	GTP-binding protein LepA	fl6_d3	3.2	0.4	non-essential	yes	yes
0628	spermidine synthase	pbr_grazed	3.0	0.6	beneficial	no	no
1126	ABC transporter permease protein	fl6_d3	4.1	1.7	non-essential	no	no
		pbr_grazed	4.6	0.3			
1244	ATPase	pbr_grazed	2.9	0.3	beneficial	no	no
1902	putative glycosyltransferase	fl6_d3	4.3	0.9	beneficial	no	no
		ll	-3.9	0.8			
		pbr_56	-3.1	0.3			
		pbr_grazed	4.9	0.1			
1904	hemolysin secretion protein-like	fl4_d3	2.9	0.7	non-essential	no	no
		fl6_d3	3.6	0.3			
		hl	2.6	0.5			
2063	stationary phase survival protein SurE	pbr_grazed	2.9	0.9	non-essential	yes	no
2292	hypothetical protein	fl4_d3	2.9	0.7	non-essential	no	no
		fl6_d3	3.8	0.5			
2293	hypothetical protein	fl4_d3	2.9	0.7	non-essential	no	no
		fl6_d3	3.8	0.7			

Table 2.5: Genes in cluster C5, supported by both DESeq2 and Clust, that also had a fitness effect $>= 2$ in at least one treatment. Additional genes in C5 with no phenotype identified: 0025, 0110, 0169, 0173, 0201, 0266, 0279, 0398, 0431, 0457, 0458, 0465, 0494, 0592, 0594, 0596, 0650, 0707, 0739, 0743, 0811, 0868, 0882, 0936, 0949, 0962, 1004, 1010, 1071, 1084, 1130, 1146, 1152, 1161, 1223, 1228, 1266, 1288, 1295, 1318, 1367, 1393, 1430, 1450, 1452, 1508, 1566, 1569, 1589, 1599, 1620, 1646, 1691, 1716, 1718, 1746, 1748, 1794, 1893, 1921, 1978, 2054, 2056, 2080, 2088, 2093, 2172, 2185, 2264, 2279, 2309, 2335, 2388, 2462, 2469, 2600, B2628, B2635, B2638, B2651, B2659, and B2664.

Locus ID	Annotation	Treatment	Mean	SD	Essentiality	Cyanos	GreenCut
0024							
0109	DNA-binding ferritin-like protein (oxidative damage protectant)-like	hl	-3.8	0.3	non-essential	no	no
		ll	-2.9	0.2			
		nl	-3.4	0.3			
0625	Single-stranded nucleic acid binding R3H	hl	-2.3	0.2	non-essential	no	no
0966							
1444							
2573							
2574							
2575							

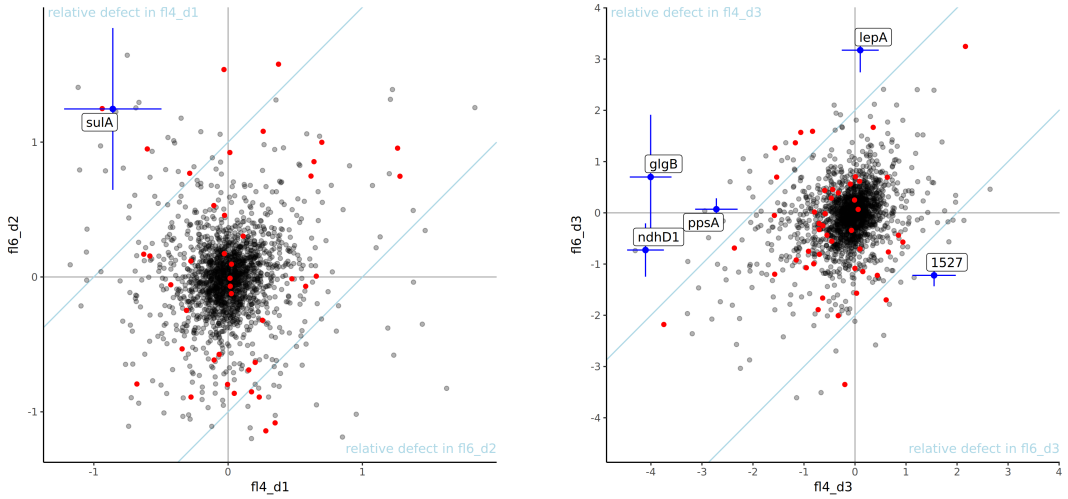
Table 2.7: Genes in cluster C6, supported by both DESeq2 and Clust, along with fitness values for beneficial and detrimental genes. Besides these 8 genes on the chromosome, 40 more were identified on pANL: B2615, B2616, B2617, B2618, B2619, B2620, B2621, B2622, B2623, B2624, B2625, B2627, B2629, B2630, B2631, B2632, B2633, B2634, B2636, B2640, B2642, B2643, B2644, B2649, B2650, B2653, B2654, B2655, B2657, B2658, B2660, B2661, B2662, and B2663.

Locus ID	Annotation	PsbW Cofitness
Synpcc7942_0568	cytosine deaminase	0.87
Synpcc7942_0508	geranylgeranyl reductase	0.85
Synpcc7942_0774	esterase	0.83
Synpcc7942_2282	GAF sensor signal transduction histidine kinase	0.83
Synpcc7942_0817	putative ferric uptake regulator, FUR family	0.82
Synpcc7942_0259	hypothetical protein	0.82
Synpcc7942_0360	hypothetical protein	0.81
Synpcc7942_1110	response regulator receiver domain protein (CheY-like)	0.81
Synpcc7942_0625	Single-stranded nucleic acid binding R3H	0.8
Synpcc7942_2052	probable oligopeptides ABC transporter permease protein	0.79
Synpcc7942_1869	probable cation efflux system protein	0.78
Synpcc7942_1870	Secretion protein HlyD	0.77
Synpcc7942_1947	hypothetical protein	0.76
Synpcc7942_0805	hypothetical protein	0.75
Synpcc7942_0690	ATP-dependent Clp protease adaptor protein ClpS	0.74
Synpcc7942_0434	hypothetical protein	0.74
Synpcc7942_2094	Beta-Ig-H3/fasciclin	0.73
Synpcc7942_0862	hypothetical protein	0.73
Synpcc7942_2324	glutathione synthetase	0.72
Synpcc7942_0650	hypothetical protein	0.72
Synpcc7942_0247	ABC-type permease for basic amino acids and glutamine	0.71
Synpcc7942_1088	plastocyanin	0.71
Synpcc7942_2464	N-acetylmannosamine-6-phosphate 2-epimerase	0.71
Synpcc7942_0771	hypothetical protein	0.71
Synpcc7942_0878	ribonuclease, Rne/Rng family	0.7
Synpcc7942_1682	Sulphate transport system permease protein 2	0.7
Synpcc7942_2355	hypothetical protein	0.7
Synpcc7942_0431	hypothetical protein	0.7
Synpcc7942_0567	hypothetical protein	0.7
Synpcc7942_2266	periplasmic polyamine-binding protein of ABC transporter	0.7
Synpcc7942_0246	extracellular solute-binding protein, family 3	0.69
Synpcc7942_2104	cyanate hydratase	0.69
Synpcc7942_0652	hypothetical protein	0.69
Synpcc7942_0642	bacterioferritin comigratory protein	0.69
Synpcc7942_1628	hypothetical protein	0.68
Synpcc7942_1140	hypothetical protein	0.68

Table 2.8: Genes strongly associated with PsbW (Psb28) via FEBA co-fitness.

Locus ID	N	Treatments
0388	9	fl1 fl3 fl4_d1 fl4_d3 fl6_d3 hl ll pbr_524 pbr_grazed
2160	9	fl2 fl3 fl4_d3 fl5 fl6_d3 hl ll pbr_524 pbr_617
1902	8	fl3 fl4_d1 fl4_d3 fl6_d3 ll pbr_524 pbr_56 pbr_grazed
1043	7	fl4_d3 fl6_d3 hl pbr_524 pbr_56 pbr_617 pbr_grazed
0006	5	fl4_d3 fl6_d3 pbr_524 pbr_617 pbr_grazed
0007	5	fl4_d3 fl6_d3 pbr_524 pbr_617 pbr_grazed
0008	5	fl4_d3 fl6_d3 pbr_524 pbr_617 pbr_grazed
1904	5	fl3 fl4_d1 fl4_d3 fl6_d3 hl
2292	5	fl3 fl4_d1 fl4_d3 fl6_d3 pbr_617
0399	4	fl4_d3 hl ll nl
1410	4	fl1 nl pbr_524 pbr_56
1694	4	fl5 fl6_d3 pbr_617 pbr_grazed
2293	4	fl3 fl4_d3 fl6_d3 pbr_617
0109	3	hl ll nl
0379	3	fl4_d3 fl6_d3 pbr_grazed
0773	3	fl4_d3 fl6_d3 pbr_617
1126	3	fl3 fl6_d3 pbr_grazed
1140	3	fl4_d3 hl pbr_617
1244	3	pbr_56 pbr_617 pbr_grazed
1334	3	fl3 fl4_d3 fl6_d3
1448	3	pbr_524 pbr_56 pbr_617
2099	3	fl4_d3 hl nl
2324	3	fl4_d3 hl pbr_grazed
2355	3	fl4_d3 hl pbr_617
2586	3	fl1 fl2 fl4_d3

Table 2.10: Genes discovered (at a cutoff of 2) in at least 3 different treatments, including those that were removed from other analyses.



(a) Pre-recovery FL4 vs 6 **(b)** Post-recovery FL4 vs 6

Figure 2.12: Differential fitness between FL4 and FL6 before and after recovery outgrowth. Note that the plots use different cutoffs.

under high light with a fitness of -1.14. This is in agreement with previous results [29,15] showing that they must both be knocked out before a growth defect appears.

2.3.5 Comparison of harsh low-light conditions

The FL4 and FL6 patterns caused near-complete growth arrest of *S. elongatus* during their outgrowths, so after 8 days each was transferred back to the Time 0 condition (constant 70 µmol photons m^{-2} sec^{-1}) for recovery (Figure 2.4). Samples from before and after recovery were sequenced. Neither tells the complete story: pre-recovery samples are noisy because there had been fewer generations of growth, but post-recovery samples may primarily measure fitness during the recovery itself. Comparing the pre-recovery samples using a permissive cutoff of +/- 1 reveals only one differentially fit gene: sulA (Synpcc7942_2477). It is beneficial in FL4 but detrimental in FL6. sulA directly destabilizes FtsZ ring formation [32] as part of the stringent response [10], which has recently been shown [23] to be triggered by darkness ("starvation" of light) in *S. elongatus* . The fitness effect here suggests that inhibiting division is an important component of survival under low/dark fluctuating light, but is unhelpful in the PBR light regime. Further study would be needed to determine whether the effect depends on the difference between low and zero light during dark periods. Assuming that preventing division is advantageous during both conditions, the most plausible explanation may be that sulA is required to maintain growth arrest only during periods of complete darkness.

Comparison of the post-recovery samples is also interesting: glycogen-branching enzyme (glgB, Synpcc7942_1085) and ndhD1 (Synpcc7942_1976) are both strongly beneficial in FL4 only, while lepA (Synpcc7942_0479, renamed Elongation Factor 4) is detrimental only in FL6. It is not obvious why an elongation factor found to be necessary for efficient translation during high light in *Arabidopsis* chloroplasts [24] would be detrimental in fluctuating light, but one clue is that it is moderately co-fit with Synpcc7942_0388 and Synpcc7942_1902, which are detrimental under multiple FL conditions. The glgB and ndhD effects are easier to interpret: they suggest that alternative electron sinks (glycogen synthesis and oxidation of quinones/plastoquinones) are important in FL4 during the repeated sudden dark-to-light transitions. This is expected since the Calvin/Benson cycle takes time to

locusId	treatments	desc
Synpcc7942_1566	fl1	polyphosphate kinase
Synpcc7942_1674	fl1	hypothetical protein
Synpcc7942_1996	fl1, fl2	hypothetical protein
Synpcc7942_2586	fl1, fl2	hypothetical protein
Synpcc7942_0095	fl2	two component transcriptional regulator, winged helix family
Synpcc7942_2160	fl2, fl3	alanine-glyoxylate aminotransferase
Synpcc7942_1334	fl3	aminodeoxychorismate synthase, subunit I
Synpcc7942_2293	fl3	hypothetical protein
Synpcc7942_2292	fl3	hypothetical protein
Synpcc7942_1126	fl3	ABC transporter permease protein

Table 2.11: Genes with fitness effects during fast, medium, and slow 20/500 µmol photons m^{-2} sec^{-1} square waves.

reach full sink capacity, and was found to impose a "penalty" on photosynthetic productivity in both *S. elongatus* and *C. reinhardtii* during fluctuating light on similar timescales [18].

2.3.6 Effect of fluctuation period

S. elongatus cells contain polyphosphate bodies in their nucleoids. When grown in a diurnal cycle the number of bodies increases during light periods, whereas their size increases in the dark [51]. They have been shown to associate closely with DNA, and are hypothesized to provide phosphate for DNA replication [51]. The polyphosphate kinase identified here (Synpcc7942_1566) could be involved in regulating that process. If so, the fitness benefit (+2) of inturrupting it only in the fastest condition suggests that it responds fast enough to cope with light/dark cycles on the scale of hours or minutes, but not seconds. Inturrupting the two-component response regulator Synpcc7942_0095 also confers a positive fitness effect (+2), but only on the 5 minute timescale. This could also be explained in a similar manner if it operates on a timescale of minutes: the 30-second fluctuations may be fast enough to average out, and 4-hour fluctuations may be handled by other mechanisms overlapping with the high light response. Either theory would need to be tested.

2.3.7 Effect of sudden dark-to-light transitions

No genes were found with differential fitness > 2 (or even 1) between the FL2 and FL5 treatments. This is also an interesting result, suggesting that the suddenness of low-to-high light transitions makes little difference to *S. elongatus* cells.

2.3.8 Constant HL is more stressful than 30 min pulses

Constant HL is much more stressful than 30 minute periods of HL interrupted by low(er) light. Only glgB (glycogen branching enzyme, see 2.3.5 above) was more important in the slow FL treatment, consistent with a role as the initial electron sink upon shift to HL.

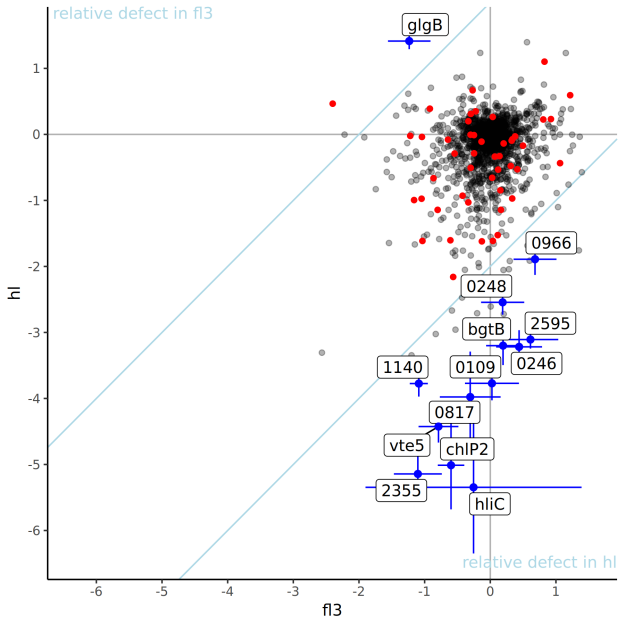


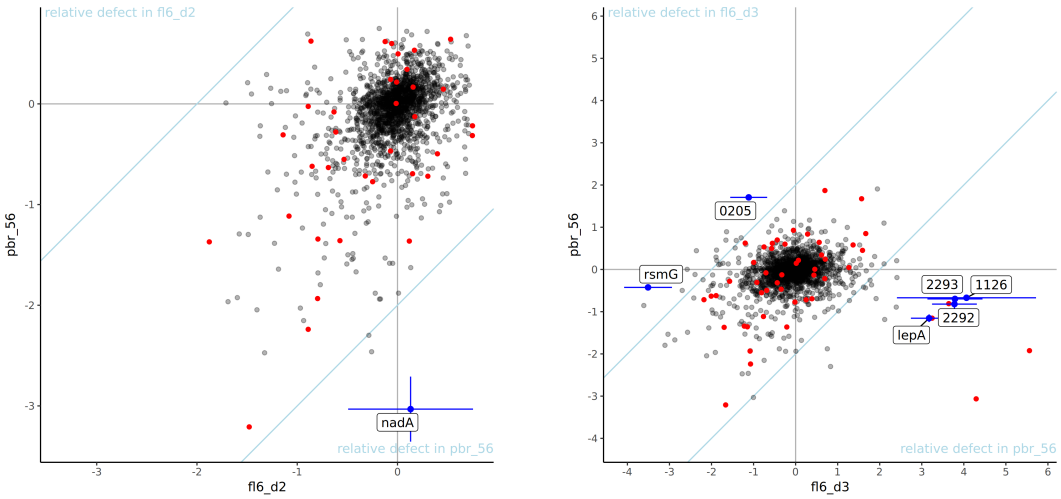
Figure 2.13: HL vs FL3 (30 min HL every 4 hr).

2.3.9 The modeled ePBR light regime approximates growth in an ePBR

Perhaps surprisingly, given the growth response, gene fitness at both timepoints was broadly similar to fitness in the photobioreactors FL6 was designed (Figure 2.3) to mimic. The only prominent exception vs pre-recovery is nadA (Synpcc7942_1448, quinolate synthetase), which has a defect in the photobioreactors. It is required for *de novo* synthesis of NAD from aspartate[17]. The process which should be beneficial but not essential in both conditions. The protein contains a (4Fe-4S)2+ cluster and is inactivated by exposure to oxygen[41], so a reasonable interpretation is that bubbling the PBRs with air increases its degradation rate. A follow-up experiment including similarly aerated flasks would be necessary to confirm that.

Post-recovery, the two treatments were still similar. The main difference is the benefit of knocking out LPS export pathway genes during the recovery phase, a phenomenon found under other conditions as well; see Section 2.3.3. Synpcc7942_0205, which increases fitness in the PBR but decreases it in FL6, is also involved in NAD synthesis. Its homolog in *Synechocystis* has been shown[46] to be bifunctional, able to act as a nicotinamide mononucleotide adenylyltransferase and also a 'Nudix' hydrolase. A recent study[37] demonstrated that several Nudix hydrolases are involved in adjusting the relative levels of redox carriers in Arabidopsis chloroplasts, so the role of Synpcc7942_0205 here may be more complicated. It would make sense for its contribution to depend on growth rate and photosynthetic conditions, and therefore to be different between the pre- and post-recovery samples.

Two last two genes with large differential fitness in the post-recovery outgrowth are rsmG (rRNA small subunit methyltransferase, Synpcc7942_0267)



(a) Pre-recovery (dilution 2)

(b) Post-recovery (dilution 3)

Figure 2.14: Fitness in PBRs vs PBR light pattern (FL6) in flasks, before and after recovery of the flask culture under constant 70 µmol photons m^{-2} sec^{-1} for an additional 4 generations.

2.3.10 Genes specific to constant light

Two genes, Synpcc7942_0109 and Synpcc7942_0399, caused fitness defects when interrupted in any of the constant light levels (averaging -3.4 and -2.7 respectively), but no significant defects in fluctuating light or ePBRs. The only exception is that Synpcc7942_0399 also caused a defect (-3.3) during recovery from FL4; the recovery outgrowth was also under constant 70 µmol photons m^{-2} sec^{-1} . Synpcc7942_0109 is annotated as a DNA-binding ferritin-like protein that could protect against oxidative damage and be involved in iron storage. Its expression was also correlated with many genes on the pANL plasmid (Section 2.3.2). Synpcc7942_0399 is unknown but conserved across cyanobacteria.

2.3.11 Long-term growth in photobioreactors

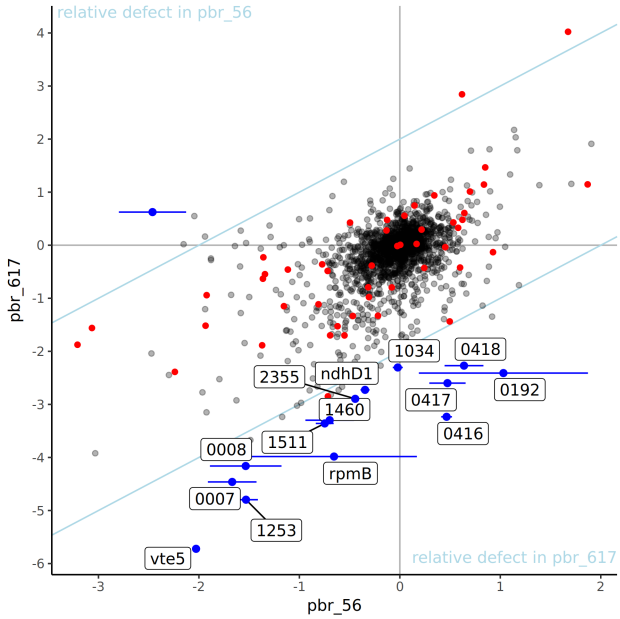


Figure 2.15: Short- vs long-term PBR fitness.

2.3.12 Grazer-resistant biofilms are a natural feature

Collapsed PBR cultures were examined under the microscope and found to be filled with small, fast-moving, eukaryotic heterotrophs. The heterotrophs remained after recovery, but coexisted with large masses of *S. elongatus* cells (Figure 2.19). They were observed skimming rapidly along the surfaces of the biofilms, but presumably do not impact the cells in their interior.

It was unclear whether heterotrophs caused the biofilm, or only appeared afterward and scavenged dead cells. *S. elongatus* has also been shown to have an "apoptosis-like" mechanism in which it releases a compound into the media causing rapid oxidative damage and death of its own cells, as well as those of other photosynthetic microbes[13]. That mechanism could be abnormally activated in some subset of the TnSeq mutants, causing collapse of the whole culture only under particular conditions. As shown in Figure 2.20, supernatant from a freshly collapsed had no effect on wild type, while the washed pellet caused biofilm formation. Grazing heterotrophs were also present in the pellet-challenged culture. The TnSeq biofilm culture remained a biofilm in all conditions as expected, but also produced a population of planktonic cells in the supernatant-challenged culture only.

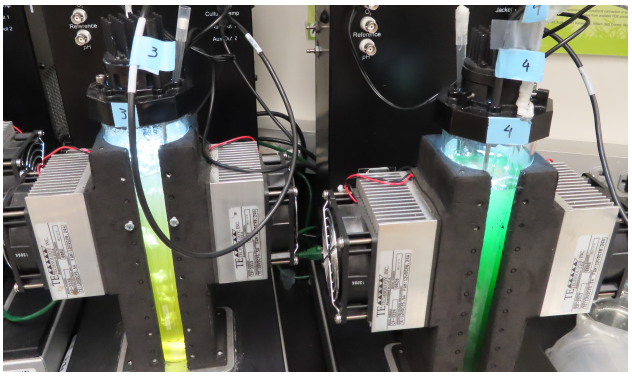


Figure 2.17: PBR culture collapse. **Left:** collapsed culture. **Right:** healthy culture.

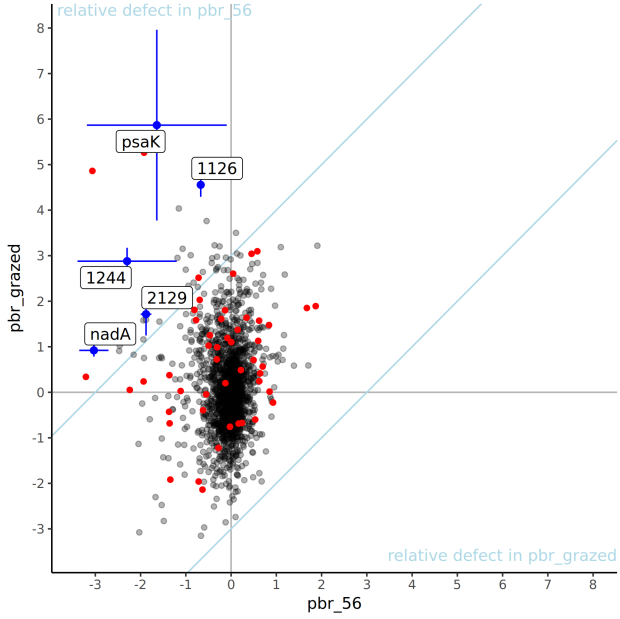
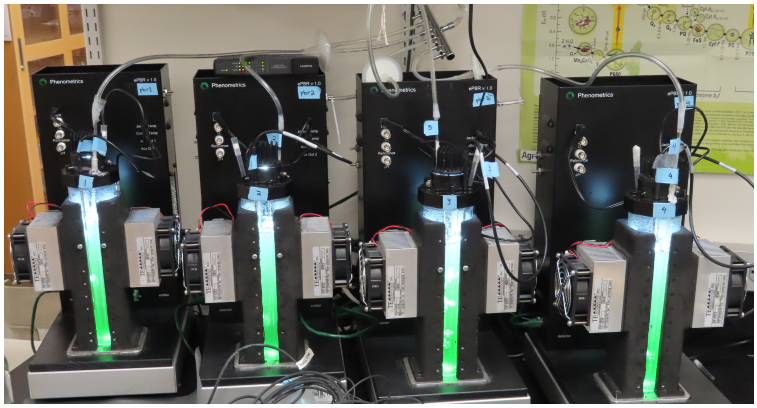


Figure 2.16: Grazed vs healthy PBR fitness.

Consistent with biofilms as a natural predation defense, the PBR grazed treatment (2 replicate PBRs) showed a relatively normal distribution of mutant fitness. It was skewed far enough to fail the FEBA quality controls and to necessitate removal from clustering (Figure 2.9), but still showed far more diversity than would be expected if only a few resistant mutants had survived a severe bottleneck. Many genes have differential fitness, but the strongest (Figure 2.16) are mostly LPS export pathway genes (Section 1.3.8), consistent with a role in grazer resistance and/or rapid recovery. Surprisingly, there is also an even stronger hit in psaK, a small protein associated with the periphery of PSI monomers[59]. It is unclear what effect is would have on biofilm formation or grazer resistance.

(a) Growth chambers



(b) Photobioreactors

Figure 2.18: Growth setup for the fluctuating light experiments. **a.** Two treatments at a time were exposed to FL, isolated from each other by a cardboard light barrier (middle), and an additional blackout curtain for low light treatments (bottom half). **b.** The PBR FL treatments were grown in 4 photobioreactors (Phenometrics ePBR v1.1), bubbled with air from an aquarium pump.

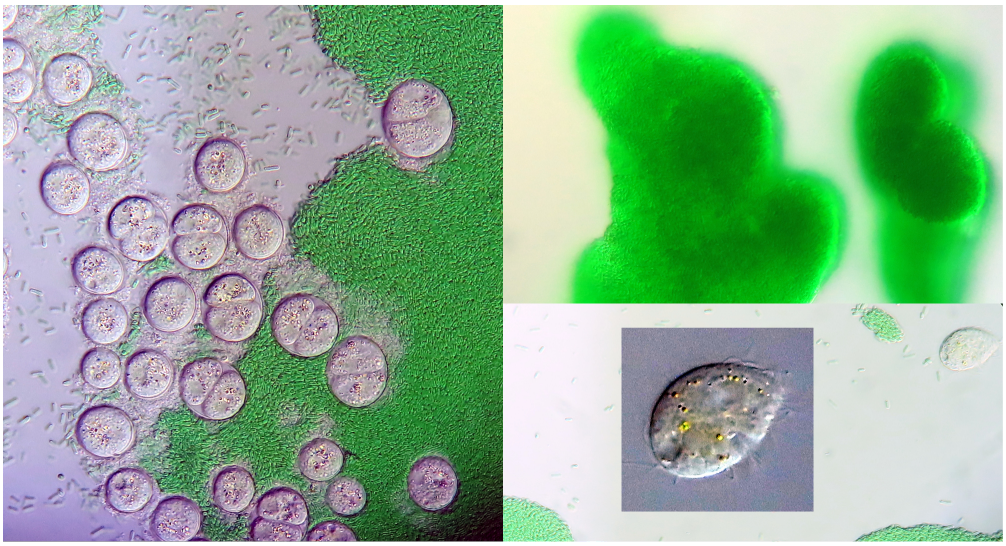


Figure 2.19: Grazers in collapsed PBR culture. Zeiss AxioImager A.2 400X. **Left:** unidentified bodies resembling Amoeba HGG1 cysts found by Simkovsky *et al.*[54]. **Top right:** a 3-dimensional biofilm formed by *S. elongatus*in the presence of the predator. **Bottom right:** the unidentified predator compared to *S. elongatus*cells, and enlarged (inset).

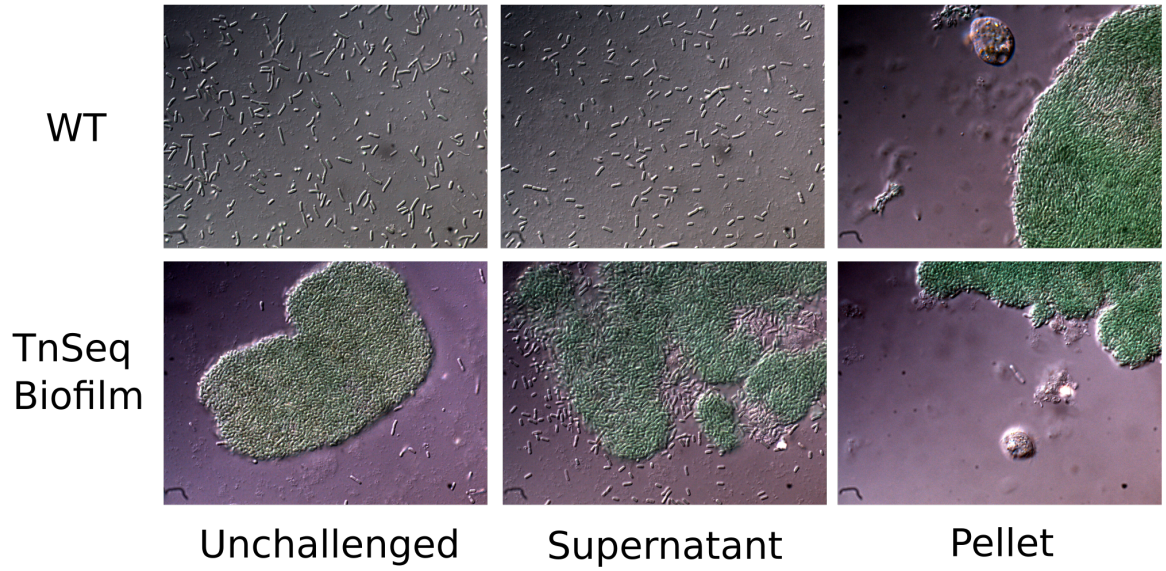


Figure 2.20: Grazers trigger the biofilm response. Differential interference contrast (DIC) microscope images at 100X of *S. elongatus*and the TnSeq biofilm culture. Each was challenged with supernatant or pellet derived from a collapsed TnSeq PBR culture, then imaged the next day.

2.4 Discussion

Gene counts were noisier in this set of experiments compared to the constant-light outgrowths in Chapter 1. Several differences probably contribute: shorter and fewer reads per sample, less uniform library prep due to differences in the number and growth stage of cells that could be sampled between treatments. However I suspect the major difference is that the first set of experiments started directly from my original outgrowth, whereas these went through an extra cycle of freezing, thawing, and recovery under standard conditions. The treatments with the best resolution of phenotypes here were the ones grown longest, so future experiments with the *S. elongatus* TnSeq library should aim to continue the outgrowths past 7 generations.

I found only relatively minor phenotypes for interruptions of flv1/3, despite picking the FL conditions specifically to detect them. I expect that starting with an earlier copy of the library, continuing the outgrowths past 7 generations, or deeper sequencing would reveal phenotypes for them under these same experimental conditions. Unfortunately, if true that also suggests this study may have missed other genes of similar effect.

The particulars of this study did lead to another interesting finding, though: in contrast to the Rubin paper[48], I found prominent deleterious genes under multiple conditions. The most likely explanation is that they are not evident at first, but come to dominate as the library goes through successive outgrowths. I also found them to be more prominent in conditions involving stress followed by recovery, which the initial study sought to minimize.

Harsh fluctuating light never continues for days at a time–or even for one entire day—in natural systems, so no organism would be expected to evolve optimal responses to it. This study bears that out, finding a range of genes that can be knocked out to increase fitness under various FL regimes. Knocking out the positive-fitness enzymes and regulatory factors identified here could be a promising strategy to increase yields in reactors with fluctuations of similar periods (around 30 seconds to 5 minutes/cycle), or even faster. The impromptu grazer resistance screen presented here could also be useful for engineering more effective biofuels strains, since contamination is a common challenge in large-scale cultures[20].

2.5 Future Directions

The *S. elongatus* TnSeq library is a valuable tool for dissecting photosynthetic pathways, and this study shows that there are many genes of large individual effect remaining to be found under various light conditions. Rather than comparing several discrete conditions as I have done here, one promising direction would be to miniaturize the process to study light and/or nutrient gradients, much as is already being done with heterotrophic bacteria [45]. Graham *et al.* [18] used an LED projector to illuminate individual wells in a 384-well plate with a range of fluctuating light patterns. The same setup could be used to grow the RB-TnSeq library, allowing us to investigate much more subtle photosynthetic phenotypes than have been possible before by isolating fitness defects to specific light conditions. Alternatively, the experiment could be scaled up by growth in multiple large plates, the way I did my first high-throughput growth curves (Appendix A).

Chapter 3

ShortCut: A domain-specific language to facilitate reproducible phylogenomic searches

3.1 Summary

Phylogenomic "cuts" are one name for lists of genes whose pattern of evolutionary conservation suggests they may be important for a process of interest, such as photosystem assembly or chloroplast retrograde signaling. The GreenCut[34,27] and similar lists have historically been successful at identifying candidates for further study, but no standard methodology exists for making them or measuring their quality. They tend to involve a small number of common tools—BLAST searches, set subtraction, perhaps construction of gene or genome trees, and manual curation—that must be combined in unique ways depending on the organisms and traits involved. Because a variety of workflows are needed, the overall process is difficult to automate with any single program. ShortCut is an attempt to overcome that limitation using a domain-specific language: rather than one finished algorithm, it provides many small functions that can be rearranged to create algorithms tailored to specific questions. It also provides a novel method of measuring their robustness to changes in the search parameters. The scripts are interactive, facilitating quick comparison of many possible methods to find the most reliable list of candidate genes. But more importantly, they are reproducible; their adoption has the potential to accelerate research by allowing existing searches to be reused in future projects, or continuously updated as new genomes are published.

3.1.1 Motivation

This project began as a means to find genes likely involved in PSII assembly, but quickly took on a life of its own. Robert Calderon, a previous student in the lab, was studying the PSII assembly factor RBD1[11]. He noticed that it was absent from the recently-discovered cyanobacterium *Atelocyanobacterium thalassa*[19], which adopted a symbiotic nitrogen-fixing lifestyle and subsequently lost PSII. He found several other genes with similar profiles manually, and recruited me to help with a larger-scale automated search.

I wrote several early iterations of the search in Bash (the standard language for scripting Linux commands), but found it difficult and error-prone to keep both the biology and programming details in mind simultaneously. I began trying to separate the code from a minimal description of the problem to be solved so each part of it could be verified on its own. Eventually I realized that my "minimal description" constituted a domain-specific language, and the code for translating that description into a series of steps could be better written as a compiler for that language. Furthermore, I noticed that this is a general problem faced by many biologists. My language

```
pnas[journal] AND (photosynthesis OR cyanobacteria)
```

```
{ stdenv, fetchurl, cmake, perl, zlib, unzip }:

stdenv.mkDerivation rec {
  name = "mmseqs2-${version}";
  version = "67b4ca0708a5cd2a0f87220b97f11e562f6c7842";
  src = fetchurl {
    url = "https://github.com/soedinglab/MMseqs2/archive/${version}.zip";
    sha256 = "0fm3k2y4qc3830pvgixjjsra8ssfdsfr7yyzrcca42pmr750rwzc";
  };
  buildInputs = [ cmake perl zlib unzip ];
  postUnpack = ''
    patchShebangs MMseqs2-${version}
  '';
  preConfigure = ''
    cmakeFlags="$cmakeFlags -DCMAKE_BUILD_TYPE=RELEASE \
      -DCMAKE_INSTALL_PREFIX=$out -DHAVE_AVX_EXTENSIONS=0 -DHAVE_AVX2_EXTENSIONS=0"
  '';
}
```

Listing 1: DSLs range from simple to complicated.

was already close to general enough to be useful in other studies... why not go all the way, and develop something that can be widely reused?

3.1.2 Background

Domain-specific languages (DSLs) are distinguished from general-purpose programming languages by being relatively simple and limited. They can only express solutions to a particular type of problem, but they do it succinctly. Like technical jargon, they reduce ambiguity and make communication more efficient. A simple example is the text-based "advanced search" features found on websites such as Google or PubMed. Each website uses a different syntax that makes one type of search easier by not needing to handle the other use cases. A more advanced example is Nix[14], which I use as part of ShortCut to reproducibly install dependences (see Listing 1). It borders on being general-purpose.

The key feature of a problem well suited to being solved with a domain-specific language is that it should have a relatively small number of basic operations, but a large number of ways they may need to be combined. They are designed to hide irrelevant detail while exposing anything that a "domain expert" might need to change. In the case of ShortCut irrelevant details are things like the command line arguments required by each program or where to save intermediate files, and details that need to be exposed are things like which sequence files to compare, which search programs to use at each step, and in some cases an e-value cutoff.

The difference between a program and a programming language interpreter is fuzzier than most non-programmers imagine; "code" can be viewed as just an unusually flexible configuration file for the main program, called the interpreter. ShortCut started as a simple BLAST script, then grew into a confusing series of scripts managed by configuration files (See Figure 2). When those became unmanageable, the interpreter was written to generate a similar series of steps from more human-readable (and therefore less error-prone) descriptions.

```
[blastconserved]
script        = bin/blast.sh
skip          = false
verbosity     = 3
cache_dir     = tmp/blast_cache
genomes_dir   = tmp/fna
output_csv    = tmp/blast1.csv
queries_fasta = tmp/conserved.faa

[comparegenes]
script    = bin/comparegenes.R
skip      = false
verbosity = 3
file1     = tmp/ps2cut_robbie.txt
file2     = tmp/ps2cut_jeff.txt

[choosegenomes]
script         = bin/choosegenomes.R
skip           = false
verbosity      = 3
include_cutoff = 1e-3
include_regex  = .*
input_csv      = tmp/blast1.csv
output_txt     = tmp/genomes.txt

[blastvariable]
script        = bin/blast.sh
skip          = false
verbosity     = 3
cache_dir     = tmp/blast_cache
exclude_regex = jgi_IMG-2502171196
genomes_dir   = tmp/fna

list_genomes  = tmp/genomes.txt
output_csv    = tmp/blast2.csv
queries_fasta = tmp/plastidcut2.faa

[choosegenes]
script         = bin/choosegenes.R
skip           = false
verbosity      = 3
exclude_cutoff = 1.0
exclude_regex  = jgi_IMG-2502171196
include_cutoff = 15
include_regex  = .*
input_csv      = tmp/blast2.csv
output_txt     = tmp/ps2cut_jeff.txt
```

Listing 2: Complicated configuration file or simple language? An early description of the PSIIcut lists scripts to run and the arguments to run them with.

3.2 Results

3.2.1 Example code

First, what does the code look like? Based on early feedback, the most common first step for new users will probably be finding reciprocal BLAST best hits between species not yet published in the major databases. Here is an example with BLASTp.

```
mgen = load_faa "data/Mycoplasma_genitalium_protein_refseq.faa"
syne = concat_faa [gbk_to_faa "cds" (load_gbk "data/SynPCC7942_chr.gbk"),
                   gbk_to_faa "cds" (load_gbk "data/SynPCC7942_pANL.gbk")]
result = blastp_rbh 1e-10 mgen syne
```

Listing 3: First steps toward a cut: reciprocal best BLASTp between two proteomes, one of which needs to be converted to FASTA first by extracting features labeled "cds" in two GenBank files and concatenating them.

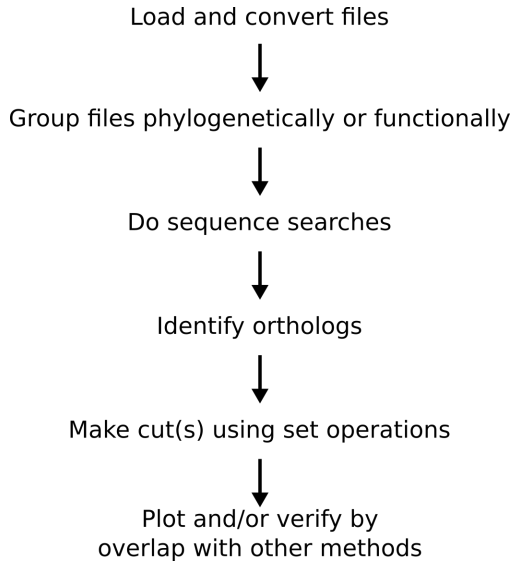


Figure 3.1: Steps in a typical cut script.

With code saved to `my-first-blast.cut` in the same directory as the proteome files, the script can be run like this:

```
shortcut --script my-first-blast.cut --output results.txt
```

It should result in a `results.txt` like this:

```
WP_010869318.1    Synpcc7942_0563   47.583      662   325   5     11    ...
WP_014894509.1    Synpcc7942_0848   49.796      980   444   9     8     ...
WP_014894005.1    Synpcc7942_1662   33.424      368   226   8     16    ...
WP_009885724.1    Synpcc7942_1795   33.571      140   92    1     3     ...
WP_010869493.1    Synpcc7942_1100   27.209      430   279   11    23    ...
...
```

For simple questions like "How many LHC (light harvesting complex) proteins does this genome encode?", the only remaining step would be to make a list of unique hit IDs with the `extract_targets` function. More detailed scripts can start simple, then be elaborated until they follow a pattern more like Figure 3.1 and look more like the GreenCut2 (Listing 4). They might also build from another cut as a starting point, compare multiple search methods, or include a custom script to analyze/plot the output.

3.2.2 Write searches interactively

Cut scripts can be written in a text file and run with `shortcut --script`, but they can also be written interactively with `shortcut --interactive`. There are several advantages:

- Tab-completion of types, function, and variable names
- Type-checking of each line as it is written (Section 3.2.7)
- Ability to print or plot variables, checking that they make sense

```
# Two major criteria were used to generate the inventory of GreenCut2 proteins...

# First, the Chlamydomonas proteins of the GreenCut2 must have an ortholog
# encoded by the nuclear genomes of the green lineage organisms A. thaliana
# (TAIR v8), P. patens (JGI v1.1), O. sativa (japonica) (TIGR v5.0), P.
# trichocarpa (JGI v1.1), ...
chlamy  = load_faa "Creinhardtii_281_v5.5.protein_primaryTranscriptOnly.fa"
athal   = load_faa "TAIR8_pep_20080412"
physco  = load_faa "Physcomitrella_patens_UP000006727.faa" # TODO try removing it for Robbie
osativa = load_faa "Oryza_sativa.IRGSP-1.0.pep.all.fa"
ptricho = load_faa "Populus_trichocarpa.Pop_tri_v3.pep.all.fa"
greens = [chlamy, athal, physco, osativa, ptricho]

# ... and one of the three Ostreococcus species with fully
# sequenced genomes (O. lucimarinus (JGI v2.0), O. tauri (JGI v2.0), or
# Ostreococcus sp. RCC809 (JGI v2.0)).
oluci   = load_faa "Ostreococcus_lucimarinus_UP000001568.faa"
otauri  = load_faa "Ostreococcus_tauri_UP000009170.faa"
orcc809 = load_faa "OstreococcusRCC809v2.allModels.proteins.fasta"
ostreococcus = [oluci, otauri, orcc809]

# Second, proteins with orthologs in the green lineage organisms listed above
# were only included in the GreenCut2 if they had no ortholog in Pseudomonas
# aeruginosa str. PA01, Staphylococcus aureus subsp. aureus str. N315,
# Dictyostelium discoideum AX4, Phytophthora sojae, Neurospora crassa OR74A,
# Methanosarcina acetivorans str. C2A, Sulfolobus solfataricus str. P2,
# Caenorhabditis elegans, and Homo sapiens.
pseudomonas = load_faa "Pseudomonas_aeruginosa_PAO1_107.faa"
staph       = load_faa "Staph_aureus_N315.faa"
disco       = load_faa "Dictyostelium_discoideum_UP000002195.faa"
p_sojae     = load_faa "Phytophthora_sojae_UP000002640.faa"
neurospora  = load_faa "Neurospora_crassa_UP000001805.faa"
methano     = load_faa "Methanosarcina_acetivorans_UP000002487.faa"
saccharo    = load_faa "Saccharolobus_solfataricus_UP000001974.faa" # formerly sulfolobus
c_elegans   = load_faa "Caenorhabditis_elegans_UP000001940.faa"
human       = load_faa "GRCh38_latest_protein.faa"
non_ps = [pseudomonas, staph, disco, p_sojae, neurospora, methano, saccharo, c_elegans, human]
proks  = [pseudomonas, staph, methano, saccharo]

# Searches for orthologs in Cyanidioschyzon merolae str. 10D, ...
cmero = load_faa "Cmero_UP000007014.faa"
red_algae = [cmero]

# ... Thalassiosira pseudonana (JGI v2.0), and Phaeodactylum tricornutum (JGI
# v3.0) also were conducted, but for inclusion in the GreenCut2, we did not
# require that a Chlamydomonas protein have an ortholog in these organisms.
ptr = load_faa "Phaeodactylum_tricornutum_UP000000759.faa"
tps = load_faa "Thalassiosira_pseudonana_UP000001449.faa"
diatoms = [ptr, tps]

everyone = greens | ostreococcus | non_ps | red_algae | diatoms
```

```
# A mutual best BLASTP hit (E-value <1e-10) was used to establish orthology to
# a Chlamydomonas protein. Additional eukaryotic proteins that were not a
# mutual best hit but had >50% amino acid identity to a Chlamydomonas protein
# within an ortholog cluster were selected as in-paralogs (co-orthologs
# throughout).
fwd_hits     = blastp_each 1e-10 chlamy everyone
fwd_euk_hits = blastp_each 1e-10 chlamy (everyone ~ proks)
rev_hits     = blastp_rev_each 1e-10 chlamy everyone
rb_hits      = reciprocal_best_all fwd_hits rev_hits
more_hits    = concat_bht (filter_pident_each 50 fwd_euk_hits)
families     = greencut2_families rb_hits more_hits

greencut2 = ortholog_in_all   families [chlamy, ptricho, athal, osativa, physco]
          & ortholog_in_min 1 families ostreococcus
          ~ ortholog_in_any   families non_ps

plantcut2   = greencut2 & ortholog_in_all   families red_algae
diatomcut2  = greencut2 & ortholog_in_min 1 families diatoms
viridicut2  = greencut2 ~ plantcut2 ~ diatomcut2
plastidcut2 = plantcut2 & diatomcut2

# save lists of Chlamydomonas IDs for each cut
cre_ids = extract_ids chlamy
greencut2_cre   = any greencut2   & cre_ids
plantcut2_cre   = any plantcut2   & cre_ids
diatomcut2_cre  = any diatomcut2  & cre_ids
viridicut2_cre  = any viridicut2  & cre_ids
plastidcut2_cre = any plastidcut2 & cre_ids

# and same with Arabidopsis
at_ids = extract_ids athal
greencut2_at   = any greencut2   & at_ids
plantcut2_at   = any plantcut2   & at_ids
diatomcut2_at  = any diatomcut2  & at_ids
viridicut2_at  = any viridicut2  & at_ids
plastidcut2_at = any plastidcut2 & at_ids

# The final output lists:
result =
  [ greencut2_cre
  , plantcut2_cre
  , diatomcut2_cre
  , viridicut2_cre
  , plastidcut2_cre
  , greencut2_at
  , plantcut2_at
  , diatomcut2_at
  , viridicut2_at
  , plastidcut2_at
  ]
```

Listing 4: Re-implementation of the GreenCut2 in ShortCut based on Karpowicz *et al.*[27].

```
Welcome to the ShortCut interpreter!
Type :help for a list of the available commands.

shortcut —▸ :help
You can type or paste ShortCut code here to run it, same as in a script.
Unlike in a script though, you can also evaluate and redefine variables.
There are also some extra commands specific to --interactive mode:

:help      to print info about a function or filetype
:load      to clear the current session and load a script
:reload    to reload the current script
:write     to save the whole script (or dependencies of a specific variable)
:needs     to show which variables a given variable depends on
:neededfor to show which variables depend on the given variable
:drop      to discard the current script (or a specific variable)
:quit      to discard the current script and exit the interpreter
:type      to print the type of an expression
:show      to print an expression along with its type
:!         to run the rest of the line as a shell command (disabled in --secure mode)
```

Listing 5: Interactive commands specific to the interpreter.

The interpreter can be used to set up and typecheck a long script to be run later, or to run (evaluate) specific variables live. A common pattern for combining the two modes is to experiment live on a subset of the data, then run the same script on everything separately; see Section 3.3.2).

3.2.3 Save work and reproduce it later

At any point, work in the interpreter can be saved by writing it to a script. Either the whole script with `:write all-my-work-so-far.cut`, or everything needed for a particular variable with

`:write myvar just-the-steps-to-myvar.cut`. This is a major advantage compared to most programming languages, because people typically experiment for a while before deciding they need to save their work. At that point, other languages have no built-in way to reconstruct the series of steps needed to get to the current interesting result; one has to learn from experience to write down the steps in a separate script as they go. ShortCut makes reproducibility the default instead.

3.2.4 Reliable software installation

Bioinformatics software is typically messy and difficult to install. Sometimes it can so idiosyncratic that another paper is required to improve the installation process[3], or the authors may not even grant permission to try[57]. ShortCut reproducibly installs all the programs it depends on and guarantees that they will be compatible versions. See Listing 8 for instructions.

```
Welcome to the ShortCut interpreter!
Type :help for a list of the available commands.

shortcut –▸ load_list "data/proteomes-mmseqs.txt"
["data/Mycoplasma_agalactiae_small.faa",
 "data/Mycoplasma_bovis_small.fa",
 "data/Mycoplasma_genitalium_small.faa",
 "data/Mycoplasma_hyopneumoniae_small.faa"]

shortcut –▸ load_faa_each (load_list "data/proteomes-mmseqs.txt")
[>gi|290752267|emb|CBH40238.1|
MNINSPNDKEIALKSYTETFLDILRQELGDQMLYKNFFANFEIKDVSKIGHITIGTTNVTPNSQYVIRAY
ESSIQKSLDETFERKCTFSFVLLDSAVKKKVKRERKEAAIENIELSNREVDKTKTFENYVEGNFNKEAIR
IAKLIVEGEEDYNPIFIYGKSGIGKTHLLNAICNELLKKEVSVKYINANSFTRDISYFLQENDQRKLKQI
RNHFDNADIVMFDDFQSYGIGNKKATIELIFNILDSRINQKRTTIICSDRPIYSLQNSFDARLISRLSMG
...

shortcut –▸ :load my-first-blast.cut
mgen = load_faa "data/Mycoplasma_genitalium_protein_refseq.faa"
syne = concat_faa [gbk_to_faa "cds" (load_gbk "data/SynPCC7942_chr.gbk"),
                   gbk_to_faa "cds" (load_gbk "data/SynPCC7942_pANL.gbk")]
result = blastp_rbh 1e-10 mgen syne

shortcut –▸ result
WP_010869318.1    Synpcc7942_0563   47.583      662   325   5     ...
WP_014894509.1    Synpcc7942_0848   49.796      980   444   9     ...
WP_014894005.1    Synpcc7942_1662   33.424      368   226   8     ...
WP_009885724.1    Synpcc7942_1795   33.571      140   92    1     ...
WP_010869493.1    Synpcc7942_1100   27.209      430   279   1     ...
...
```

Listing 6: The interactive ShortCut interpreter. You can load, save, run and redefine particular variables live.

```
shortcut —▸ :help score_replacments

score_replaced : <outputnum> <inputvar> <inputlist> -> <input>.scores

Shorthand for the common use case where you want to score your cut by some
metric, for example number of hits, then vary one of the input parameters and
see how the score changes.

Arguments are the same as for replace_each, except the output variable must be a
number. It returns a table of scores and the corresponding inputs. You can
inspect them yourself, or plot them in ShortCut.

Examples:

linegraph
  "How does the cutoff change the number of hits?"
  (score_replaced (length hits) initial_cutoff alternate_cutoffs)
```

Listing 7: Built-in help with modules, types, and functions.

```
# install Nix if needed
curl https://nixos.org/nix/install | sh

# install ShortCut
git clone https://github.com/jefdaj/shortcut.git
cd shortcut
nix-build --verbose -j$(nproc)

# run self-tests
./result/bin/shortcut --test
```

Listing 8: Installation of ShortCut using the Nix package manager[14]. The last step is optional and runs about 700 self-tests after installation to confirm all the included programs are working. This process should work on Linux and MacOS; Windows is unsupported for now.

ShortCut includes a variety of common sequence search programs:

- NCBI BLAST+
- CRB-BLAST
- HMMER
- DIAMOND
- MMSeqs2

And several different ortholog-finding programs:

- Orthofinder
- SonicParanoid
- Reciprocal best hits (implemented here)
- Greencut2 gene families (re-implemented here based on the published method)

There are also several other miscellaneous programs I found useful, including MUCSLE for sequence alignments and BUSCO to assess genome quality based on how many of the expected universally conserved single-copy orthologs it contains.

3.2.5 Simplified commands and file organization

One of the main goals is to simplify the use of each program as much as possible while maintaining enough flexibility to meet most use cases. For example, the Bash commands and ShortCut commands in Listing 3.2 are roughly equivalent, but the ShortCut code is much closer to how it would typically be described out loud or in a methods section. Biologically important details like whether to make a protein or nucleic acid database are controlled implicitly by choosing the `load_faa` (FASTA amino acid) and `blastp` functions.

Each time it evaluates a variable, ShortCut writes the result to a file of the same name in the temporary directory (`/.shortcut/vars` by default). That eliminates having to remember what things were named, but still makes them available to open in another program when needed.

3.2.6 Math and set Operations

Most of what ShortCut does is glue together external scripts. But there are also a few common operations built in to help compare the results. Set operations are a natural way to express phylogenomic cuts (See Figure 3.3 and Section 4), and math is often useful for defining quality metrics (Listing 9).

```
n_found        = length (mycut & previously_known)
percent_found  = n_found / length mycut * 100
percent_missed = 100 - (n_found / length previously_known * 100)
```

Listing 9: Three simple ways of estimating the quality of a cut: number of known genes rediscovered, percentage of known genes rediscovered, and percentage of known genes missed (not rediscovered).

```
# Look manually for unusual characters in sequence IDs (commas in this case)
# and remove them. This step varies based on your intended downstream use.
sed 's/,/-/g' query-transcriptome.faa   > query-transcriptome-safe.faa
sed 's/,/-/g' subject-transcriptome.faa > subject-transcriptome-safe.faa

# Make the BLAST database
mkdir -p blastdb
makeblastdb -in subject-transcriptome-safe.faa -out blastdb/subject \
        -title subject -dbtype prot -parse_seqids

# Then run BLAST
blastp -db blastdb/subject -query query-transcriptome-safe.faa \
        -evalue 1e-10 -outfmt 6 > hits.tsv
```

```
# Load files, then run BLAST
query   = load_faa "query-transcriptome.faa"
subject = load_faa "subject-transcriptome.faa"
blastp 1e-10 query subject
```

Figure 3.2: Roughly equivalent code for running NCBI BLAST+ through its native interface and wrapped by ShortCut.

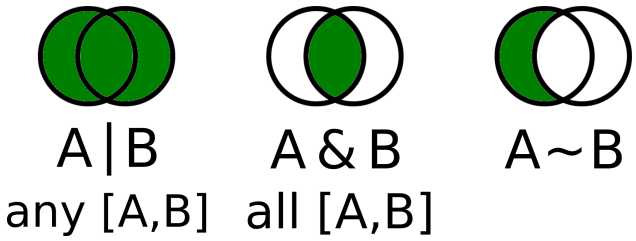


Figure 3.3: Set operations can be used to represent anything that could be drawn as a Venn Diagram. In ShortCut they operate on lists of any other type, and come as either binary operators (top row) that take two lists and produce a third, or as regular functions (bottom) that operate on a list of lists.

3.2.7 Prevent simple mistakes using types

ShortCut tags each expression in a script with a filetype: string, number, FASTA amino acid, list of FASTA amino acid, etc. Each function has rules for the types it will accept, and will prevent the user from accidentally passing it the wrong ones. For example in `my-first-blast.cut` (Listing 3), it would cause an error to use one of the `load_gbk` functions directly in `blastp_rbh`, because it only accepts FASTA amino acid files. It would also cause an error to pass it a FASTA nucleic acid file (`blastn_rbh` or `tblastn` would be appropriate for that instead), if the arguments in the wrong order (e-value cutoff last instead of first), or if there is a typo in one of the variable names. Without typechecking, these sorts of trivial mistakes will almost always derail a long script at some point.

3.2.8 Simplified format conversions

A few standard formats—mainly FASTA files and BLAST hit tables—can be used by multiple programs included in Shortcut, but otherwise it is not possible to pass the output of one to another directly. Bioinformaticians

would normally inspect the data from each step in their pipelines and write scripts specifically to put each in a format that can be understood by the next. ShortCut simplifies that process, first by putting the output of each step in a single standard format when possible, and second by providing conversion functions and using types to prevent them being used in invalid ways.

An example of the first strategy is that all the sequence searching programs are configured to return BLAST hit tables, or are wrapped by extra scripts to convert their output to that format.

An example of the second is that each ortholog-finding program is given its own result type (see `ofr`, `spr`, and `gcr` in Table B.1). To compare their results, a list of orthogroups needs to be extracted first with the `orthogroups` function. That could be done automatically of course, but a separate step leaves open the possibility of using their results in other ways too, which might not be compatible. For example a `species_tree` function could be written that would extract a consensus species tree from Orthofinder results, but would not work on GreenCut results. Similarly, GreenCut results could be separated into lists of orthologs (reciprocal best hits) and paralogs (the rest), but Orthofinder makes no such distinction.

3.2.9 Compare alternative methods

Another major advantage in a scientific context is that ShortCut explicitly includes trying alternatives as part of the code. In most languages this is done simply by editing the code and re-running it each time. That works well when programming, but when conducting experiments it can lead to accidental p-hacking (trying various study designs until one gives the "correct" result), or at least to publishing only the final working version and forgetting to document the series of experiments that led to it.

As an example, consider the e-value cutoff in Listing 3 above. Would lowering it lead to more or fewer hits? The script could be edited and re-run to find out, but this is more explicit:

```
mgen = load_faa "data/Mycoplasma_genitalium_protein_refseq.faa"
syne = concat_faa [gbk_to_faa "cds" (load_gbk "data/SynPCC7942_chr.gbk"),
                   gbk_to_faa "cds" (load_gbk "data/SynPCC7942_pANL.gbk")]

# first try the same as before, but make cutoff a variable
cutoff = 1e-10
hits = blastp_rbh cutoff mgen syne

# now try it with a range of cutoffs
alternate_hit_tables = replace_each hits cutoff [1e-5, 1e-10, 1e-50, 1e-20, 1e-50]
result = length_each alternate_hit_tables
```

Listing 10: An explicit experiment in code: how does changing the e-value cutoff affect the number of reciprocal best hits? The `replace_each` line can be thought of as "repeatedly calculate hits, replacing cutoff with each of these alternate cutoffs".

ShortCut also standardizes the inputs of the ortholog-finding programs (OrthoFinder, SonicParanoid, and my re-implementation of the GreenCut algorithm) so they can each be run starting from precalculated all-vs-all BLAST hits, and provides functions (ending in `_ava`) to generate those all-vs-all hits with each sequence searching program.

```
mgen  = load_faa "data/Mycoplasma_genitalium_small.faa"
mycos = load_faa_each (load_list "data/100-mycoplasma-proteomes.txt")

# alternative sequence search algorithms
# note that you might want different cutoffs
blast_hits   = blastp_ava         1e-10 mycos
diamond_hits = diamond_blastp_ava 1e-10 mycos
mmseqs_hits  = mmseqs_search_ava  1e-10 mycos

# alternative ortholog-finding algorithms, all using the blast hits for simplicity
# note that unlike the other two, the greencut algorithm requires a reference species
of_groups = orthogroups (orthofinder_ava             blast_hits)
sp_groups = orthogroups (sonicparanoid_ava           blast_hits)
gc_groups = orthogroups (greencut2_families_ava mgen blast_hits)

# make a very conservative list of only the orthogroups that all
# 3 algorithms agree span at least 90% of the mycoplasma species
result = all [ortholog_in_min 0.9 of_groups mycos,
              ortholog_in_min 0.9 sp_groups mycos,
              ortholog_in_min 0.9 gc_groups mycos]
```

Listing 11: Separate sequence searches and ortholog finding with the all-vs-all hits type.

Decoupling the two makes it possible to compare them separately. One can try multiple ortholog finders with minimal extra compute time or compare sequence searches in an attempt to make the chosen ortholog finder faster or more sensitive, depending on the species involved. This is the first program that allows comparing them directly on research data this way, which will also make it easier to verify whether newly released programs give similar results to older ones.

3.2.10 Export plots, lists, tables

ShortCut also includes functions to make basic plots (Listing 12), as well as simple text files and "set membership tables", spreadsheets in which rows are typically sequence IDs and columns sets of results. It does not aim for publication quality plots, but a custom script could be written for that if needed (Section 3.3.4).

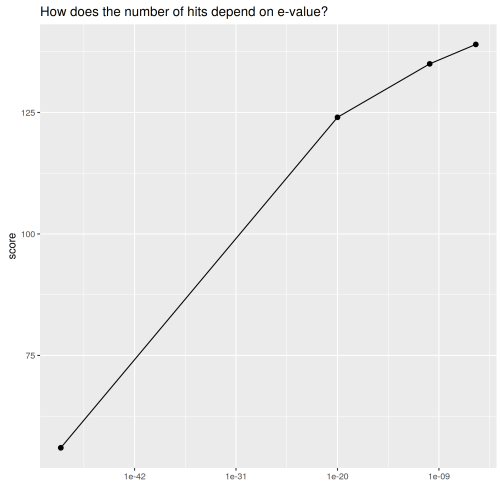
3.2.11 Automatic parallelization

ShortCut will run multiple functions in parallel as long as they do not depend on each other. For example in Listing 12 it will load and convert files in parallel, then do the alternate versions of the BLAST search in parallel, then finally make the plot.

3.2.12 Automatic updating of results

If one of the input files mentioned in the cut script changes, ShortCut will recalculate any variables that depend on it when re-run. This is especially useful for updating cuts as new genome annotations are released, but it could also be used to keep updated on local changes. For example there could be a directory of "species to base my orthogroups on" or a file listing genes to include or exclude at some step, and it would continuously update the results.

```
scores = score_repeats (length hits) cutoff [1e-5, 1e-10, 1e-50, 1e-20, 1e-50]
result = linegraph "How does the number of hits depend on e-value?" scores
```



Listing 12: Plot how the number of reciprocal best hits depends on the e-value cutoff. Assumes the rest of the BLASTp script is loaded from above.

```
cyano_proteomes = load_faa_each (glob_files "cyanobacteria/*_refseq.faa")
cyano_orthogroups = orthogroups (sonicparanoid cyano_proteomes)
result = cyano_orthogroups
```

Listing 13: A script designed to auto-update. If the list of files matching the `glob` pattern has changed since the last run, any variables that depend on it will be recalculated. Note that unless the algorithm is meant to handle changing files, explicitly listing the ones it does handle should be preferred for clarity.

3.2.13 Demo website

Though easier to install than comparable bioinformatics software, trying ShortCut still takes enough time that most potential users will not bother. Therefore, I built a web-based demo. The site (`shortcut.pmb.berkeley.edu`) is a work in progress but already includes the demo terminal, a tutorial, auto-generated function reference, and loadable examples. Code for the website is also available at `github.com/jefdaj/shortcut-demo`. It includes ShortCut itself, and can be reproducibly built with `nix-build` in the same way (Listing 8). This could be useful for labs or institutions to host a version of the demo for internal use, either for intellectual property reasons or to use a faster computer.

3.3 Methods

3.3.1 Interpreter design

This section explains the process of turning text descriptions in the DSL into a series of commands to run to actually calculate the result. The design leads naturally to most of the unique features of ShortCut relative to other languages.

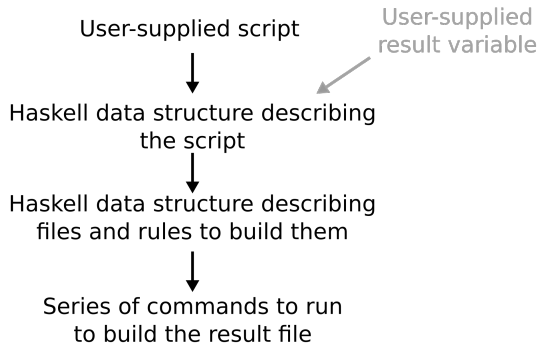


Figure 3.4: The overall process of interpreting a cut script. Each script includes a default `result` variable, which is overridden interactively by the user each time they type an expression.

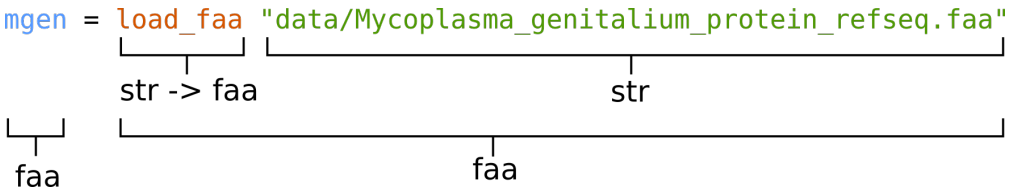


Figure 3.5: Typechecking an assignment statement. The `load_faa` function takes a string (`str`) and produces a FASTA amino acid (`faa`) file by loading it. Since it was passed a string as input, the types match. Finally, the same type is assigned to the textttmgen variable. When that variable is passed to another function later, it will have to be one that accepts an `faa` file.

Parsing

The first step is to transform human-readable text into a data structure annotated with the information that will be needed during the rest of the evaluation process. This is also the step where types are checked, and any malformed statements (typos in function names or missing parentheses, for example) are rejected.

A script is a list of assignment statements (Figure 3.6), and the entire program state consists of a script, stored command line arguments, and sets of the files and sequence IDs currently loaded. Whereas traditional imperative (step-by-step, rather than dependency-based) programming languages maintain a large and complex state that depends on the sequence of user actions taken, ShortCut only maintains the script itself and derives everything else from that as needed. This ensures that the script can always be written to a file and used to reproduce the current state later (Section 3.2.3).

Compilation

Once the input text has been validated and put in a more useful internal format, ShortCut matches each part of the code to a destination file, and decides what commands would need to be run to generate that file. Note that these commands will never actually be run unless the user asks to produce the file, or another file that depends on it.

One of the major goals of ShortCut is to ensure that two equivalent expressions map to the same temporary file ("tmpfile"). This is important both to reduce unnecessary computation and to ensure that set operations

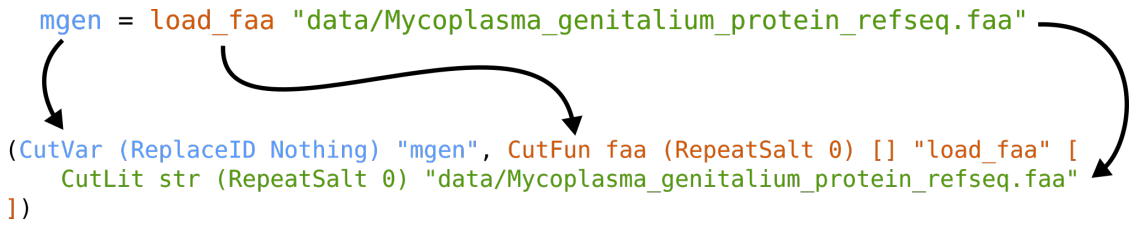


Figure 3.6: Parsing text into a useful data structure. ShortCut constructs an internal description with information like the function or variable name and the assigned type, validating each part of the text as it goes. This function call does not depend on any earlier variables, but if it did the empty list here (`[]`) would hold references to them. `ReplaceID` and `RepeatSalt` are only used for the replace and repeat operations respectively.

```
blastp_rbh 1e-5 species1 species2
       ↓
reciprocal_best (blastp     1e-5 species1 species2)
                (blastp_rev 1e-5 species1 species2)
       ↓
reciprocal_best (blastp 1e-5 species1 species2)
                (blastp 1e-5 species2 species1)
       ↓
reciprocal_best (blastp_db 1e-5 species1 (makeblastdb_prot species2))
                (blastp_db 1e-5 species2 (makeblastdb_prot species1))
```

Figure 3.7: Expansion of function calls. Some functions can be easily expressed in terms of other, simpler intermediates. ShortCut expands them during the compilation step. The internal data structure saved (see Figure 3.6) contains the first (top) version, so upon saving it reacreates the original code. The compiled file-building rules are based on the last version.

(Section 3.2.6) do not include the same set element twice under different filenames. Therefore each expression is assigned one canonical path, and may also have alternative paths that link to that.

```
mgen_genbank = load_gbk "data/Mycoplasma_genitalium_M2321.gbk"
mgen_fasta = gbk_to_faa "cds" mgen_genbank
result = mgen_fasta
```

```
/home/jefdaj/.shortcut
├── cache
│   ├── lines
│   │   ├── f094bac04c.txt
│   │   └── fcfb7a47a6.txt
│   ├── load
│   │   └── 28ce925871.gbk -> /home/jefdaj/shortcut/data/Mycoplasma_genitalium_M2321.gbk
│   └── seqio
├── exprs
│   ├── gbk_to_faa
│   │   └── 262cf7e4e4_cdab12f059_0.faa
│   ├── load_gbk
│   │   └── 15e3d91521_0.gbk -> ../../cache/load/28ce925871.gbk
│   └── str
│       ├── 90811d06ee_0.str -> ../../cache/lines/f094bac04c.txt
│       └── b4da62b027_0.str -> ../../cache/lines/fcfb7a47a6.txt
├── history.txt
├── profile.html
└── vars
    ├── mgen_fasta.faa -> ../exprs/gbk_to_faa/262cf7e4e4_cdab12f059_0.faa
    ├── mgen_genbank.gbk -> ../exprs/load_gbk/15e3d91521_0.gbk
    └── result -> ../vars/mgen_fasta.faa
```

Listing 14: Mapping expressions to tmpfiles. Variables are mapped to human-readable filenames supplied by the user and expressions are mapped to paths determined by hashing each component of their data structures (see Figure 3.6). Expressions are further de-duplicated by pointing symbolic links to module-specific cache directories to ensure that set operations do not count them twice.

This complicated tree of tmpfiles is also well-suited to moving data around, as discussed in Section 3.4.1. In Listing 14, notice that only the `cache/load` directory contains any references to files outside the temporary directory, and those references are hashed to depend on the file content rather than its name. Therefore the temporary files could be copied to another directory or another computer, and Shortcut would still match everything up and resume where it left off. Only variables depending on different input data in the new location would be recalculated.

Evaluation

The last step is to pick a file that needs building (the result variable), build all of its dependency files first, then run the commands that the compiled rules say will generate it. To build the dependencies, their dependencies are built first and then their commands are run, and so on. It naturally leads to calculating only what needs to be calculated for the current result variable, to building things in parallel when possible, and to re-using any files that have been built already.

3.3.2 Repetition

Replace

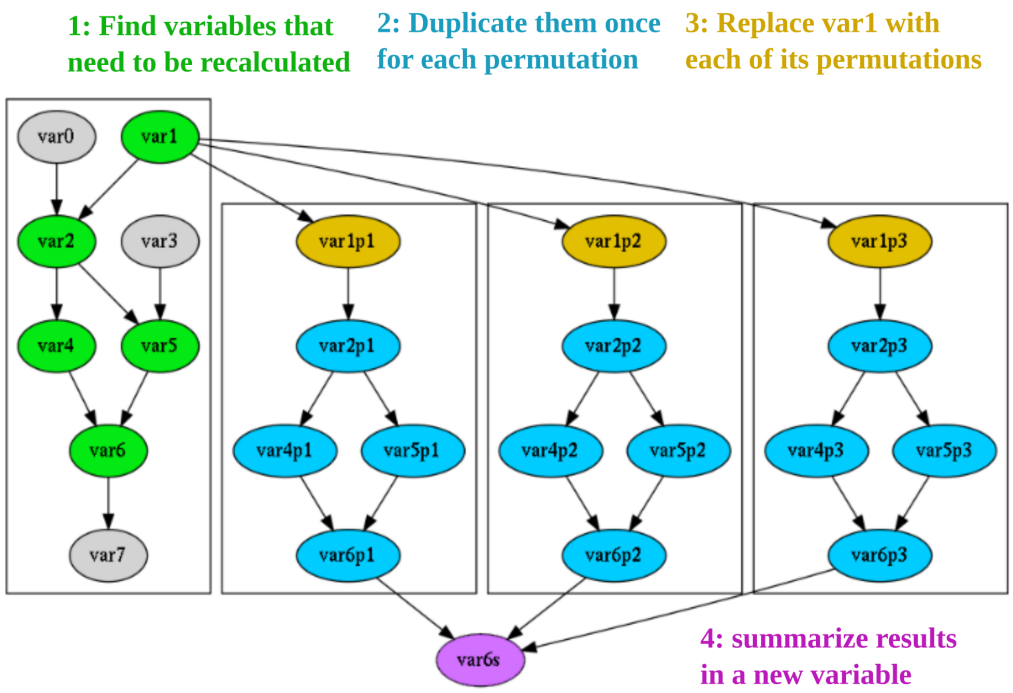
Since scripts are stateless data (all actions are explicit in the script itself), they can be edited during compilation and remain valid. ShortCut uses this property to implement the special `replace` and `replace_each` functions. They create copies of the script up to that point, edit the duplicate(s) by replacing one variable with another, and concatenate them. It sounds complicated, but in practice the code is simple to use. One common pattern is to figure out the search interactively on a small amount of data, then scale up by repeating it with more (Listing 15).

```
mgen = load_faa "data/Mycoplasma_genitalium_small.faa"
maga = load_faa "data/Mycoplasma_agalactiae_small.faa"
mycoplasma = load_faa_each (load_list "data/100-mycoplasma-proteomes.txt")

# make proteomes to use a variable so it can be replaced later,
# and do a small test run to make sure the code runs properly
proteomes_to_use = [maga]
small = diamond_blastp_each 1.0e-50 mgen proteomes_to_use

# try scaling it up: first to 5 proteomes, then all the rest
bigger  = replace small proteomes_to_use (sample 5 mycoplasma)
biggest = replace small proteomes_to_use mycoplasma
```

Listing 15: Test code with small data, then scale up. Note that no extra BLAST searches are being done by adding the `small` and `bigger` parts relative to running `biggest` only, because ShortCut reuses the ones that have already been done each time.

Permute, Replace, Summarize

Rather than trying different versions of a variable one at a time, it is often helpful to try a whole list of them at once and summarize the results. The "permute" and "summarize" steps take different forms depending on the purpose. For example the permutations could be a list of e-value cutoffs to try or a list of alternative reference genomes to start the search from. The summary could be to print the results, plot them as in Listing 12, or keep just the minimum or maximum.




```
var6s = replace_each var6 var1 [var1p1, var1p2, var1p3]
```

Figure 3.8: Visualizing the PRS pattern. You have the program on the left and want to know, "What happens to `var6` if I change `var1`?". The `replace_each` function recalculates `var6` starting from 3 alternate versions of `var1`. This example only contains the "permute" and "replace" steps; "summarize" would be applied to `var6s` afterward. Omitted for simplicity: each `var2` permutation also depends on `var0` and each `var5` permutation on `var3`.

```
# blast S. elongatus genes against Synechocystis with a standard cutoff
cutoff = 1e-10
hits = extract_queries (blastp pcc7942 pcc6803 cutoff)

# re-run it with a range of cutoffs and report the number of hits for each
cutoffs = [1e-5, 1e-10, 1e-20, 1e-50, 1e-100, 0]
lengths = repeat_each (length hits) cutoff cutoffs
```

Figure 3.9: A more practical example of the PRS pattern, which tests how the number of BLAST hits depends on the e-value cutoff.

Force actual repetition

By default, ShortCut deduplicates (caches) all function calls. Most of the time this is a good idea. There are at least two cases when it interferes though: when measuring runtime of the code, and when assessing determinism—do the results vary when the code is run more than once?

```
shortcut —▶ sample 10 (range_integers 1 100)
[36, 10, 43, 88, 87, 42, 3, 80, 72, 51]

shortcut —▶ sample 10 (range_integers 1 100)
[36, 10, 43, 88, 87, 42, 3, 80, 72, 51]

shortcut —▶ nums = range_integers 1 100
shortcut —▶ nums10 = sample 10 nums
shortcut —▶ repeat nums10 nums 2
[[36, 10, 43, 88, 87, 42, 3, 80, 72, 51], [98, 55, 72, 45, 59, 85, 58, 12, 34, 79]]
```

Listing 16: Repeat is required for random sampling.

The most common case that will probably trip users up is random samples. Most people would expect the code in Listing 16 to take two different random samples, but they refer to the same expression which is only evaluated once. To get around that the special function `repeat` alters the `RepeatSalt` of each of its arguments (See Figure 3.6), causing them to be compiled to different files and evaluated separately.

3.3.3 Hiding irrelevant details

Not everything can be fixed by an elegant design; the individual programs involved have many idiosyncracies that need to be worked around individually. I have put significant effort into working around as many of them as possible so that everything runs reliably, and the user only has to specify things that depend on the kind of search they want to run.

Installation requirements

Installing dependencies is almost certainly the biggest barrier preventing biologists from trying new bioinformatic methods. Most people would try running a command to see if it works, but would not spend time reading the manual and running various install commands to get to the point of first trying it.

ShortCut solves this by using a package manager called Nix (`nixos.org/nix`) to install the exact dependencies needed, separate from anything else the user might have on their system. It also keeps them separate from each other, which is necessary because they are sometimes incompatible. I wrote a build script ("Nix package") for each program that did not have one already (See Listing 1 for an example).

Nix also helps solve the related problem of how to ensure ShortCut itself will keep working in the future. An archive of all software needed to run it (besides Nix) can be exported to a file and re-imported from another computer later (Listing 17). The archive includes everything—several incompatible versions of Python, parts of Linux, Perl, R, all the way up to the user-facing programs. The resulting file is several gigabytes, but it is self-contained enough to ensure that the code will not break due to updates.

Runtime requirements

Once installed in an idiosyncratic way, many programs expect to be run in an idiosyncratic way too. For some functions, generating a file just means running the corresponding script. But for others there are more steps: create a directory, make links to all the input files in that directory, run the script, sanitize the output file, etc. The "build and run rules" paradigm is flexible enough to handle whatever steps are needed. For example, SonicParanoid requires the user to write a custom configuration file before running the program, explicitly listing the files

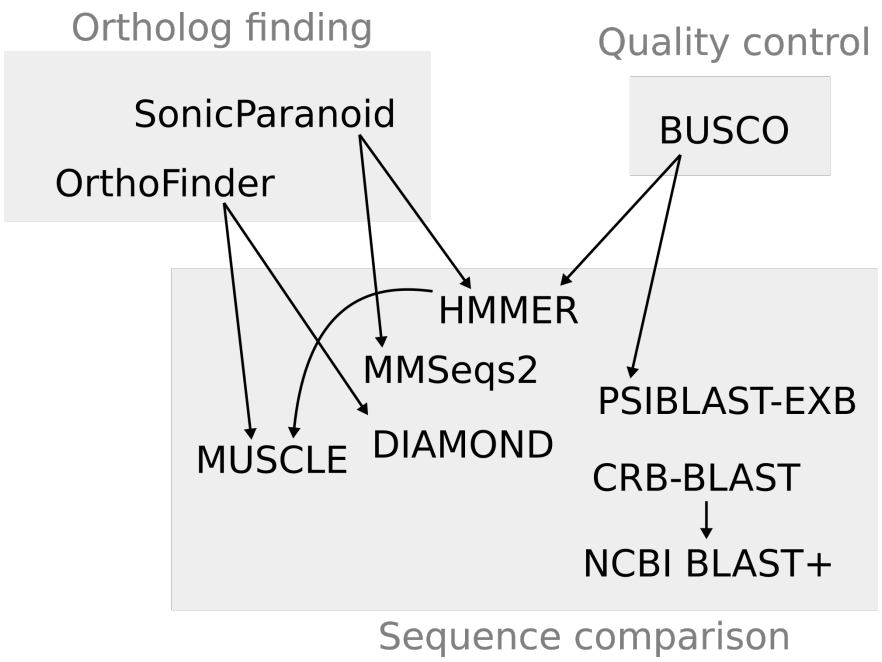


Figure 3.10: Dependencies between reusable Nix packages included in ShortCut. Not shown: custom scripts, small programs for loading, converting, or plotting, programming languages, built-in features like set operations.

```
# export now
nix-store --export $(nix-store -qR $(type -p shortcut)) \
  > 2019-08-16_shortcut-long-term-archive.nix

# restore later
nix-store --import < 2019-08-16_shortcut-long-term-archive.nix
```

Listing 17: Exporting all dependencies to a file and re-importing them later.

and settings that will be used. ShortCut generates one from a template each time the function is called. Another example is that BLAST often fails to find its database file unless run from the same directory and given a name without the file extension. Individually these are minor annoyances, but they tend to build up enough friction that experimenting with a different way of designing a cut could take hours rather than a few minutes after going through them manually.

Formatting requirements

There have also been bugs caused by differing expectations between programs. For example, some cannot handle certain characters in FASTA sequence IDs, and will mysteriously fail if given the wrong ones. To get around it, when ShortCut "loads" a FASTA file it removes the IDs and replaces them with short standardized names including the hash of the original (Figure 18). It uses the meaningless but reliable hashed names internally, then puts back the originals when writing final output for the user.

```
seqid_0169657d2a        WP_041593677.1 adhesin [Mycoplasma genitalium]
seqid_01e28e6809        WP_010869388.1 hypothetical protein [Mycoplasma genitalium]
seqid_01f33cdd44        WP_010869346.1 ribonuclease Y [Mycoplasma genitalium]
...
```

Listing 18: Sanitizing FASTA sequence IDs. ShortCut generates the IDs on the left for internal use, then puts the originals back at the end.

3.3.4 Adapting to custom code

Although ShortCut can do many things, the reality of science is that a given study is likely to need something more advanced, and researchers will not necessarily know what they need beforehand. Therefore it is important to include "escape hatches" to inspect the intermediate results themselves or add custom code.

Inspecting results outside ShortCut

ShortCut only categorizes the input and output files from the scripts it runs. They are out of the way, but can still be found if needed in the `.shortcut/cache` directory (Listing 19).

Including custom scripts

The `script` function can be used to make a ShortCut function out of a separate script. This would be useful if to alter only one step in the script, for example by removing certain hits from a hit table, or to tack on an entire custom analysis. Figure 3.11 and Listing 20 give an example of the second option. I expect another common use will be to take the final results of the script and plot them with a custom R script to make publication-quality figures.

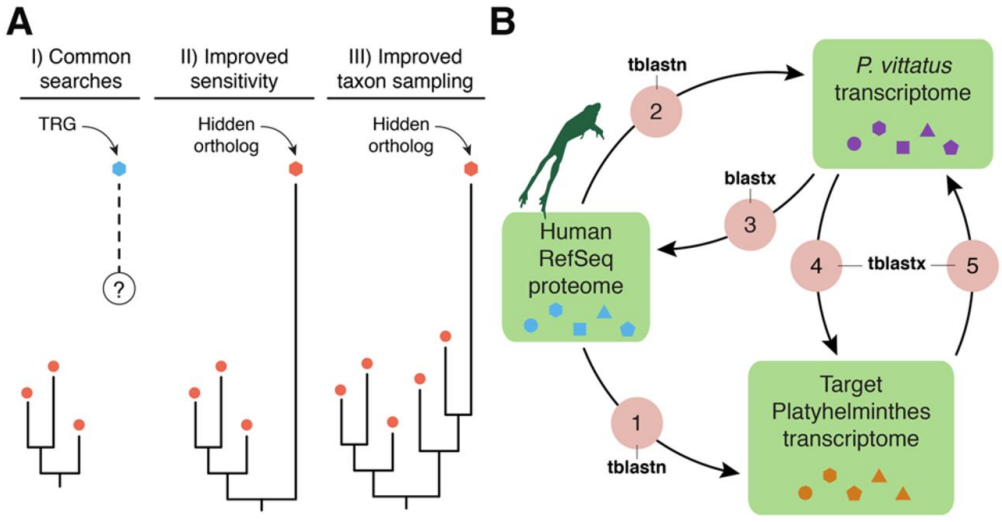


Figure 3.11: Overview of the LeapFrog pipeline. Reproduced with permission from[33], and available at `github.com/josephryan/leapfrog`. A frog transcriptome is used as a "bridge" to find orthologs in two species (Human and a flatworm) by transitive homology that are too distant to be discovered by ortholog-finding programs otherwise.

```
shortcut —► :load tests/scripts/orthofinder_basic.cut
smallbacteria = load_faa_each ["data/Mycoplasma_agalactiae_small.faa",
                               "data/Mycoplasma_genitalium_small.faa"]
ofres = orthofinder smallbacteria
result = ofres

orthofinder_basic.cut —► ofres
Number of species       2
Number of genes 374
Number of genes in orthogroups  75
Number of unassigned genes      299
Percentage of genes in orthogroups      20.1
Percentage of unassigned genes  79.9
Number of orthogroups   34
Number of species-specific orthogroups  0
Number of genes in species-specific orthogroups 0
Percentage of genes in species-specific orthogroups     0.0
Mean orthogroup size    2.2
Median orthogroup size  2.0
G50 (assigned genes)    2
G50 (all genes) 1
O50 (assigned genes)    16
O50 (all genes) 146
Number of orthogroups with all species present  34

/home/jefdaj/.shortcut/cache/orthofinder/1553eeea0a/OrthoFinder/Results_
├── Comparative_Genomics_Statistics
│   ├── Duplications_per_Orthogroup.tsv
│   ├── Duplications_per_Species_Tree_Node.tsv
│   ├── Orthogroups_SpeciesOverlaps.tsv
│   ├── OrthologuesStats_many-to-many.tsv
│   ├── OrthologuesStats_many-to-one.tsv
│   ├── OrthologuesStats_one-to-many.tsv
│   ├── OrthologuesStats_one-to-one.tsv
│   ├── OrthologuesStats_Totals.tsv
│   ├── Statistics_Overall.tsv
│   └── Statistics_PerSpecies.tsv
├── Gene_Duplication_Events
│   ├── Duplications.tsv
│   └── SpeciesTree_Gene_Duplications_0.5_Support.txt
├── Gene_Trees
│   ├── OG0000000_tree.txt
│   └── OG0000001_tree.txt
...
```

Listing 19: Accessing cached tmpfiles. Orthofinder calculates many results besides the orthogroups used by ShortCut. They can be found in the cache directory.

```
reference = load_faa "Human_refseq_proteome.faa"         # 1
bridge    = load_fna "P_vittatus_transcriptome.fna"      # 2
target    = load_fna "Platyhelminthes_transcriptome.fna" # 3

# First version
e = 1e-5
hits1v2 = tblastn e reference bridge
hits2v1 =  blastx e bridge reference
hits1v3 = tblastn e reference target
# note that there's no 3v1 (target -> reference)
hits2v3 = tblastx e bridge target
hits3v2 = tblastx e target bridge
# the "bht" here tells shortcut that the script returns a BLAST hits table
leap1 = script "leapfrog.sh" "bht" e hits1v2 hits2v1 hits1v3 hits2v3 hits3v2

# Second version
leap2 = leapfrog e reference bridge target
```

```
#!/usr/bin/env bash
outpath="$1"; shift
leapfrog --eval="$(cat $1)" \
         --1v2="$2" --2v1="$3" --1v3="$4" --2v3="$5" --3v2="$6" > "$outpath"
```

Listing 20: Including a custom script. **Top:** Two ways of integrating the LeapFrog pipeline into a cut script. The first (straighforward) version uses only the code shown and is appropriate for a single study. The second uses a ShortCut "module" (Haskell code), and allows for simpler re-use by others. **Bottom:** Bash script used in combination with either option to wrap the `leapfrog` command.

3.3.5 Adapting to custom environments

Research code also needs to run on unusual computers with institution-specific requirements. ShortCut has been used developed for use on the Berkeley High-Performance Compute cluster[9], and should also be adaptable to most other environments. The main tool for adjusting ShortCut to run on a computer with unusual requirements is the wrapper script option (`--wrapper`). Rather than running commands directly as usual, ShortCut will send everything it would run through the wrapper script instead, which can be edited as needed to include any system-specific quirks. An example is included for running on the Berkeley cluster. Rather than running commands immediately it queues each one to run on a separate compute node when available and waits for it to finish. The Berkeley script also needs to work around the inability to install software on the cluster directly. ShortCut and all its dependencies are included in an isolated virtual machine instead, and the script calls that. The process is fairly convoluted, and shows that the wrapper script idea is flexible enough to be used in most academic or industry environments.

3.4 Future directions

3.4.1 Sharing and re-use of code and data between users

ShortCut maps the result of every function call to a unique filename (See Listing 14) independent of both the machine used and what the user names their variables—only the actual data is considered. Therefore, the files

could be shared between users. Before computing a result, ShortCut could check whether the corresponding file already exists on one or more servers, and if so fetch it rather than recalculating it. In fact, Shake[35] (the library used by ShortCut internally to decide which files to build in which order) already includes an option to do that. Only a few modifications would be needed to enable the ShortCut demo server (Section 3.2.13) to cache and distribute files requested by users. It could be used to dramatically speed up common BLAST searches by precalculating them. Furthermore, a similar mechanism is used by Nix to cache and distribute software dependencies. It could also be enabled on the server, speeding up initial installation of ShortCut.

3.4.2 Mark deterministic functions to speed up repeats

Some functions, for example `concat_faa` and `reciprocal_best`, are completely deterministic. ShortCut should map them to the same tmpfile no matter their repeat salt (Section 3.6). The resulting speed-up may or may not be significant depending on the overall cut script design; in many cases nondeterministic sequence searches would still dominate.

3.4.3 Automatic runtime estimation and comparison

Removing repeats would, however, make runtime estimation more difficult. The simplest solution is probably to have a "timing" mode in which all function deduplication is turned off.

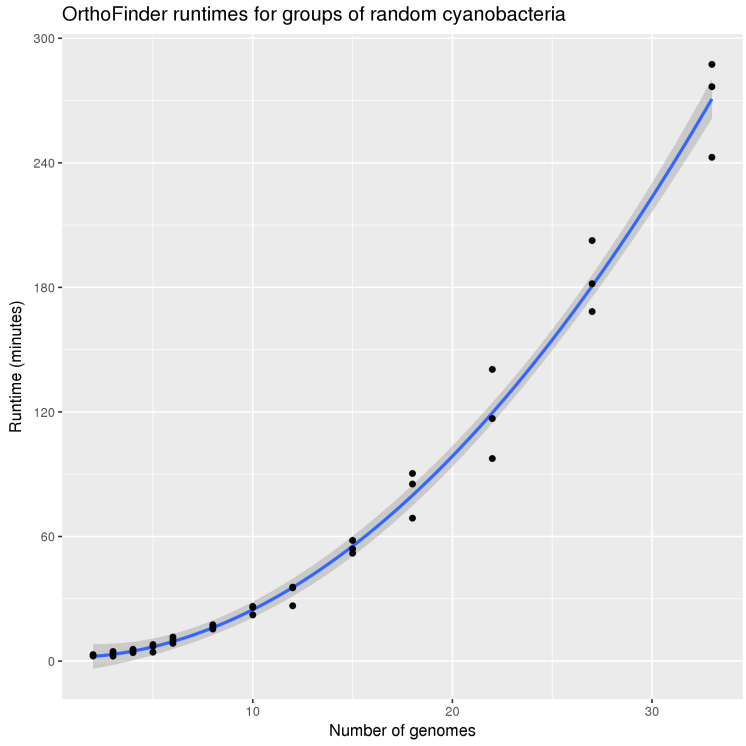


Figure 3.12: OrthoFinder runtimes plotted for random lists of increasing size sampled from a 172 cyanobacterial genomes (using their predicted proteomes).

Figure 3.12 shows an experiment to determine how OrthoFinder scales with increasing numbers of small proteomes. ShortCut could automatically generate figures like this using the `sample`, `score_replaced`, and plot

functions. They would work with more complicated series of steps as well, and be especially helpful for determining the behavior of multi-step algorithms where the relationship may not follow any obvious curve.

3.5 Conclusion

Why go to the trouble of making a (relatively) user-friendly scripting language when I could have found all the bugs in my configuration files and published the PSIIcut much more quickly instead? The main reason is that I had to go through much of the same work to be confident in my own searches. The first few did not produce trustworthy candidate gene lists. They were missing obvious genes that should have been included, or they had a much smaller number of hits than expected. By the time I simplified the descriptions far enough and put in enough error checks to be confident, it was most of a language. I hope that by going a little further than strictly necessary—creating the website, writing documentation, and incorporating a large number of possible search functions (see Table B.2 for the complete list)—I can help other people be confident in theirs too.

The system developed here helps codify a particular type of common phylogenomic search, sometimes called a "cut". It simplifies the common task of searching for homologs of across multiple species and comparing the results. It is simple enough that most biologists with the inclination should be able to install and use it, and flexible enough to answer a variety of research questions that are hard to address using existing tools, such as the NCBI website. Perhaps most importantly, it facilitates reuse of methods between studies by introducing a language that is human-readable for inclusion in supplemental materials, yet able to reliably reproduce the same results years later on different computers.

The Nix packages developed here can also be used separately from ShortCut. I will be uploading them to the Nix packages repository at `github.com/nixos/nixpkgs`, and to my knowledge they will represent the easiest way to install most of these tools—especially for the ones containing code from more than one language, which usually require keeping multiple language-specific release schedules in sync.

The GreenCut update will be of independent interest regardless of the code used to make it of course. But I hope that including the code will also help make future updates easier and more frequent, and encourage incremental improvement as new genome annotations and software tools are released.

Chapter 4

Conclusion

Clearly I have more aptitude for computers than for molecular biology. Despite that, I think the most exciting direction to take the TnSeq experiments from here is to make knockouts and discover the actual functions of some of the genes identified so far, rather than only speculating. Is Synpcc7942_2355 actually a phasin, or does it perform some unexpected role in stabilizing D1 repair? Or something else? Do the putative chlorophyll insertion pathway genes act in a complex? Does cofitness with PsbW imply physical interactions between many of the proteins? Do knockouts of positive-fitness genes increase growth under industrially relevant bioreactor conditions?

TnSeq libraries themselves are also a very promising tool. They have been successfully scaled to hundreds of simultaneous experiments with heterotrophic bacteria, and with only small modifications the same protocols could be adjusted to map out detailed fitness landscapes showing the interactions between, for example, bicarbonate concentration and light intensity for all nonessential genes. The fluctuating light experiments in Chapter 2 suggest that photosynthetic phenotypes can readily be identified under nonstandard conditions.

Finally, having spent so long developing ShortCut as a tool for phylogenomic searches, I think it will be exciting to create a variety of "cuts" and demonstrate that they improve on previous efforts: the GreenCut update (nearly done), an expanded PSIIcut taking into account the newly discovered melainabacteria as a non-photosynthetic outgroup to cyanobacteria, as well as several recently discovered *A. thalassa* strains; cuts for diatoms, red algae, and other lineages. Cuts focused on particular structures like bacterial microcompartments. I am also interested in encouraging other people to join in and/or make their own cuts; the point is to enable "domain experts" to investigate the species they are experts in, rather than for a programmer to try to do it for them. Most exciting will be if some of them can be convinced to build on each others' work.

References

[1] Abu-Jamous, B. & Kelly, S. (2018). Clust: Automatic extraction of optimal co-expressed gene clusters from gene expression data. *Genome Biology*, 19(1), 172. https://doi.org/10.1186/s13059-018-1536-8.

[2] Allahverdiyeva, Y., Mustila, H., Ermakova, M., Bersanini, L., Richaud, P., Ajlani, G., Battchikova, N., Cournac, L., & Aro, E.-M. (2013). Flavodiiron proteins Flv1 and Flv3 enable cyanobacterial growth and photosynthesis under fluctuating light. *Proceedings of the National Academy of Sciences of the United States of America*, 110(10), 4111–4116.

[3] Almeida-E-Silva, D. C. & Vêncio, R. Z. (2015). Sifter-T: A scalable framework for phylogenomic probabilistic protein domain functional annotation. *BMC Bioinformatics*, 16(8), A4. https://doi.org/10.1186/1471-2105-16-S8-A4.

[4] Álvarez-Escribano, I., Vioque, A., & Muro-Pastor, A. M. (2018). NsrR1, a Nitrogen Stress-Repressed sRNA, Contributes to the Regulation of nblA in Nostoc sp. PCC 7120. *Frontiers in Microbiology*, 9. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6166021/.

[5] Anders, S. & Huber, W. (2010). Differential expression analysis for sequence count data. *Genome Biology*, 11(10), R106. http://genomebiology.com/2010/11/10/R106.

[6] Andersson, B., Shen, C., Cantrell, M., Dandy, D. S., & Peers, G. (2019). The Fluctuating Cell-Specific Light Environment and its Effects on Cyanobacterial Physiology. *Plant Physiology*.

[7] Ansari, S. & Fatma, T. (2016). Cyanobacterial Polyhydroxybutyrate (PHB): Screening, Optimization and Characterization. *PLOS ONE*, 11(6), e0158168. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0158168.

[8] Aravind, L. & Ponting, C. P. (1997). The GAF domain: An evolutionary link between diverse phototransducing proteins. *Trends in Biochemical Sciences*, 22(12), 458–459. http://www.sciencedirect.com/science/article/pii/S0968000497011481.

[9] Berkeley, U. (2019). CGRL (Vector/Rosalind) User Guide.

[10] Bi, E. & Lutkenhaus, J. (1993). Cell division inhibitors SulA and MinCD prevent formation of the FtsZ ring. *Journal of Bacteriology*, 175(4), 1118–1125. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC193028/.

[11] Calderon, R. H., García-Cerdán, J. G., Malnoë, A., Cook, R., Russell, J. J., Gaw, C., Dent, R. M., de Vitry, C., & Niyogi, K. K. (2013). A Conserved Rubredoxin Is Necessary for Photosystem II Accumulation in Diverse Oxygenic Photoautotrophs. *Journal of Biological Chemistry*, 288(37), 26688–26696. http://www.jbc.org/content/288/37/26688.

[12] Clerico, E. M., Ditty, J. L., & Golden, S. S. (2007). Specialized techniques for site-directed mutagenesis in cyanobacteria. In *Circadian Rhythms*, volume 362 of *Methods in Molecular Biology* (pp. 155–171). Springer. http://link.springer.com/10.1007/978-1-59745-257-1_11.

[13] Cohen, A., Sendersky, E., Carmeli, S., & Schwarz, R. (2014). Collapsing Aged Culture of the Cyanobacterium Synechococcus elongatus Produces Compound(s) Toxic to Photosynthetic Organisms. *PLoS ONE*, 9(6), e100747. http://dx.plos.org/10.1371/journal.pone.0100747.

[14] Dostra, E. (2019). Nix: The Purely Functional Package Manager. https://nixos.org/nix/.

[15] Durán, R. V., Hervás, M., De La Rosa, M. A., & Navarro, J. A. (2004). The efficient functioning of photosynthesis and respiration in Synechocystis sp. PCC 6803 strictly requires the presence of either cytochrome c6 or plastocyanin. *The Journal of Biological Chemistry*, 279(8), 7229–7233.

[16] Escudero, L., Mariscal, V., & Flores, E. (2015). Functional Dependence between Septal Protein SepJ from Anabaena sp. Strain PCC 7120 and an Amino Acid ABC-Type Uptake Transporter. *Journal of Bacteriology*, 197(16), 2721–2730. https://jb.asm.org/content/197/16/2721.

[17] Gerdes, S. Y., Kurnasov, O. V., Shatalin, K., Polanuyer, B., Sloutsky, R., Vonstein, V., Overbeek, R., & Osterman, A. L. (2006). Comparative Genomics of NAD Biosynthesis in Cyanobacteria. *Journal of Bacteriology*, 188(8), 3012–3023. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1446974/.

[18] Graham, P. J., Nguyen, B., Burdyny, T., & Sinton, D. (2017). A penalty on photosynthetic growth in fluctuating light. *Scientific Reports*, 7(1), 12513. https://www.nature.com/articles/s41598-017-12923-1.

[19] Hagino, K., Onuma, R., Kawachi, M., & Horiguchi, T. (2013). Discovery of an endosymbiotic nitrogen-fixing cyanobacterium UCYN-A in Braarudosphaera bigelowii (Prymnesiophyceae). *PloS One*, 8(12), e81749.

[20] Hannon, M., Gimpel, J., Tran, M., Rasala, B., & Mayfield, S. (2010). Biofuels from algae: Challenges and potential. *Biofuels*, 1(5), 763. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3152439/.

[21] Hauf, W., Watzer, B., Roos, N., Klotz, A., & Forchhammer, K. (2015). Photoautotrophic Polyhydroxybutyrate Granule Formation Is Regulated by Cyanobacterial Phasin PhaP in Synechocystis sp. Strain PCC 6803. *Applied and Environmental Microbiology*, 81(13), 4411–4422. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4475881/.

[22] Henkes, S., Sonnewald, U., Badur, R., Flachmann, R., & Stitt, M. (2001). A small decrease of plastid transketolase activity in antisense tobacco transformants has dramatic effects on photosynthesis and phenylpropanoid metabolism. *The Plant Cell*, 13(3), 535–551.

[23] Hood, R. D., Higgins, S. A., Flamholz, A., Nichols, R. J., & Savage, D. F. (2016). The stringent response regulates adaptation to darkness in the cyanobacterium Synechococcus elongatus. *Proceedings of the National Academy of Sciences*, 113(33), E4867–E4876. http://www.pnas.org/content/113/33/E4867.

[24] Ji, D.-L., Lin, H., Chi, W., & Zhang, L.-X. (2012). CpLEPA Is Critical for Chloroplast Protein Synthesis Under Suboptimal Conditions in Arabidopsis thaliana. *PLoS ONE*, 7(11). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3499520/.

[25] Johnson, J. D. (2019a). Jefdaj / tnseq7942-hl — Bitbucket.

[26] Johnson, J. D. (2019b). Nix Packages collection. https://github.com/jefdaj/nixpkgs.

[27] Karpowicz, S. J., Prochnik, S. E., Grossman, A. R., & Merchant, S. S. (2011). The GreenCut2 Resource, a Phylogenomically Derived Inventory of Proteins Specific to the Plant Lineage. *Journal of Biological Chemistry*, 286(24), 21427–21439. http://www.jbc.org/cgi/doi/10.1074/jbc.M111.233734.

[28] Kehr, J.-C. & Dittmann, E. (2015). Biosynthesis and Function of Extracellular Glycans in Cyanobacteria. *Life*, 5(1), 164–180. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4390846/.

[29] Laudenbach, D. E., Herbert, S. K., McDowell, C., Fork, D. C., Grossman, A. R., & Straus, N. A. (1990). Cytochrome c-553 is not required for photosynthetic activity in the cyanobacterium Synechococcus. *The Plant Cell*, 2(9), 913–924.

[30] Liepman, A. H. & Olsen, L. J. (2003). Alanine Aminotransferase Homologs Catalyze the Glutamate:Glyoxylate Aminotransferase Reaction in Peroxisomes of Arabidopsis. *Plant Physiology*, 131(1), 215–227. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC166801/.

[31] López-Gomollón, S., Sevilla, E., Bes, M. T., Peleato, M. L., & Fillat, M. F. (2009). New insights into the role of Fur proteins: FurB (All2473) from Anabaena protects DNA and increases cell survival under oxidative stress. *Biochemical Journal*, 418(1), 201–207. http://www.biochemj.org/content/418/1/201.

[32] Marbouty, M., Saguez, C., Cassier-Chauvat, C., & Chauvat, F. (2009). Characterization of the FtsZ-Interacting Septal Proteins SepF and Ftn6 in the Spherical-Celled Cyanobacterium Synechocystis Strain PCC 6803. *Journal of Bacteriology*, 191(19), 6178–6185. https://jb.asm.org/content/191/19/6178.

[33] Martín-Durán, J. M., Ryan, J. F., Vellutini, B. C., Pang, K., & Hejnol, A. (2017). Increased taxon sampling reveals thousands of hidden orthologs in flatworms. *Genome Research*, 27(7), 1263–1272. http://genome.cshlp.org/content/27/7/1263.

[34] Merchant, S. S., Prochnik, S. E., Vallon, O., Harris, E. H., Karpowicz, S. J., Witman, G. B., Terry, A., Salamov, A., Fritz-Laylin, L. K., Marechal-Drouard, L., Marshall, W. F., Qu, L.-H., Nelson, D. R., Sanderfoot, A. A., Spalding, M. H., Kapitonov, V. V., Ren, Q., Ferris, P., Lindquist, E., Shapiro, H., Lucas, S. M., Grimwood, J., Schmutz, J., Cardol, P., Cerutti, H., Chanfreau, G., Chen, C.-L., Cognat, V., Croft, M. T., Dent, R., Dutcher, S., Fernandez, E., Fukuzawa, H., Gonzalez-Ballester, D., Gonzalez-Halphen, D., Hallmann, A., Hanikenne, M., Hippler, M., Inwood, W., Jabbari, K., Kalanon, M., Kuras, R., Lefebvre, P. A., Lemaire, S. D., Lobanov, A. V., Lohr, M., Manuell, A., Meier, I., Mets, L., Mittag, M., Mittelmeier, T., Moroney, J. V., Moseley, J., Napoli, C., Nedelcu, A. M., Niyogi, K., Novoselov, S. V., Paulsen, I. T., Pazour, G., Purton, S., Ral, J.-P., Riano-Pachon, D. M., Riekhof, W., Rymarquis, L., Schroda, M., Stern, D., Umen, J., Willows, R., Wilson, N., Zimmer, S. L., Allmer, J., Balk, J., Bisova, K., Chen, C.-J., Elias, M., Gendler, K., Hauser, C., Lamb, M. R., Ledford, H., Long, J. C., Minagawa, J., Page, M. D., Pan, J., Pootakham, W., Roje, S., Rose, A., Stahlberg, E., Terauchi, A. M., Yang, P., Ball,

S., Bowler, C., Dieckmann, C. L., Gladyshev, V. N., Green, P., Jorgensen, R., Mayfield, S., Mueller-Roeber, B., Rajamani, S., Sayre, R. T., Brokstein, P., Dubchak, I., Goodstein, D., Hornick, L., Huang, Y. W., Jhaveri, J., Luo, Y., Martinez, D., Ngau, W. C. A., Otillar, B., Poliakov, A., Porter, A., Szajkowski, L., Werner, G., Zhou, K., Grigoriev, I. V., Rokhsar, D. S., & Grossman, A. R. (2007). The Chlamydomonas Genome Reveals the Evolution of Key Animal and Plant Functions. *Science*, 318(5848), 245–250. http://www.sciencemag.org/cgi/doi/10.1126/science.1143609.

[35] Mitchell, N. (2019). Shake Build System.

[36] Montesinos, M. L., Herrero, A., & Flores, E. (1995). Amino acid transport systems required for diazotrophic growth in the cyanobacterium Anabaena sp. strain PCC 7120. *Journal of Bacteriology*, 177(11), 3150–3157.

[37] Ogawa, T. & Yoshimura, K. (2019). Modulation of the subcellular levels of redox cofactors by Nudix hydrolases in chloroplasts. *Environmental and Experimental Botany*, 161, 57–66. http://www.sciencedirect.com/science/article/pii/S0098847218312814.

[38] Ohnishi, N., Kashino, Y., Satoh, K., Ozawa, S.-I., & Takahashi, Y. (2007). Chloroplast-encoded polypeptide PsbT is involved in the repair of primary electron acceptor QA of photosystem II during photoinhibition in Chlamydomonas reinhardtii. *The Journal of Biological Chemistry*, 282(10), 7107–7115.

[39] Ohnishi, N. & Takahashi, Y. (2001). PsbT polypeptide is required for efficient repair of photodamaged photosystem II reaction center. *The Journal of Biological Chemistry*, 276(36), 33798–33804.

[40] Ohnishi, N. & Takahashi, Y. (2008 Oct-Dec). Chloroplast-encoded PsbT is required for efficient biogenesis of photosystem II complex in the green alga Chlamydomonas reinhardtii. *Photosynthesis Research*, 98(1-3), 315–322.

[41] Ollagnier-de Choudens, S., Loiseau, L., Sanakis, Y., Barras, F., & Fontecave, M. (2005). Quinolinate synthetase, an iron-sulfur enzyme in NAD biosynthesis. *FEBS letters*, 579(17), 3737–3743.

[42] Parnasa, R., Nagar, E., Sendersky, E., Reich, Z., Simkovsky, R., Golden, S., & Schwarz, R. (2016). Small secreted proteins enable biofilm development in the cyanobacterium Synechococcus elongatus. *Scientific Reports*, 6, 32209.

[43] Parnasa, R., Sendersky, E., Simkovsky, R., Waldman Ben-Asher, H., Golden, S. S., & Schwarz, R. (2019). A microcin processing peptidase-like protein of the cyanobacterium Synechococcus elongatus is essential for secretion of biofilm-promoting proteins. *Environmental Microbiology Reports*.

[44] Plöchinger, M., Schwenkert, S., von Sydow, L., Schröder, W. P., & Meurer, J. (2016). Functional Update of the Auxiliary Proteins PsbW, PsbY, HCF136, PsbN, TerC and ALB3 in Maintenance and Assembly of PSII. *Frontiers in Plant Science*, 7. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4823308/.

[45] Price, M. N., Zane, G. M., Kuehl, J. V., Melnyk, R. A., Wall, J. D., Deutschbauer, A. M., & Arkin, A. P. (2018). Filling gaps in bacterial amino acid biosynthesis pathways with high-throughput genetics. *PLOS Genetics*, 14(1), e1007147. http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1007147.

[46] Raffaelli, N., Lorenzi, T., Amici, A., Emanuelli, M., Ruggieri, S., & Magni, G. (1999). Synechocystis sp. slr0787 protein is a novel bifunctional enzyme endowed with both nicotinamide mononucleotide adenylyltransferase and 'Nudix' hydrolase activities. *FEBS Letters*, 444(2-3), 222–226. https://febs.onlinelibrary.wiley.com/doi/abs/10.1016/S0014-5793%2899%2900068-X.

[47] Rubin, B. E., Huynh, T. N., Welkie, D. G., Diamond, S., Simkovsky, R., Pierce, E. C., Taton, A., Lowe, L. C., Lee, J. J., Rifkin, S. A., Woodward, J. J., & Golden, S. S. (2018). High-throughput interaction screens illuminate the role of c-di-AMP in cyanobacterial nighttime survival. *PLOS Genetics*, 14(4), e1007301. http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1007301.

[48] Rubin, B. E., Wetmore, K. M., Price, M. N., Diamond, S., Shultzaberger, R. K., Lowe, L. C., Curtin, G., Arkin, A. P., Deutschbauer, A., & Golden, S. S. (2015). The essential gene set of a photosynthetic organism. *Proceedings of the National Academy of Sciences of the United States of America*.

[49] Sakata, S., Mizusawa, N., Kubota-Kawai, H., Sakurai, I., & Wada, H. (2013). Psb28 is involved in recovery of photosystem II at high temperature in Synechocystis sp. PCC 6803. *Biochimica Et Biophysica Acta*, 1827(1), 50–59.

[50] Schatz, D., Nagar, E., Sendersky, E., Parnasa, R., Zilberman, S., Carmeli, S., Mastai, Y., Shimoni, E., Klein, E., Yeger, O., Reich, Z., & Schwarz, R. (2013). Self-suppression of biofilm formation in the cyanobacterium Synechococcus elongatus. *Environmental Microbiology*, 15(6), 1786–1794.

[51] Seki, Y., Nitta, K., & Kaneko, Y. (2014). Observation of polyphosphate bodies and DNA during the cell division cycle of Synechococcus elongatus PCC 7942. *Plant Biology (Stuttgart, Germany)*, 16(1), 258–263.

[52] Shi, L.-X., Lorković, Z. J., Oelmüller, R., & Schröder, W. P. (2000). The Low Molecular Mass PsbW Protein Is Involved in the Stabilization of the Dimeric Photosystem II Complex in Arabidopsis thaliana. *Journal of Biological Chemistry*, 275(48), 37945–37950. http://www.jbc.org/content/275/48/37945.

[53] Shukla, M. K., Llansola-Portoles, M. J., Tichý, M., Pascal, A. A., Robert, B., & Sobotka, R. (2018). Binding of pigments to the cyanobacterial high-light-inducible protein HliC. *Photosynthesis Research*, 137(1), 29–39.

[54] Simkovsky, R., Daniels, E. F., Tang, K., Huynh, S. C., Golden, S. S., & Brahamsha, B. (2012). Impairment of O-antigen production confers resistance to grazing in a model amoeba–cyanobacterium predator–prey system. *Proceedings of the National Academy of Sciences of the United States of America*, 109(41), 16678–16683. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3478625/.

[55] Simkovsky, R., Effner, E. E., Iglesias-Sánchez, M. J., & Golden, S. S. (2016). Mutations in Novel Lipopolysaccharide Biogenesis Genes Confer Resistance to Amoebal Grazing in Synechococcus elongatus. *Applied and Environmental Microbiology*, 82(9), 2738–2750. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4836432/.

[56] Sobotka, R. (2014). Making proteins green; biosynthesis of chlorophyll-binding proteins in cyanobacteria. *Photosynthesis Research*, 119(1), 223–232. https://doi.org/10.1007/s11120-013-9797-2.

[57] Sonnhammer, E. L. & Östlund, G. (2015). InParanoid 8: Orthology analysis between 273 proteomes, mostly eukaryotic. *Nucleic Acids Research*, 43(Database issue), D234–D239. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4383983/.

[58] Ungerer, J. & Pakrasi, H. B. (2016). Cpf1 Is A Versatile Tool for CRISPR Genome Editing Across Diverse Species of Cyanobacteria. *Scientific Reports*, 6, 39681. http://www.nature.com/srep/2016/161221/srep39681/full/srep39681.html.

[59] Vanselow, C., Weber, A. P. M., Krause, K., & Fromme, P. (2009). Genetic analysis of the Photosystem I subunits from the red alga, Galdieria sulphuraria. *Biochimica et Biophysica Acta (BBA) - Bioenergetics*, 1787(1), 46–59. http://www.sciencedirect.com/science/article/pii/S0005272808006920.

[60] Vavilin, D. & Vermaas, W. (2007). Continuous chlorophyll degradation accompanied by chlorophyllide and phytol reutilization for chlorophyll synthesis in Synechocystis sp. PCC 6803. *Biochimica et Biophysica Acta (BBA) - Bioenergetics*, 1767(7), 920–929. http://www.sciencedirect.com/science/article/pii/S0005272807000783.

[61] Wang, L., Li, Q., Zhang, A., Zhou, W., Jiang, R., Yang, Z., Yang, H., Qin, X., Ding, S., Lu, Q., Wen, X., & Lu, C. (2017). The Phytol Phosphorylation Pathway Is Essential for the Biosynthesis of Phylloquinone, which Is Required for Photosystem I Stability in Arabidopsis. *Molecular Plant*, 10(1), 183–196. http://www.sciencedirect.com/science/article/pii/S1674205216303069.

[62] Wetmore, K. M., Price, M. N., Waters, R. J., Lamson, J. S., He, J., Hoover, C. A., Blow, M. J., Bristow, J., Butland, G., Arkin, A. P., & Deutschbauer, A. (2015). Rapid Quantification of Mutant Fitness in Diverse Bacteria by Sequencing Randomly Bar-Coded Transposons. *mBio*, 6(3), e00306–15. http://mbio.asm.org/content/6/3/e00306-15.

Appendix A

KNLab Scripts

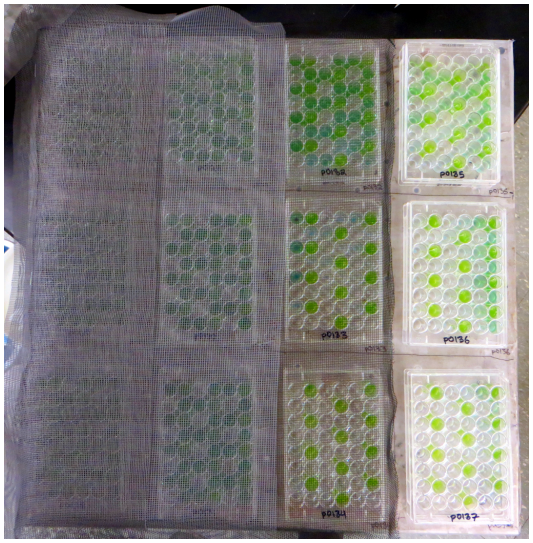
This section briefly describes several scripts I developed to help with common lab tasks, which may be useful to future researchers. The scripts are available at `github.com/jefdaj/knlab-scripts`.

A.1 High-throughput growth curves

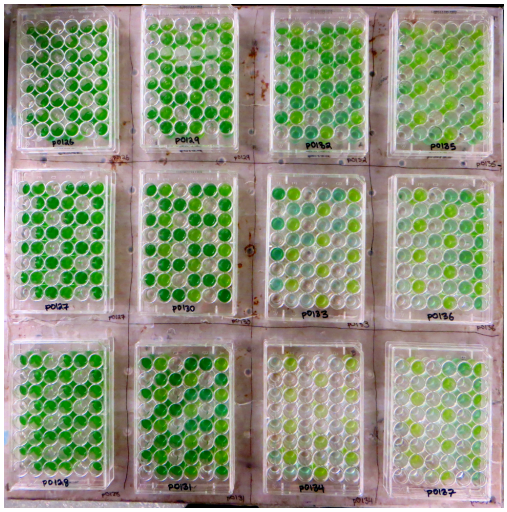
Before switching to the use of the *Synechococcus elongatus* PCC 7942 RB-TnSeq library, I started a similar investigation of 57 knockouts of predicted transcription factors (TFs) made by a previous student in the lab, Patrick Shih. I grew the strains in clear plates, plotted their growth, and looked for TFs whose absence caused high-light-specific growth defects. Growth in individual wells is highly variable, depending on factors like shaking speed, differential evaporation of wells on the edges of the plate, and irregular illumination of the growth chamber.

In the process of adjusting for that, I developed an efficient workflow for high-throughput growth curves using the Tecan M-1000 plate reader. I used many replicates (typically 3-6 per plate) of each strain, arranged in repeating patterns. The plates themselves can be filled quickly with a multichannel pipettor, so data analysis became the bottleneck. The `tecan-growth-curves` family of scripts streamlines it by extracting data from many Microsoft Excel sheets (one per plate per timepoint), merging it with separate metadata about the wells, and plotting it in R. Any relevant metadata can be included, for example strain, light level, antibiotic concentration, distance from the nearest edge or corner, or days grown.

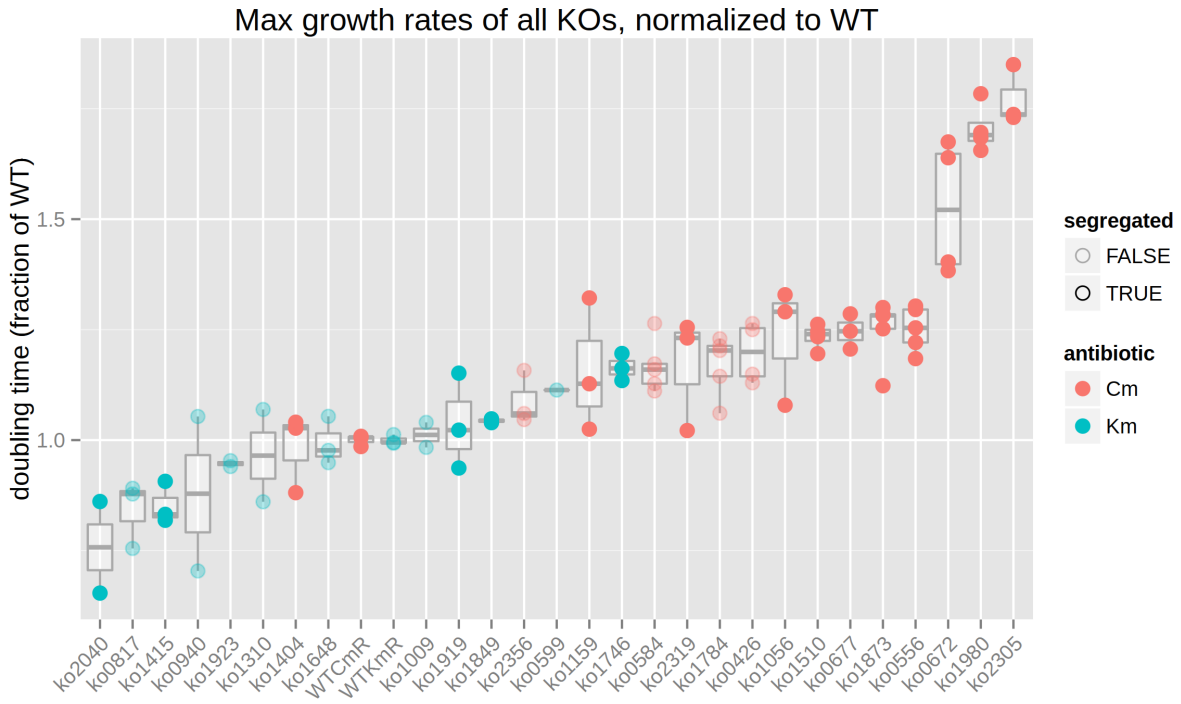
Figure A.1 shows an example of the plates and maximum growth rates per strain calculated from OD_{750} measurements. Figure A.2 shows an example of calculating those growth rates using the `tecan-growth-curves` scripts.



(a) Plates with shading mesh

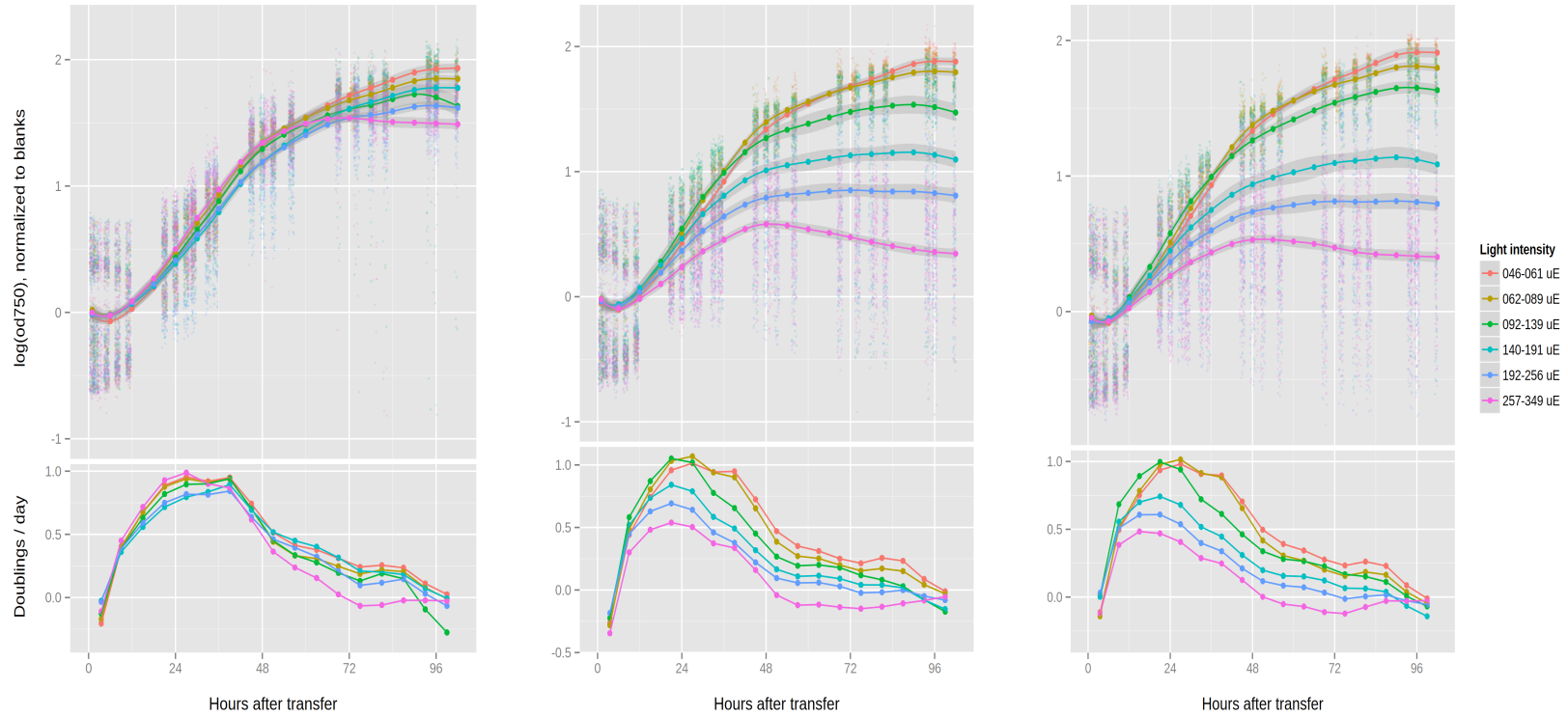


(b) Plates uncovered



(c) Growth rates

Figure A.1: a. A series of light levels created by shading mesh over clear 24-well plates. **b.** The same plates uncovered. An overall low to high light gradient phenotype is visible, as well as a "bright spot" in the bottom right caused by uneven illumination and repeating patterns caused by average differences in high light response between strains. **c.** Comparison of maximum growth rates derived from twice-daily OD_{750} measurements of all plates.



(a) WT **(b)** WT CmR **(c)** WT KmR

Figure A.2: Growth curves reveal a HL-specific defect caused by antibiotic resistance cassettes. The chloroamphenicol and kanamycin resistance genes are detrimental at high light, even when grown in plain BG11 media with no antibiotics. Individual noisy OD_{750} data points are shown as dots, overlayed with lines showing mean plus or minus standard deviation per strain. Doublings per day are calculated as the derivative of the mean growth curves.

A.2 QR code labels

I also developed a mundane but generally applicable `qrlabels` script for labeling small tubes or boxes with randomly-generated QR codes. The IDs can be tied to a lab database, which anyone can easily access by scanning the URL with their phone. The labels are freezer-safe to -20°C, but tend to fall off at -80°C or in liquid nitrogen.

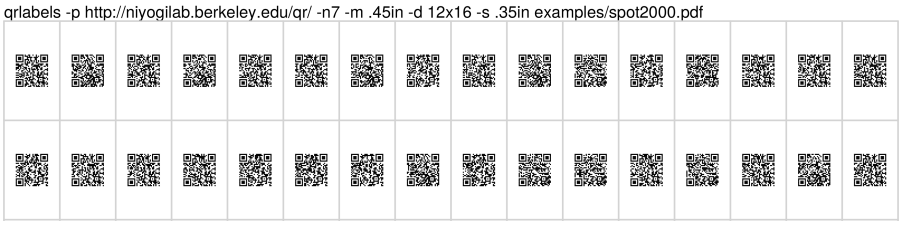


Figure A.3: QR codes suitable for printing on "Tough Spot" freezer-safe stickers. The `qrlabels` script generates small QR codes with a custom prefix + random UUID. Set the prefix to a database query URL to look up sample information via smartphone.

A.3 Cpf1-based knockout constructs in cyanobacteria

Finally, I am in the process of developing a script to automate initial primer selection for making cyanobacterial knockout and complementation constructs based on the Cpf1 nuclease, which is preferred in cyanobacteria due to Cas9 toxicity[58]. The script has been tested (Figure A.4) only on the *S. elongatus*genome so far, but the method should work in diverse cyanobacteria[58]. Primer design functions work, but the resulting constructs have not been transformed into cyanobacteria yet.

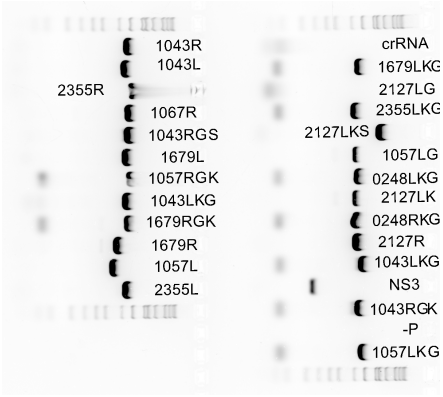


Figure A.4: PCR of flanking sequences for homologous recombination designed with the script.

Appendix B

ShortCut Reference

Table B.1: ShortCut file types as of version `0.8.4.11`.

Extension	File Type
aln	multiple sequence alignment
ava	all-vs-all hit table listing
bht	tab-separated table of blast hits (outfmt 6)
blh	BUSCO lineage HMMs
bsr	BUSCO results
bst	BUSCO scores table
crb	tab-separated table of conditional reciprocal blast best hits
dmnd	DIAMOND database
fa	FASTA (nucleic OR amino acid)
faa	FASTA (amino acid)
faa.gz	gzipped fasta amino acid (proteome)
fna	FASTA (nucleic acid)
fna.gz	gzipped fasta nucleic acid acid (gene list or genome)
gbk	genbank
gcr	GreenCut results
hht	HMMER hits table
hittable	BLAST hit table-like
hmm	hidden markov model
listlike	files that can be treated like lists
mms	MMSeqs2 sequence database
ndb	BLAST nucleotide database
num	number in scientific notation
ofr	OrthoFinder results
og	orthogroups (orthofinder, sonicparanoid, or greencut results)
pdb	BLAST protein database
png	plot image
pssm	PSI-BLAST position-specific substitution matrix as ASCII
search	intermediate table describing biomartr searches
spr	SonicParanoid results
tsv	set membership table (spreadsheet)

Table B.2: ShortCut function types as of version `0.8.4.11`.

Module	Function	Type Signature
Replace	replace	[oldend] [oldstart] [newstart] → [newend]
	replace_each	[oldend] [oldstart] [newstart].list → [newend].list
Repeat	repeat	[end] [start] num → [end].list
Math	*	num num → num
	+	num num → num
	-	num num → num
	/	num num → num
Load	glob_files	str → str.list
	load_list	str → str.list
Sets	&	[type].list → [type].list → [type].list
	all	[type].list.list → [type].list
	any	[type].list.list → [type].list
	diff	[type].list.list → [type].list
	some	[type].list.list → [type].list
	|	[type].list → [type].list → [type].list
		[type].list → [type].list → [type].list
SeqIO	concat_faa	faa.list → faa
	concat_faa_each	faa.list.list → faa.list
	concat_fna	fna.list → fna
	concat_fna_each	fna.list.list → fna.list
	extract_ids	fa → str.list
	extract_ids_each	fa.list → str.list.list
	extract_seqs	fa str.list → fa
	extract_seqs_each	fa.list → str.list.list
	gbk_to_faa	str gbk → faa
	gbk_to_faa_each	str gbk.list → faa.list
	gbk_to_fna	str gbk → fna
	gbk_to_fna_each	str gbk.list → fna.list
	load_faa	str → faa
	load_faa_each	str.list → faa.list
	load_faa_glob	str → faa.list
	load_fna	str → fna
	load_fna_each	str.list → fna.list
	load_fna_glob	str → fna.list
	load_gbk	str → gbk
	load_gbk_each	str.list → gbk.list
	load_gbk_glob	str → gbk.list
	split_faa	faa → faa.list
	split_faa_each	faa.list → faa.list.list
	split_fna	fna → fna.list
	split_fna_each	fna.list → fna.list.list
	translate	fna → faa
	translate_each	fna.list → faa.list
BiomartR	get_genomes	str.list → fna.gz.list
	get_proteomes	str.list → faa.gz.list
	parse_searches	str.list → search
BlastDB	blastdbget_nucl	str → ndb

Module	Function	Type Signature
	blastdbget_prot	str → pdb
	blastdblist	str → str.list
	load_nucl_db	str → ndb
	load_nucl_db_each	str.list → ndb.list
	load_prot_db	str → pdb
	load_prot_db_each	str.list → pdb.list
	makeblastdb_nucl	fa → ndb
	makeblastdb_nucl_all	fa.list → ndb
	makeblastdb_nucl_each	fa.list → ndb.list
	makeblastdb_prot	faa → pdb
	makeblastdb_prot_all	faa.list → pdb
	makeblastdb_prot_each	faa.list → pdb.list
	singletons	[type].list → [type].list.list
BLAST+	blastn	num fna fna → bht
	blastn_db	num fna ndb → bht
	blastn_db_each	num fna ndb.list → bht.list
	blastn_each	num fna fna.list → bht.list
	blastp	num faa faa → bht
	blastp_db	num faa pdb → bht
	blastp_db_each	num faa pdb.list → bht.list
	blastp_each	num faa faa.list → bht.list
	blastx	num fna faa → bht
	blastx_db	num fna pdb → bht
	blastx_db_each	num fna pdb.list → bht.list
	blastx_each	num fna faa.list → bht.list
	concat_bht	bht.list → bht
	concat_bht_each	bht.list.list → bht.list
	megablast	num fna fna → bht
	megablast_db	num fna ndb → bht
	megablast_db_each	num fna ndb.list → bht.list
	megablast_each	num fna fna.list → bht.list
	tblastn	num faa fna → bht
	tblastn_db	num faa ndb → bht
	tblastn_db_each	num faa ndb.list → bht.list
	tblastn_each	num faa fna.list → bht.list
	tblastx	num fna fna → bht
	tblastx_db	num fna ndb → bht
	tblastx_db_each	num fna ndb.list → bht.list
	tblastx_each	num fna fna.list → bht.list
BlastHits	best_hits	hittable → bht
	best_hits_each	hittable.list → bht.list
	extract_queries	hittable → str.list
	extract_queries_each	hittable.list → str.list.list
	extract_targets	hittable → str.list
	extract_targets_each	hittable.list → str.list.list
	filter_bitscore	num hittable → bht
	filter_bitscore_each	num hittable.list → bht.list
	filter_evalue	num hittable → bht
	filter_evalue_each	num hittable.list → bht.list

Module	Function	Type Signature
	filter_pident	num hittable → bht
	filter_pident_each	num hittable.list → bht.list
ListLike	length	listlike → num
	length_each	listlike.list → num.list
PsiBLAST	psiblast	num faa faa → bht
	psiblast_all	num faa faa.list → bht
	psiblast_db	num faa pdb → bht
	psiblast_db_each	num faa pdb.list → bht.list
	psiblast_each	num faa faa.list → bht.list
	psiblast_each_pssm	num pssm.list faa → bht.list
	psiblast_each_pssm_db	num pssm.list pdb → bht.list
	psiblast_pssm	num pssm faa → bht
	psiblast_pssm_all	num pssm faa.list → bht
	psiblast_pssm_db	num pssm pdb → bht
	psiblast_pssm_db_each	num pssm pdb.list → bht.list
	psiblast_pssm_each	num pssm faa.list → bht.list
	psiblast_pssms	num pssm.list faa → bht
	psiblast_pssms_all	num pssm.list faa → bht
	psiblast_pssms_db	num pssm.list pdb → bht
	psiblast_train	num faa faa → pssm
	psiblast_train_all	num faa faa.list → pssm
	psiblast_train_db	num faa pdb → pssm
	psiblast_train_db_each	num faa pdb.list → pssm.list
	psiblast_train_each	num faa faa.list → pssm.list
	psiblast_train_pssms	num faa.list faa → pssm.list
	psiblast_train_pssms_db	num faa.list pdb → pssm.list
CRB-BLAST	crb_blast	fna fa → crb
	crb_blast_each	fna fa.list → crb.list
HMMER	extract_hmm_targets	hht → str.list
	extract_hmm_targets_each	hht.list → str.list.list
	hmmbuild	aln → hmm
	hmmbuild_each	aln.list → hmm.list
	hmmsearch	num hmm faa → hht
	hmmsearch_each	num hmm.list faa → hht.list
BlastRBH	blastn_rbh	num fna fna → bht
	blastn_rbh_each	num fna fna.list → bht.list
	blastn_rev	num fna fna → bht
	blastn_rev_each	num fna fna.list → bht.list
	blastp_rbh	num faa faa → bht
	blastp_rbh_each	num faa faa.list → bht.list
	blastp_rev	num faa faa → bht
	blastp_rev_each	num faa faa.list → bht.list
	megablast_rbh	num fna fna → bht
	megablast_rbh_each	num fna fna.list → bht.list
	megablast_rev	num fna fna → bht
	megablast_rev_each	num fna fna.list → bht.list
	reciprocal_best	bht bht → bht
	reciprocal_best_all	bht.list bht.list → bht
	tblastx_rbh	num fna fna → bht

Module	Function	Type Signature
	tblastx_rbh_each	num fna fna.list → bht.list
	tblastx_rev	num fna fna → bht
	tblastx_rev_each	num fna fna.list → bht.list
MUSCLE	muscle	faa → aln
	muscle_each	faa.list → aln.list
Sample	sample	num [type].list → [type].list
Permute	leave_each_out	[type].list → [type].list.list
Scores	extract_scored	[type].scores → [type].list
	extract_scores	[type].scores → num.list
	score_repeats	[oldend] [oldstart] [newstart].list → [newend].scores
Plots	histogram	str num.list → png
	linegraph	str num.scores → png
	scatterplot	str num.scores → png
	venndiagram	[type].list.list → png
OrthoFinder	orthofinder	faa.list → ofr
Diamond	diamond_blastp	num faa faa → bht
	diamond_blastp_db	num faa dmnd → bht
	diamond_blastp_db_each	num faa dmnd.list → bht.list
	diamond_blastp_db_more_sensitive	num faa dmnd → bht
	diamond_blastp_db_more_sensitive_each	num faa dmnd.list → bht.list
	diamond_blastp_db_more_sensitive_rev	num dmnd faa → bht
	diamond_blastp_db_rev	num dmnd faa → bht
	diamond_blastp_db_sensitive	num faa dmnd → bht
	diamond_blastp_db_sensitive_each	num faa dmnd.list → bht.list
	diamond_blastp_db_sensitive_rev	num dmnd faa → bht
	diamond_blastp_each	num faa faa.list → bht.list
	diamond_blastp_more_sensitive	num faa faa → bht
	diamond_blastp_more_sensitive_each	num faa faa.list → bht.list
	diamond_blastp_more_sensitive_rev	num faa faa → bht
	diamond_blastp_more_sensitive_rev_each	num faa faa.list → bht.list
	diamond_blastp_rev	num faa faa → bht
	diamond_blastp_rev_each	num faa faa.list → bht.list
	diamond_blastp_sensitive	num faa faa → bht
	diamond_blastp_sensitive_each	num faa faa.list → bht.list
	diamond_blastp_sensitive_rev	num faa faa → bht
	diamond_blastp_sensitive_rev_each	num faa faa.list → bht.list
	diamond_blastx	num fna faa → bht
	diamond_blastx_db	num fna dmnd → bht
	diamond_blastx_db_each	num fna dmnd.list → bht.list
	diamond_blastx_db_more_sensitive	num fna dmnd → bht
	diamond_blastx_db_more_sensitive_each	num fna dmnd.list → bht.list
	diamond_blastx_db_more_sensitive_rev	num dmnd fna → bht
	diamond_blastx_db_rev	num dmnd fna → bht
	diamond_blastx_db_sensitive	num fna dmnd → bht
	diamond_blastx_db_sensitive_each	num fna dmnd.list → bht.list
	diamond_blastx_db_sensitive_rev	num dmnd fna → bht
	diamond_blastx_each	num fna faa.list → bht.list
	diamond_blastx_more_sensitive	num fna faa → bht
	diamond_blastx_more_sensitive_each	num fna faa.list → bht.list

Module	Function	Type Signature
	diamond_blastx_more_sensitive_rev	num faa fna → bht
	diamond_blastx_more_sensitive_rev_each	num faa fna.list → bht.list
	diamond_blastx_rev	num faa fna → bht
	diamond_blastx_rev_each	num faa fna.list → bht.list
	diamond_blastx_sensitive	num fna faa → bht
	diamond_blastx_sensitive_each	num fna faa.list → bht.list
	diamond_blastx_sensitive_rev	num faa fna → bht
	diamond_blastx_sensitive_rev_each	num faa fna.list → bht.list
	diamond_makedb	faa → dmnd
	diamond_makedb_all	faa.list → dmnd
	diamond_makedb_each	faa.list → dmnd.list
MMSeqs	mmseqs_createdb	fa → mms
	mmseqs_createdb_all	fa.list → mms
	mmseqs_search	num fa fa → bht
	mmseqs_search_db	num fa mms → bht
SonicParanoid	sonicparanoid	faa.list → spr
OrthoGroups	orthogroup_containing	og str → str.list
	orthogroups	og → str.list.list
	orthogroups_containing	og str.list → str.list.list
	ortholog_in_all	og faa.list → str.list.list
	ortholog_in_all_str	str.list.list str.list.list → str.list.list
	ortholog_in_any	og faa.list → str.list.list
	ortholog_in_any_str	str.list.list str.list.list → str.list.list
	ortholog_in_max	num og faa.list → str.list.list
	ortholog_in_max_str	num str.list.list str.list.list → str.list.list
	ortholog_in_min	num og faa.list → str.list.list
	ortholog_in_min_str	num str.list.list str.list.list → str.list.list
Busco	busco_fetch_lineage	str → blh
	busco_filter_completeness	num bst faa.list → faa.list
	busco_list_lineages	str → str.list
	busco_percent_complete	bsr → num
	busco_percent_complete_each	bsr.list → num.list
	busco_proteins	blh faa → bsr
	busco_proteins_each	blh faa.list → bsr.list
	busco_scores_table	bsr.list → bst
	busco_transcriptome	blh fna → bsr
	busco_transcriptome_each	blh fna.list → bsr.list
	concat_bst	bst.list → bst
	load_lineage	str → blh
Range	range_add	num num num → num.list
	range_exponent	num num num num → num.list
	range_integers	num num → num.list
	range_length	num num num → num.list
	range_multiply	num num num → num.list
SetsTable	sets_table	lit.list.list → tsv
GreenCut	greencut2_families	bht bht → gcr