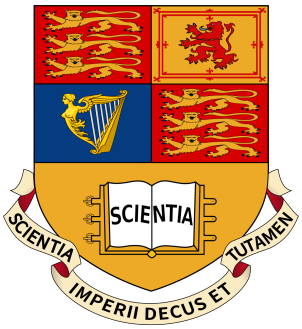
Imperial College
London

Final Report for MEng Project

Using synthetic biology to engineer yeast as biosensors for high-value chemicals

Author: Olivia Gallupová
CID: 01349379
Word count: 5997
Supervisor: Professor Tom Ellis



Department of Bioengineering
Imperial College London
15th June 2021

Acknowledgements

I would like to thank Professor Tom Ellis for his supervision, guidance and advice that continuously grounded the focus of this project. I am indebted to Fankang Meng for his mentorship in the lab, in synthetic biology and in conducting research, and for his motivation when things became difficult. I would also like to sincerely thank Will Shaw for his generous guidance from overseas without which this project would not have been attemptable.

My gratitude goes to members of the Center for Synthetic Biology who have been extremely welcoming and helped me in the lab, especially Xinyu Lu, Anastasiya Malysheva, Mateo Sanchez Lopez, Amritpal Singh, and Lucie Studena.

Contents

Abstract

Since its inception, synthetic biology has been motivated by the promise of more sustainable manufacturing methods. Engineered microbes as biochemical factories inspired some of the earliest work in the field and remain one of the drivers of industry success. However, with the near infinite number of ways that metabolic pathways may be altered to create more efficient candidate strains, better screening methods are needed. In this project, a yeast sender-receiver system was designed to automatically select for the most efficient microbial producers of a desired compound. Through the engineering of novel yeast strains, such a system achieves increased modularity and simplicity compared to previous approaches.

1 Introduction

In recent years, the rising need for sustainable manufacturing methods has spurred development in diverse areas in science, but especially in synthetic biology. Since its inception, the field has been hailed as a potential cure-all technology for solving complex world issues, such as climate change, food supply, or water remediation, in new and creative ways. One of the most straightforward and successful industrial applications working towards these and other issues has been the appropriation of finely tuned microbes as chemical factories. In fact, building with biology has been the central focus for customer-facing synthetic biology companies in many industries such as food, health, and textiles thanks to the promises of innovating more sustainable manufacturing. Simultaneously, one of the biggest criticism of synthetic biology has been on its inability to deliver on the scale of its promises. Only products in luxury markets or expensive medicines in healthcare have been competitive enough with alternative approaches to persist. Optimising production pathways in microbial population is difficult and often does not scale well to industrial use [1] [2].

Fortunately, synthetic biology as a discipline has matured greatly over the past two decades [3] [4] [5], to the point where systems can be engineered with multiple genetic circuits distributed across different microbes within a population, approaching multi-cellularity. Thanks to the increasingly diverse range of intercellular communication channels characterised or synthesized, more complex feedback behaviors can now be elicited in the classic workhorse microbes of synthetic biology [6] [7] [8]. The distribution of genetic devices throughout a cell population has many advantages for reducing the burden put on individual cells for executing a desired heterologous function. Distributed cell networks also present a novel opportunity for expanding the range of logic operations for biocomputation and modularising whole populations. Biosensing plays a central role in the underlying mechanisms enabling co-cultures to co-exist and cooperate [9], leading to a renewed wave of research on GPCR sensors, a diverse class of eukaryotic membrane receptor that present a highly customisable platform for intercellular communication [10]. Yeast has two native and well-studied GPCR response cascades, one for sensing sugar and one for pheromones (see Fig. 1), the latter of which has been engineered previously to respond to melatonin [11]. A co-culture built upon such a modular sensing system with both intra- and intercellular control mechanisms (Fig. 2) could be used to streamline the testing phase in the Design-Build-Test-Learn cycle by providing room for additional logic downstream of sensing and thereby open up more systematic approaches to optimising production. While a decoupled version of this system grown in liquid culture has already been proven to increase throughput by using a sensor strain to select for high producers via fluorescence and by-passing lengthy mass spectrometry data collection [12], the selection process could be further simplified by culturing the sensor and producer strains together and allowing a feedback mechanism to find the best producer strain.

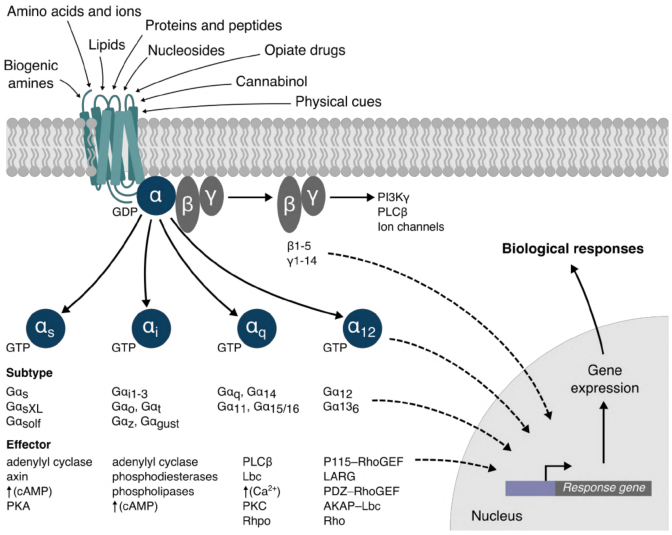


Figure 1: Yeast pheromone response pathway uses a GPCR pheromone sensor that can be replaced with other GPCR sensors, such as the melatonin or adenine GPCRs [11].

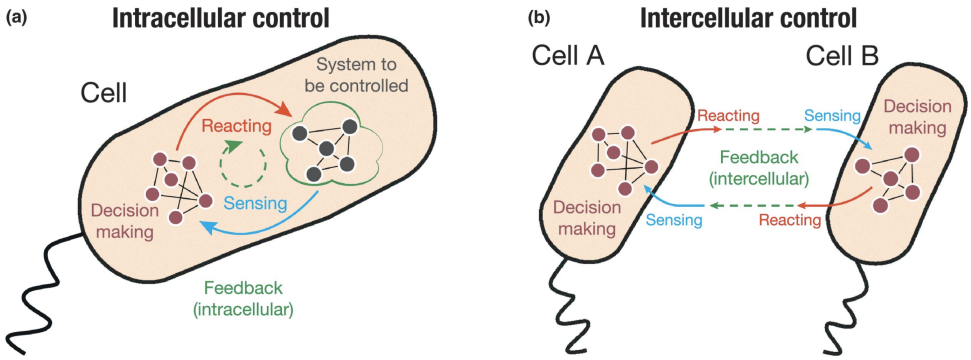


Figure 2: Reproduction of Figure 1a and 1b from [8] showing different realms of control in cells. The co-culture system in this project incorporates both intracellular and intercellular control.

1.1 Project Aims

The aim of this project is to construct a co-culture system that automatically selects for high production rates of any desired compound, as in Fig. 3. The key components include a sensor strain with a feedback mechanism that rewards the production of the high-value compound. The reward mechanism chosen in this case is the enrichment of the nutrient environment localised with higher levels of the target compound. This would be accomplished through the conditional secretion of invertase, yeast's enzyme for breaking down sucrose into glucose and fructose, upon sensing the target. Subsequently, the only requirement of the producer strain is that it cannot reward itself, in this case meaning that it is unable to produce invertase. The target molecule was chosen to be melatonin, as it is secreted from yeast naturally and a set of GPCR-based melatonin sensor strains were available. In the final system (Fig. 4), a library of producers would be mixed with a sensor strain and plated together onto a petri dish, then after sufficient growth time, the highest melatonin producer could directly be identified by colony size.

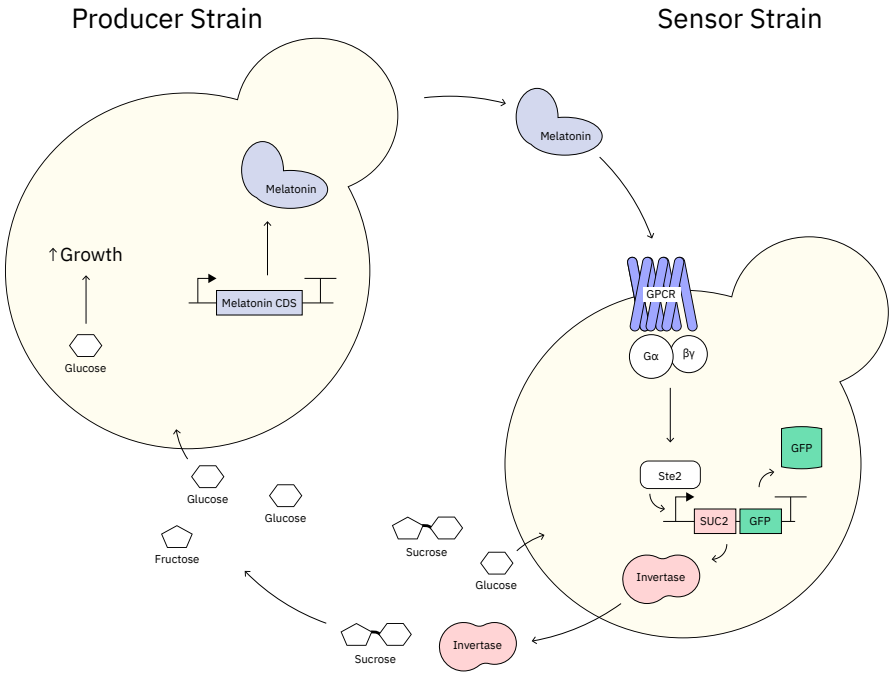
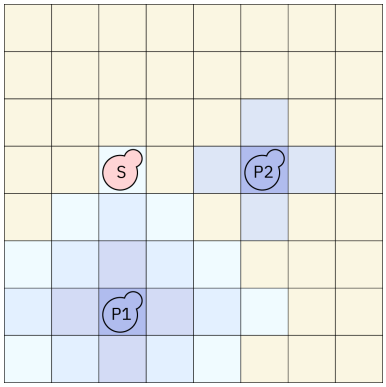
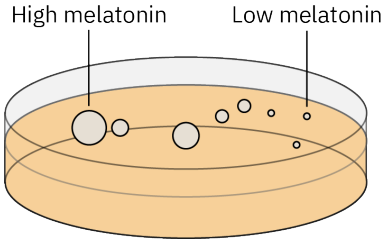


Figure 3: Sensor-producer co-culture system with MTNR1A as the Sensor strain and one of a variety of melatonin producing strains as the Producer strain. Invertase promotes growth in the producer strain, which promotes the production of melatonin.

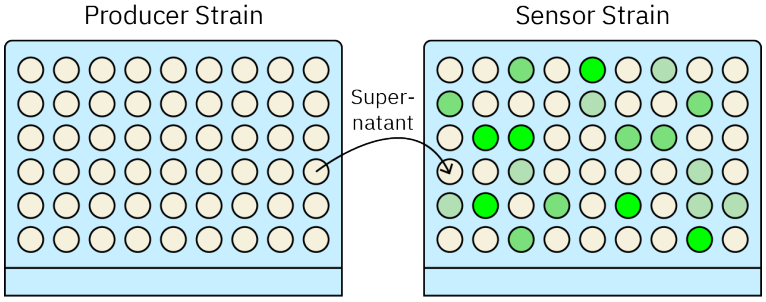
There are several challenges that come with this set-up. In terms of parameters that can be tuned, there is the operational range of the sensor, which must encompass the low and high levels of production, and the expression levels of the reward. The optimal starting densities of each strain, along with diffusion distance of the target and reward molecules, also play a role in the overall sensitivity. Inhomogeneities in the initial cell mixture may then affect the density of the reward and its ability to specifically reward higher producers, thus confounding how producers are affected by the reward. The grid in Fig. 4a presents a convenient way to visualise this problem - a sensor S is only activated by a high producer P1, but due to its equidistance from P1 and low producer P2, it ends up rewarding producers of different strengths the same. Finally, the growth media may have some effect not only on the growth rate of the strains, but also on the diffusivity of the feedback molecules.



(a) Agar dish as grid of sensor strain (S) equidistant to a high (P1) and low producer (P2), with yellow denoting media and blue melatonin.



(b) Nutrient reward from sensed melatonin makes colonies with high producers bigger on petri dish in sensor and producer co-culture system.



(c) GPF fluorescence response in sensor from supernatant of producer (as in [12]).

Figure 4: Theorised system of co-cultures (4a 4b) and previously published decoupled selection [12].

1.2 Background

Contemporary microbial screening techniques

There are currently a number of popular approaches to screening the efficiency of microbial factories. Both mass spectrometry (MS) and liquid and gas chromatography (LC) offer the benefit of quantifying the amount of a target compound in a highly versatile and universal way. Combined, LC-MS may require custom sample preparation with chemical agents to better separate molecules, similarly to other assays that can be coupled to subsequent MS such as solid phase extraction and selected-reaction monitoring. This highlights how much time and expertise is required to increase the sensitivity of MS appropriately and how its slow speed inhibits throughput [13]. Efforts aim to simplify and accelerate the process through techniques such as ultra-fast LC-MS [1] and streamlined MS in high-volume, boundary-pushing projects [14]. Novel MS technology, such as the Echo MS, also promise to progress the throughput and sample specificity achievable by internally combining steps in the pipeline. Nevertheless, the rich analysis offered by these approaches identifies possible red flags in the pathway design for debugging, this kind of fine-grained analysis often cannot guarantee the right combination of factors to yield the optimum output.

Fortunately, the explosion of viable candidate numbers through combinatorial library preparation and directed evolution [15] [2] [16] [17] may bypass the need for detailed understanding and predictability of complex biological systems. Discerning cell type via reporters, which Zhang et al. (2015) [18] argue poses an attractive method for efficient screening along with cell survival, in fluorescence-activated cell sorting (FACS) can quickly classify producers cells into a myriad of desirable categories. For example, Wu et al. (2019) [19] were able to differentiate cell state based on fluorescence intensity and identify the appropriate promoters from their library through next-generation sequencing (NGS). Others utilised FACS to similarly sort through a library of elevated producers [20] [18] [21] [22] and confirm the function of specific parts [23]. While enabling versatile

selection strategies and retention of microbes for downstream work, multiple analysis rounds may be required despite reduced accessibility due to highly specialised instruments and software, furhter highlighting the need for this project.

2 Methods

2.1 Modelling

The spatial study of cell growth is amenable to modelling by collision-based and partial derivative models (as well as a combination of the two). In the former, sometimes termed agent-based, an n-dimensional grid is set up with each unit in the grid corresponding to a certain number and shape of cells. Each unit of the grid also stores information about the environment that the cells are in, such as nutrient and molecule concentrations. A set of update rules and equations then governs the exchange of particles between grid units and cell growth, making the cells into mathematically driven machines termed cellular automata. Concurrently, partial derivatives can be used to simulate particle diffusion in space and time, cell flux and metabolism, molecular kinetics, and cell proliferation. The model used in this project is mainly based on cellular automata and uses the heat equation to simulate the diffusion of particles across a 2D grid.

The model attempts to mirror conditions on a petri dish and consists of a stack of 2D grids containing information about cell identity, probability of cell division and death, as well as the amount of melatonin, sucrose, glucose, and invertase in a given cell. Sucrose content was assumed to be much greater than could be used up by the simulation from invertase activity, while cells could only consume glucose. Grid units with no glucose content had their probability of division set to zero, while positive glucose content increased the probability of division in a scalable way. In line with literature about yeast growth on solid media [24], the growth rate follows a Gompertz equation with an additional baseline growth rate to simulate linear growth in the stationary phase. Parameters were taken from literature and are summarised in Table 1. The model was built in Python with Google Colab's Jupyter Notebook (script included in Appendix D).

Parameter	Value	Units	Reference
Asymptotic growth (A)	233.16	N/A	[24]
Maximal growth rate (μ_{max})	5.95353313081e-7	N/A	[24]
Lag time (λ)	2.11020342857e-5	N/A	[24]
Doubling time yeast	90	min	[25]
Diffusivity glucose	6e4	cm^2s^{-1}	[26]
Diffusivity invertase	1e5	cm^2s^{-1}	[26]
Diffusivity sucrose	1e5	cm^2s^{-1}	[26]
Diffusivity melatonin	4.2e4	cm^2s^{-1}	[26]
Consumption rate glucose	1.6e-6	cm^2s^{-1}	[26]
Grid length	4e-4	cm	[27]
Invertase catalysis rate (k_{cat})	1e4	s^{-1}	[28]
Michaelis-Menten constant invertase (K_M)	0.02	M	[28]
Grid total height & width	400	N/A	
Time increment (dt heat equation)	1.111	s	
Baseline probability of division	0.001	N/A	

Table 1: Parameters used in cellular automata model. Found from literature and calculations

2.2 Strains and culturing

Information about the yeast genome (eg annotations of the invertase gene) was obtained from the *Saccharomyces* Genome Database (SGD), while information on parts from the lab culture collection and YTK collection was accessed through the lab group's Benchling. The yeast strains used in this project correspond to melatonin sensor yWS1544 and producer strains previously published

by Will Shaw [11], where the producer strains originated from [29]. Three of these producers were selected that had low, medium, and high rates of melatonin production relative to all strains available. Table 2 summarises the melatonin concentration measured in the supernatant of each producer strain. The sensor strain has a corresponding operational range corresponding well to the range of the producer concentrations with a K_A in the range of 10^{-7} M, as shown by Fig. 5 (reproduced from [11]). The strains were picked from frozen -80°C glycerol stocks (picking green where there were duplicates), cultured overnight in YPD, then resuspended 1:1 in 50% glycerol and re-stored into a working stock for handling during the project to prevent contamination of the original stock.

Yeast strain index	Supernatant [Melatonin] (M)
H2	1.48310251434e-8
H5	5.95353313081e-7
H4	2.11020342857e-5

Table 2: Melatonin producer strains in order of ascending melatonin supernatant concentrations.

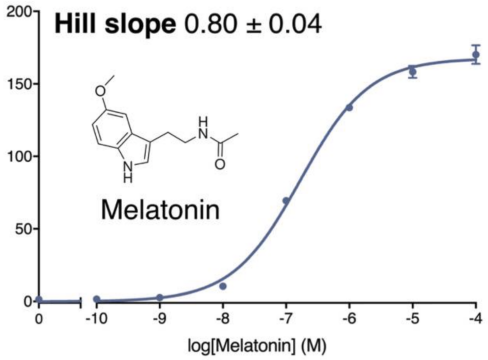


Figure 5: Reproduction of Figure S6 D from [11] showing the operational range of the GPCR melatonin sensor in yeast, where the y-axis shows GFP fluoresence (fold over background).

2.3 Gene knock-outs and knock-ins using CRISPR and YTK

There are two mechanisms by which yeast repair their DNA - non-homologous end-joining (NHEJ) and homology directed repair (HDR). In NHEJ, a double-strand break is fixed by ligating sticky ends directly together and is the dominant method of DNA repair in mammalian cells [30]. Compared to other microorganisms, yeast are proficient at homologous recombination, a process whereby a broken piece of DNA is repaired using another, identical or highly similar strand as a template. This process can be appropriated for the modification of regions of DNA by introducing a double-strand break using an enzyme together with a synthetic DNA template. The gene of choice to be introduced is then placed between the so-called template donor regions that are upstream and downstream of the modification site. In the case of a knock-in, the new gene would sit between the upstream and downstream donors, whereas in a knock-out, the donor regions would be directly adjacent. The double strand breakage can be introduced at the desired site most conveniently with CRISPR, which can be programmed with custom DNA sequences as opposed to the strict recognition patterns used by most restriction enzymes. Details about assembling with YTK and cloning methods can be found in Appendix A.

2.4 SUC2 Knock-out

The design for knocking out the SUC2 gene was based around this CRISPR and YTK framework. First, a donor region was selected 500bp upstream and downstream of the target deletion region (see C.2 details), which included the SUC2 gene and 700bp downstream of the gene to include any

termination sequences in the deletion. Next, CRISPR guides were designed of length 20bp at either end of the deletion region (gRNA Up and Dn 1). These sequences were then used as the guide RNA (gRNA) for assembling the CRISPR complex using the YTK 2.0 CRISPR Toolkit [31] (Appendix A). The gRNA guides were assembled onto a CRISPR guide vector using a gRNA scaffold plasmid as a template to include the appropriate gRNA machinery preceding the recognition sequence (see C.2 Table 15). The donor sequences were PCR-amplified from the yeast genome then integrated into an entry-level vector from the YTK standard. The CRISPR guide vectors pOG102 and the donor plasmid pOG001 were then transformed into both producer and sensor yeast strains.

Modification to original knock-out design

The transformations with the first design of gRNA recognition sites were unsuccessful after multiple repetitions, the reasons for which will be discussed in Section 4. In the redesign, two other yeast antibiotic selection markers were used besides zeocin for the gRNA plasmids to persist in the cell. A new deletion region was outlined around 800bp further upstream of the start original deletion region, with similar length. From this, a new set of gRNA guides was chosen (gRNA Up and Dn 2) again of length 20bp. The donor sequences for this were shortened from 500bp each to 100bp total with 50bp on either side of the nick cut by the CRISPR enzyme. These new donor sequences were created using long overlapping primers, while the original 500bp donor sequences were reassembled onto plasmids with zeocin and hygramycin resistance genes to reattempt the original transformation. The gRNA guides were integrated into three different backbones plasmids with zeocin, hygramycin, and nourseothricin resistance genes respectively. These antibiotics were chosen due to their effectiveness as yeast antibiotics and to offer alternatives to zeocin as a marker alone. Hygramycin and zeocin are both light-sensitive and were wrapped in aluminum foil to shield from light.

2.5 SUC2 Knock-in

Once the native, constitutive SUC2 gene is knocked out from both the sensor and producer strains, the conditional expression of SUC2 must be introduced. To accomplish this, the SUC2 gene would be placed under the control of promoters responding to the yeast pheromone GPCR pathway, which in the working sensor strain yWS1544 has previously been replaced with a melatonin GPCR sensor. This inducible promoter would then be integrated into the yeast genome along with the SUC2 coding sequence. Similarly to the knock-out, the YTK and CRISPR framework described in Section 2.3 would be used for this.

Strength characterisation

As it remains an open question as to what the optimal rate of invertase secretion is that effectively rewards surrounding cells while still maintaining selectivity, part of the SUC2 knock-in step is to characterise the levels of invertase expression. This can be accomplished by initially testing expression using a range of constitutive promoters. Based on how well the producer strain responds to the output of the invertase, how higher invertase production rates affect its diffusion rate, and how well the environment can support the enzymatic activity, the optimal expression level would be selected. The optimal constitutive promoter can then have its expression level matched up to pre-existing data collected on the expression levels of the different GPCR cascade inducible promoters. Thus a library of characterisation plasmids would be created with the Histidine auxotrophic marker serving as the integration locus, as summarised in Table 18. The plasmids for integrating the GPCR inducible promoter and SUC2 gene would be assembled with YTK and are summarised in Table 19.

3 Results

3.1 Modelling cell growth

The key aspect of the cellular automata model was to simulate the growth of the co-culture and measure the resulting differences in cell growth rates. The plot in Fig. 6 shows that over many repeats and after quasi-stationary growth is reached (which in solid media conditions becomes

linear after the log phase [cite]), the different production levels of melatonin can indeed sort the producers by cell count. The first standard deviation is shown via transparence on each of the growth curves.

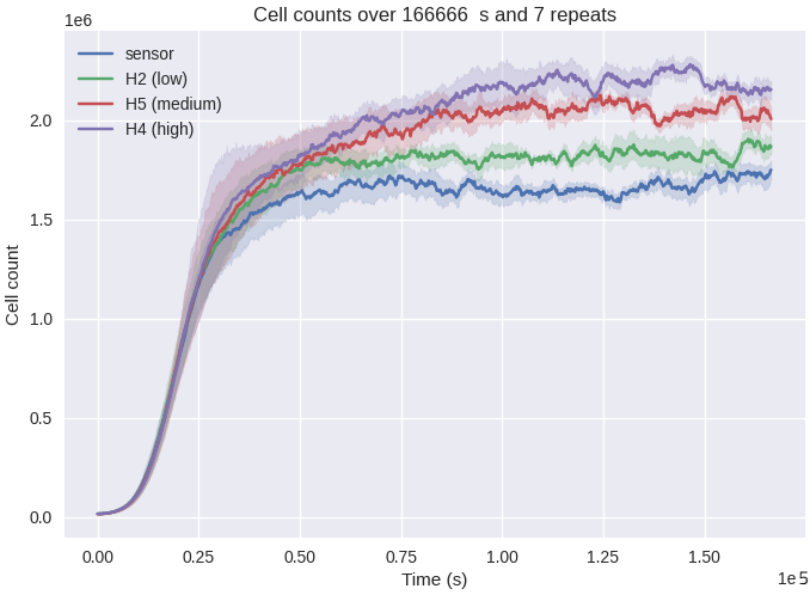


Figure 6: Time graph of cell count change in cellular automata model over ca. 46hrs. Stationary phase reached after ca. 7 hours simulation time.

3.2 Knocking out SUC2 Gene

3.2.1 Selection on antibiotics

First knock-out transformation

The transformation of yeast strains using the knock-out plasmids with zeocin resistance were difficult at first to distinguish from control plates. Initial transformations as in Fig. 7 showed no selection against untransformed yeast strains, as these petri dishes showed the same level of growth as their control counterparts. In Fig. 8, the strains were each cultured with different concentrations of zeocin, starting at the recommended 200ug/ml [cite] as advised by Will Shaw and going up to 250ug/ml and 300ug/ml. The ring of colonies identifiable around the edges of the petri dishes results from the top-streaking streaking method used, which was done in preference to embedding antibiotic into the growth medium directly, as the light-sensitivity of zeocin would make such petri dishes difficult to store and handle.

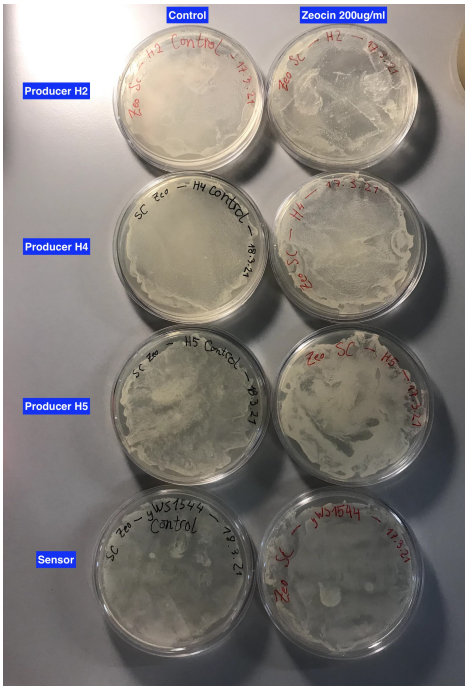
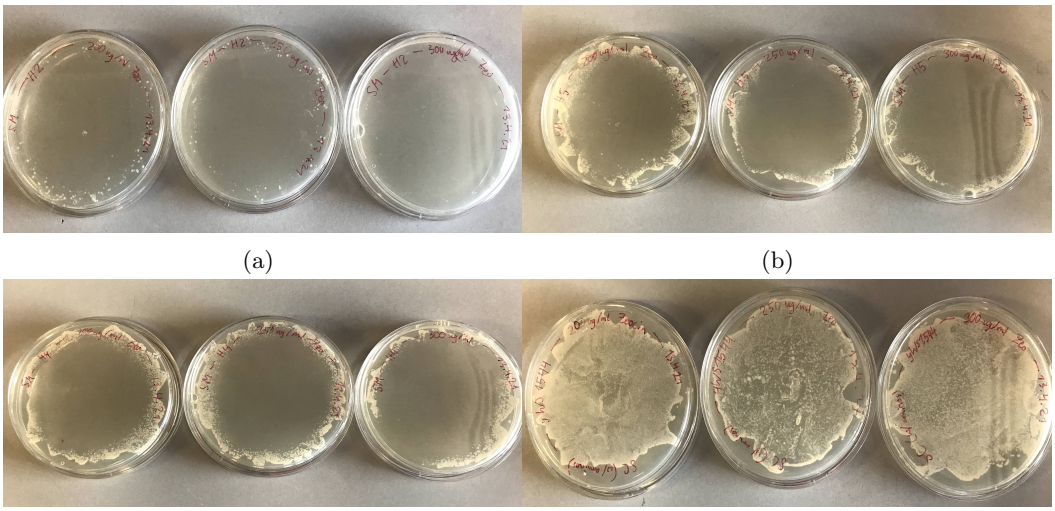


Figure 7: Transformed (with pOG001 and pOG101) and control yeast sensor and producer strains plated onto SC Agar with and without top-streaked zeocin (200ug/ml) both show colonies across plate. Zeocin initially difficult to handle as a selection marker.



(a) (b)

(c) (d)

Figure 8: Parameterisation of yeast strain response to zeocin antibiotic with concentrations 200ug/ml, 250ug/ml, and 300ug/ml shows response to recommended dose (200ug/ml) using 200ul transformed cell mix resuspended and streaked with 100ul $CaCl_2$.

3.2.2 Verification of SUC2 Knock-out

After transformation, the gene knock-outs of SUC2 were tested for via colony PCR. The colonies that showed no bands of SUC2 length were then grown overnight in SC media and cultured on plates, then tested again via colony PCR (see Fig. 9). Finally, the colonies that showed positive results from this second round of PCR were grown overnight then plated on sucrose-only SC agar

to check that they cannot grow (see Fig. 10). One of the producer strains, H5, and two of the petri dishes stemming from different colonies of the sensor strain yWS1544 had no colonies. While the WT and two of the other producer plates showed a lawn of colonies, one of the sensor plates had several colonies on it.

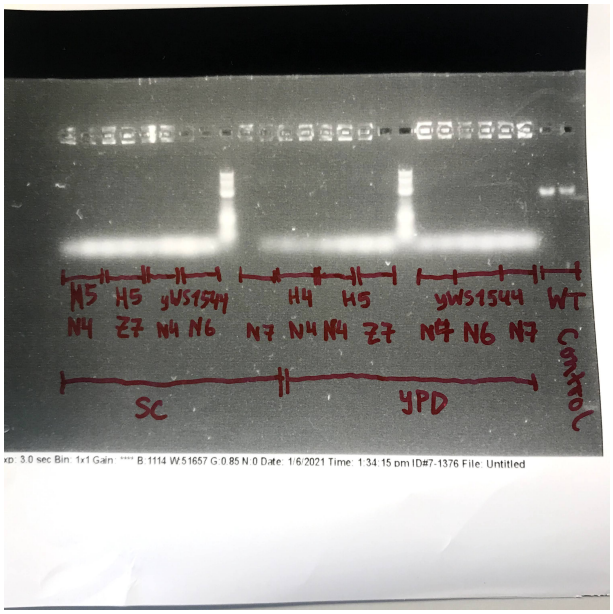
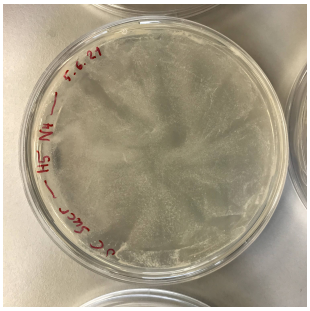


Figure 9: Gel image confirming SUC2 knockout. WT strains show bars the length of the SUC2 gene, while transformed strains do not. Strains were grown in SC media as well as a backup in YPD to ensure no loss of colonies.

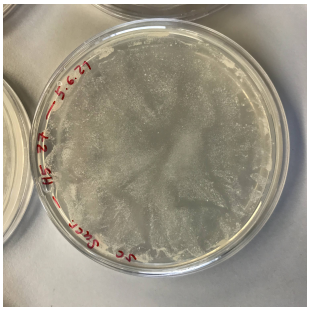
3.3 Constructing SUC2 knock-in vectors

The assembly of the YTK promoter library (as summarised in Table 18) was attempted several times and tested via colony PCR (gel image shown in Fig. 11) after Golden Gate assembly, transformation and plating, as well as enzyme digests. The backbone and terminator parts used for assembly from the YTK toolkit and the lab culture collection were checked for quality via a BsaI digest and matched the expected band lengths as generated by Benchling (see Fig. 12). The SUC2 part used in this assembly was used as a gene fragment simply amplified from the genome and purified.

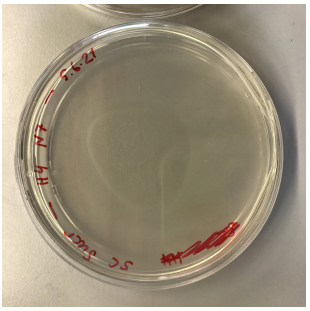
Another amplification of the SUC2 part was attempted with YTK overhangs to allow its integration into a YTK entry level vector for better storage and ease of amplification, as well as to introduce a point mutation at the BsaI site within the gene (at base pair index 38479). A set of primers with YTK-compatible overhangs was used flanking the SUC2 gene, while a second pair of primers served to introduce a point mutation at bp index 38474, changing a TCT to a TCG to disrupt the BsaI recognition sequence without modifying the Serine coding sequence. The templates used for this included 3 different isolations of the yeast genome and the SUC2 fragment previously isolated without overhangs. Several PCR settings were also were used, including Touchdown-PCR (starting annealing temperature of 72°C used with decrements for 17 cycles, then 55°C for 20 cycles) and standard PCR without changing annealing temperatures, the results of the former being shown in Fig. 13.



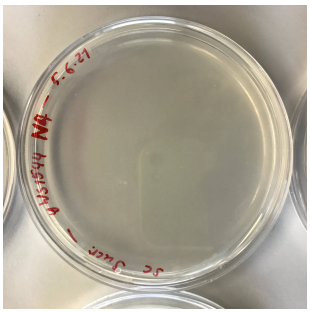
(a) Producer H5 transformed with pOG015.



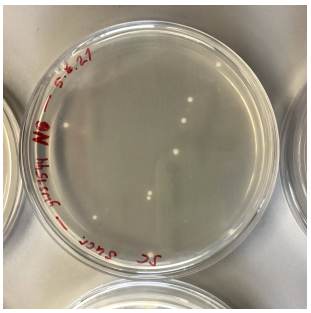
(b) Producer H5 transformed with pOG017.



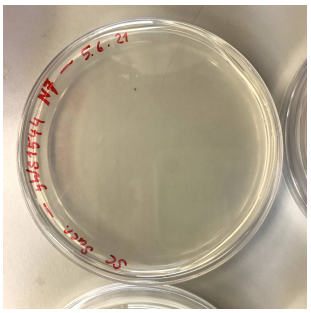
(c) Producer H4 transformed with pOG015.



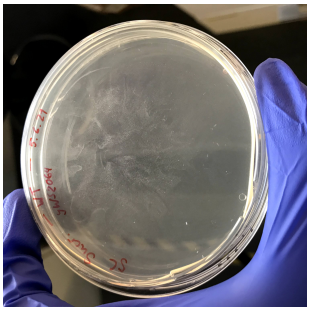
(d) Sensor yWS1544 transformed with pOG015, colony 4.



(e) Sensor yWS1544 transformed with pOG015, colony 6.

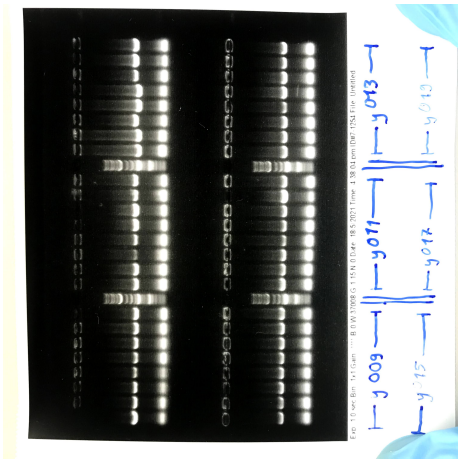


(f) Sensor yWS1544 transformed with pOG015, colony 7.

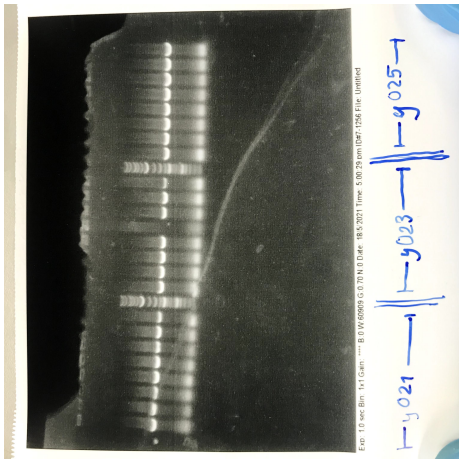


(g) WT yeast.

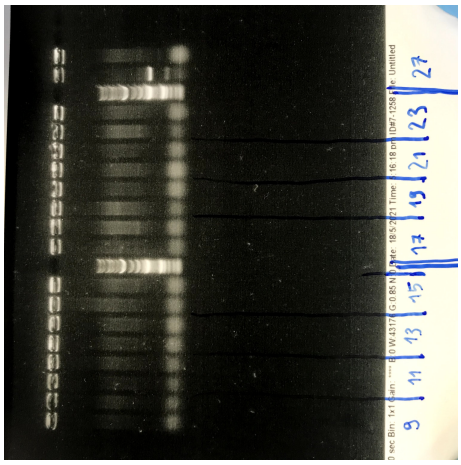
Figure 10: Verification of SUC2 knockout via plating on sucrose-only plates.



(a) pOG004, pOG005, pOG006, pOG007, pOG008, pOG009.

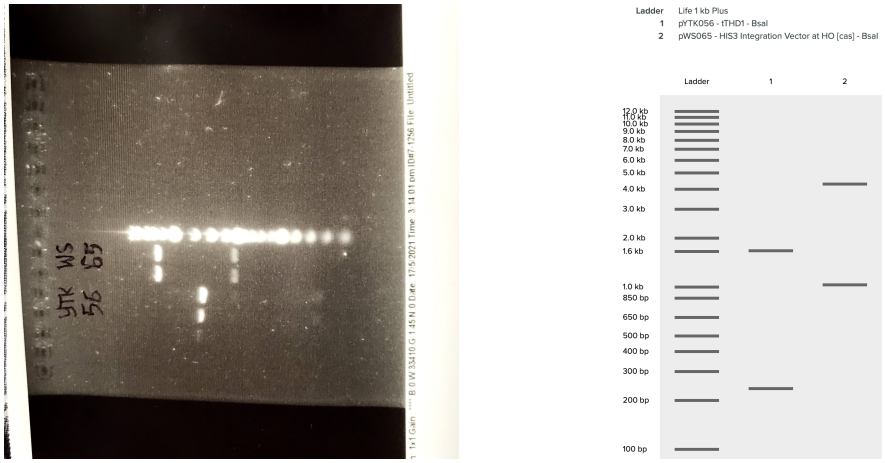


(b) pOG011, pOG012, pOG013.



(c) Control colonies for all assemblies.

Figure 11: Gel image of colony PCR for the construction of a library of YTK promoters for SUC2. Length of total plasmid insert should be ca. 2800bp, while 1200bp band appears on the majority of samples. 8 colonies picked for each assembly and 2 colonies for each control.



(a) Gel image pYTK056 (labelled YTK 56) and pWS065 (labelled WS 65), 1kb ladder on right.

(b) Virtual digest (Benchling) shows pYTK056 in column 1 and pWS065 in column 2, with 1kb ladder on left.

Figure 12: BsaI digest of pYTK056 (tTHD1) and pWS065 (His integration vector) match their virtual digests.

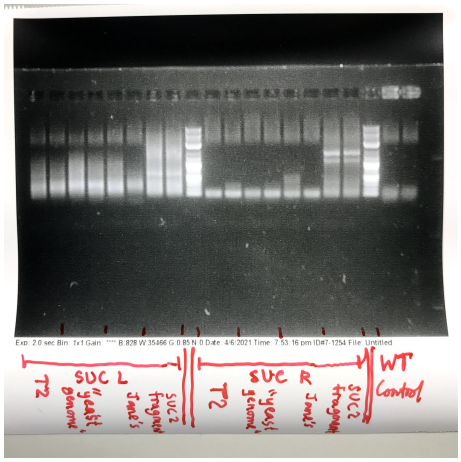


Figure 13: Gel image of SUC2 left (SUC2 L) and right (SUC2 R) fragments after PCR amplification with 3 different WT yeast genome isolations and a SUC2 fragment template.

3.4 Melatonin sensing

3.4.1 Flow cytometry with SUC2 Knock-out strain

The melatonin-sensitivity of the sensor strain with the SUC2 gene knock-out (colony N4) was parameterised by growing with melatonin at a concentration of log[Melatonin] from -3 to -10. Flow cytometry then measured the GFP output at these different concentrations (data shown in Fig. 14. Compared to the unmodified sensor strain, the GFP fluorescence of the knock-out showed a similar Hill slope but shifted left towards lower melatonin concentrations.

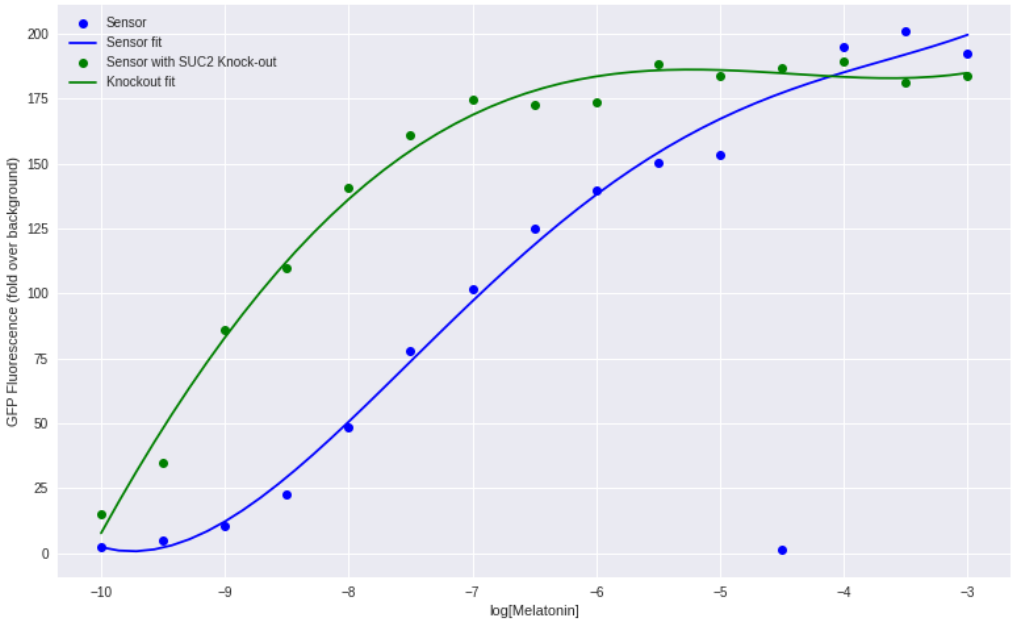
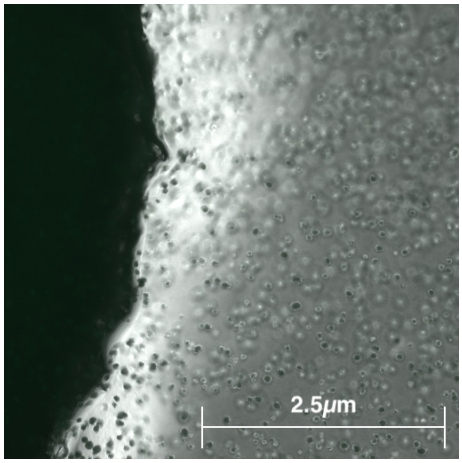


Figure 14: GFP fluorescence of yeast sensor strains baseline (yWS1544) and knock-out (yWS1544 N4) grown for 4 hrs in shaking conditions. Sensor with SUC2 knock-out shows heightened sensitivity to melatonin. Outlier shown at log[Melatonin]=-4.5 for unmodified yWS1544 sensor strain.

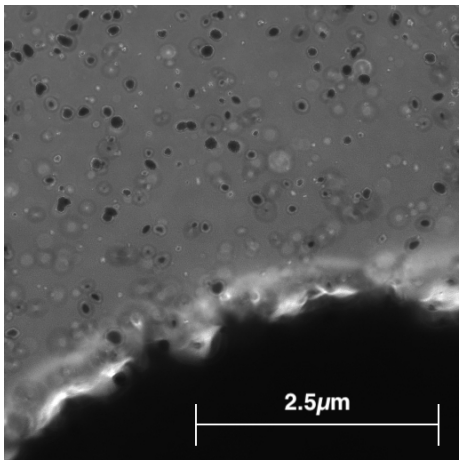
3.4.2 Halo assay

For the halo assay, the plates did not show fluorescence under blue light after 3 days of growth, so they were checked under the microscope for fluorescence. At high OD's such as 0.5 and 0.1, there was no significant fluorescence (Fig. 15), but lower OD's such as 0.01 showed fluorescence. Colonies have grown into a moon-like shape in the embedded media.



(a) White channel, sensors at OD 0.1, paper blot left.

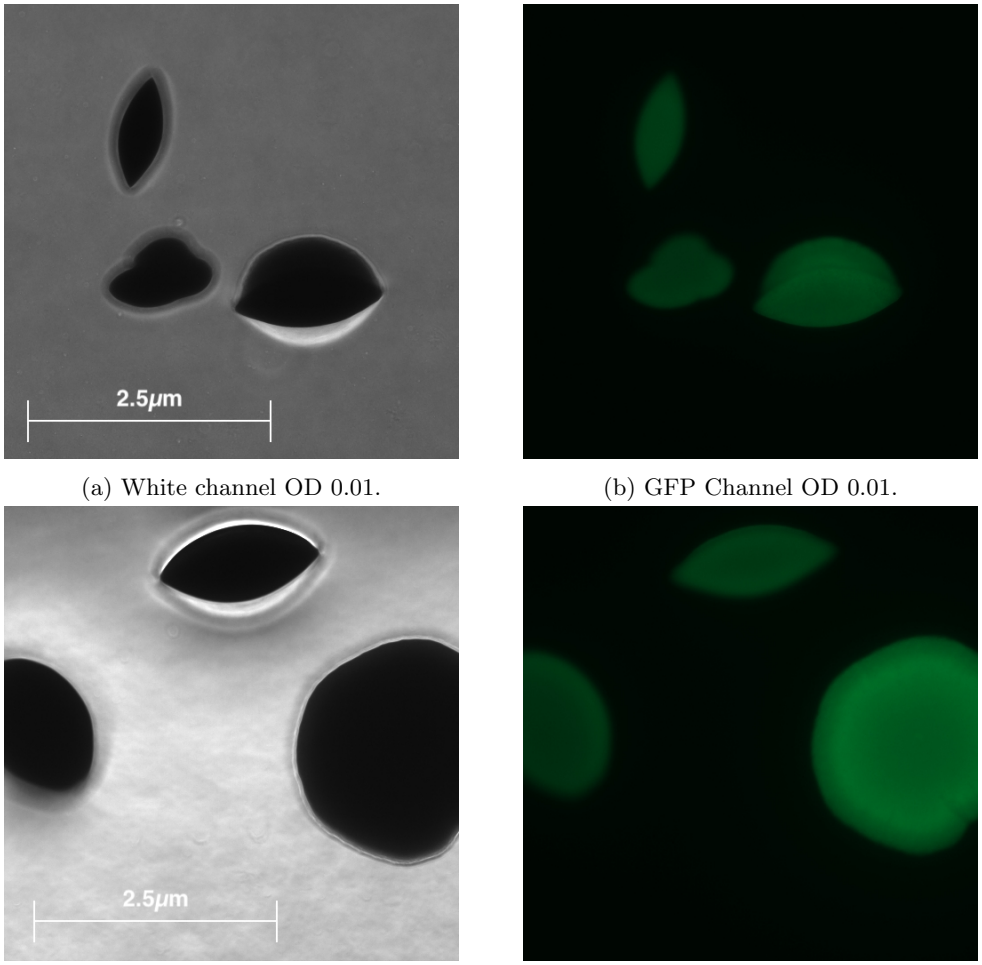
(b) GFP channel, sensors at OD 0.1, paper blot left.



(c) White channel, sensors at OD 0.05, paper blot bottom.

(d) GFP channel, sensors at OD 0.05, paper blot bottom.

Figure 15: Halo assay: Yeast sensor yWS1544 embedded in agar at varying optical densities imaged under fluorescence microscope only show autofluorescence in the optical densities OD 0.5 and OD 0.1.



(a) White channel OD 0.01.

(b) GFP Channel OD 0.01.

(c) White channel OD 0.01.

(d) GFP Channel OD 0.01.

Figure 16: Halo assay: Yeast sensor yWS1544 embedded in agar at varying optical densities imaged under fluorescence microscope show some fluorescence at OD 0.01. at magnification of 5μm.

3.5 Co-growth in liquid culture

To test that the sensor strain responds to the melatonin from the producers, the sensor and producer strain were grown together in liquid culture, with the producer strain introduced at different starting optical densities. The strains were grown overnight then cultured in 2ml SC liquid media (His, Trp) in culture tubes for 4 hours in shaking conditions prior to measurement. The sensor strains and WT control were grown at a starting OD of 0.1, while the producer strain OD was varied between [0.2, 0.1, 0.01, 0.001, 0.0001]. The GFP fluorescence levels from the producer and sensor strains being grown together are shown in Fig. 17. The WT control strain was only grown with the producer of medium strength, H5.

At the simplest level, the unmodified sensor strain yWS1544, picked from the working glycerol stock, was grown together with the three producer strains H2, H4, and H5. Additionally, a duplicate of the yWS1544 strain from the red frozen glycerol stocks was picked, labelled yWS1544 R, as well as with one of the yWS1544 sensor strains from the knock-out experiments (labelled yWS1544 N4). These additional sensor strains were only grown together with medium producer strain H5. Overall, the highest fluorescence response was achieved with the H5 producer grown with each of the working stock. Very similar responses were achieved with the co-grown H5 producer and the two other sensor strains.

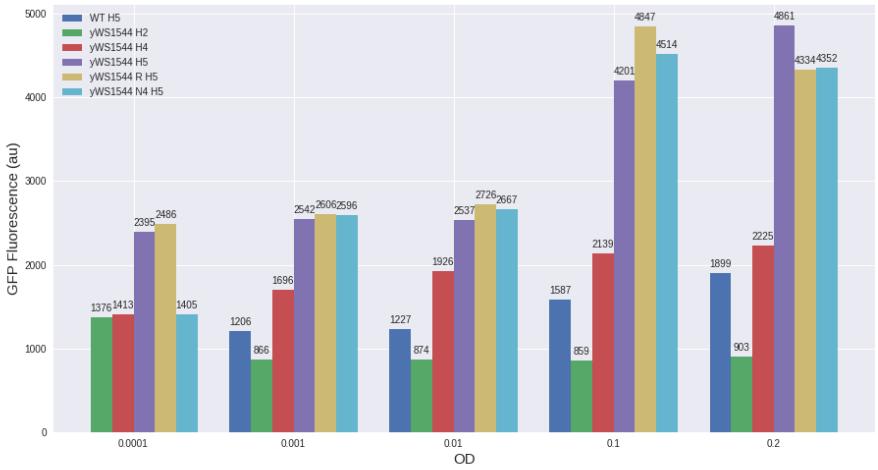
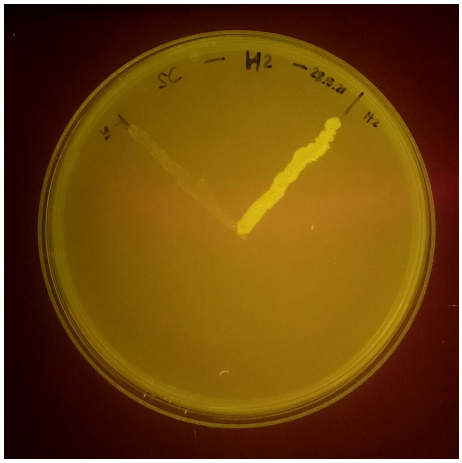


Figure 17: Flow cytometry GFP fluorescence (au) from the co-growth of sensor and producer strains in liquid culture at starting optical densities of producers at [0.2, 0.1, 0.01, 0.001, 0.0001], with sensors and WT kept at a constant OD of 0.1.

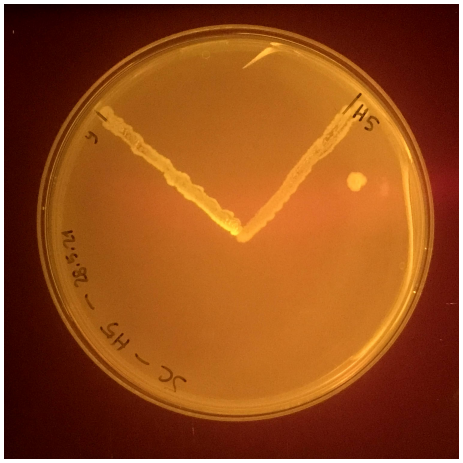
3.6 Co-growth on plates

3.6.1 V-shape plate assay

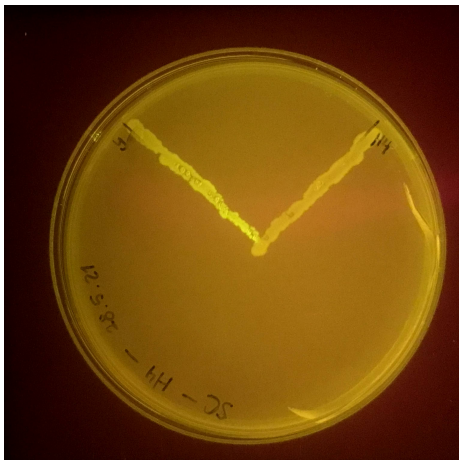
Two repetitions of the V-shape plate assay were done. The first repetition was imaged on blue light after 3 days of growth to show no quantifiable signs of fluorescence. The plates were imaged on blue light again after 10 days and showed significant fluorescence in the streak with the sensor strain, shown in Fig. 18. The highest producer, H4, in particular showed bright fluorescence that dimmed after about 75% of the length of the streak (Fig. 18c). The medium producer H5 showed mild fluorescence dimming about 20% down the length (Fig. 18b), and the plates with the lowest producer H2 did not show any fluorescence (Fig. 18a).



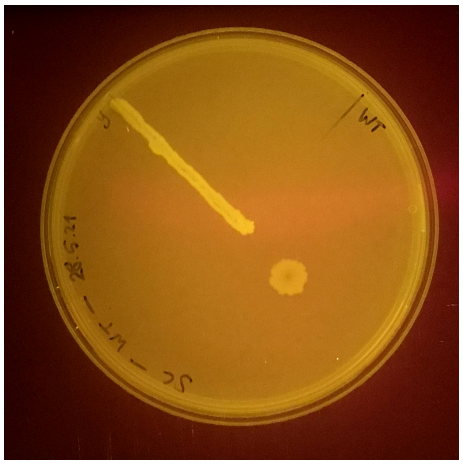
(a) Melatonin producer H2 (low) bottom, yWS1544 top. No fluorescence.



(b) Melatonin producer H5 (medium) bottom, yWS1544 top. Some fluorescence.



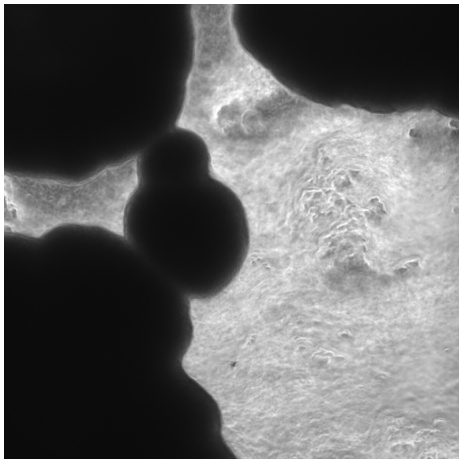
(c) Melatonin producer H4 (high) bottom, yWS1544 top. Significant fluorescence.



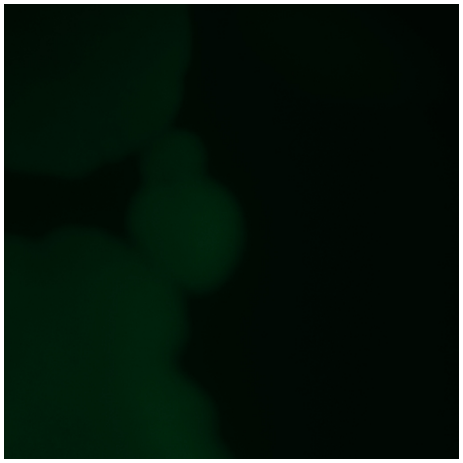
(d) Control WT yeast strain bottom, yWS1544 top.

Figure 18: Producer and sensor strains streaked onto SC Agar (His, Trp) in V-shape. Petri dishes imaged on blue light. Noticeable increase in GFP fluorescence at intersection of growth. First repetition.

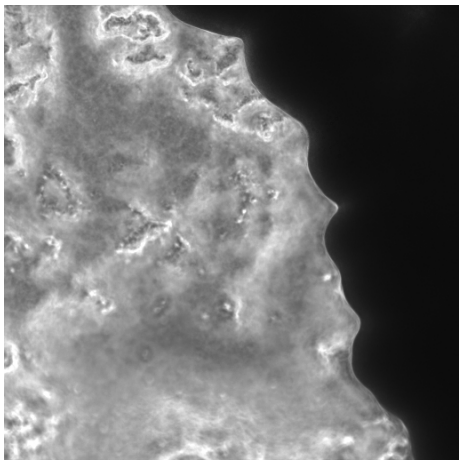
The second repetition of the V-shape plate cultures were imaged under a microscope, as it was too difficult to discern fluorescence just the blue light filter after 3 days of growth. In Fig. 19, the high producer H4 is streaked next to sensor yWS1544. The sensor strain fluoresces both at the intersection of the V streaks and at the edge of the streak line ca. 1cm away from the intersection.



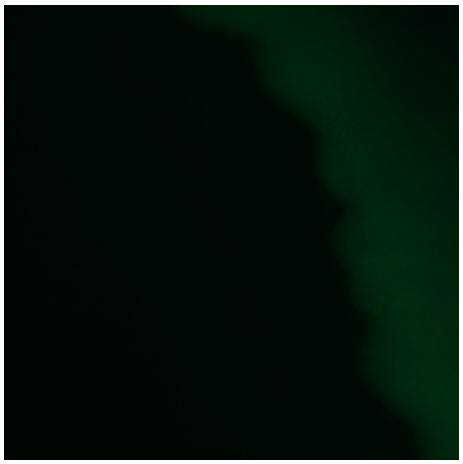
(a) White channel image of V-shape streak sensor yWS1544 (left half of image) and producer H4 (top right). Intersection point of V-shape.



(b) GFP channel equivalent to 19a.



(c) White channel image of V-shape streak sensor yWS1544 (top right of image). Left edge of V on the sensor streak.



(d) GFP channel equivalent to 19c.

Figure 19: Microscopy images of V-shape plates (SC Agar) with producer H4 and sensor strain yWS1544. The sensor strain shows fluorescence while the producer does not.

3.6.2 Mixing at different optical densities

To find a good starting density for the producer and sensor strains to be grown together at, each combination of strains was plated at a range of optical densities. The strains were diluted to OD 0.1 after overnight growth then a range of OD's generated through a further 5 serial 10x dilutions. After 2 days of growth, the plates had the appropriate relative density of growth, shown in 20.

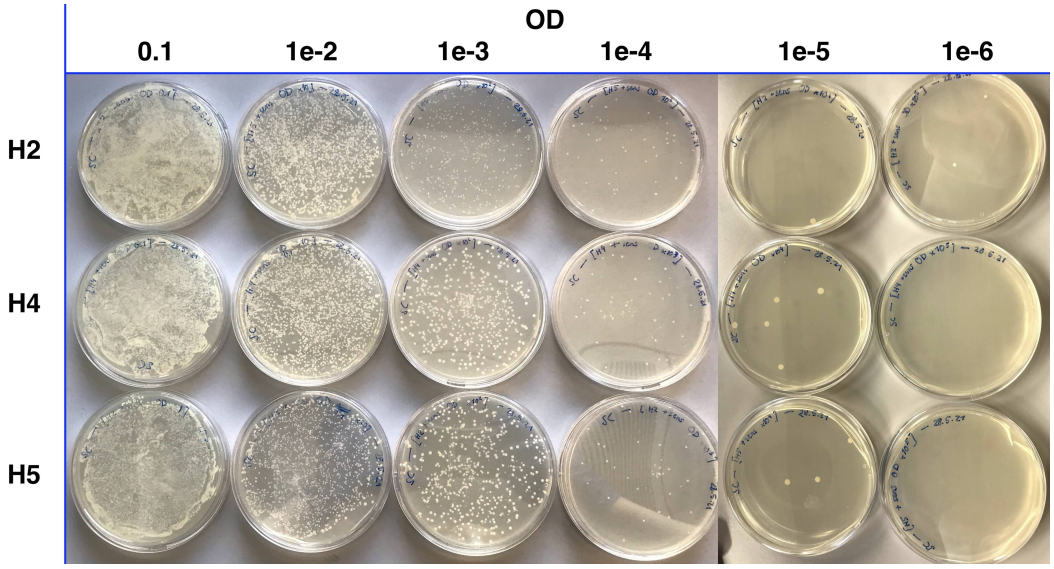
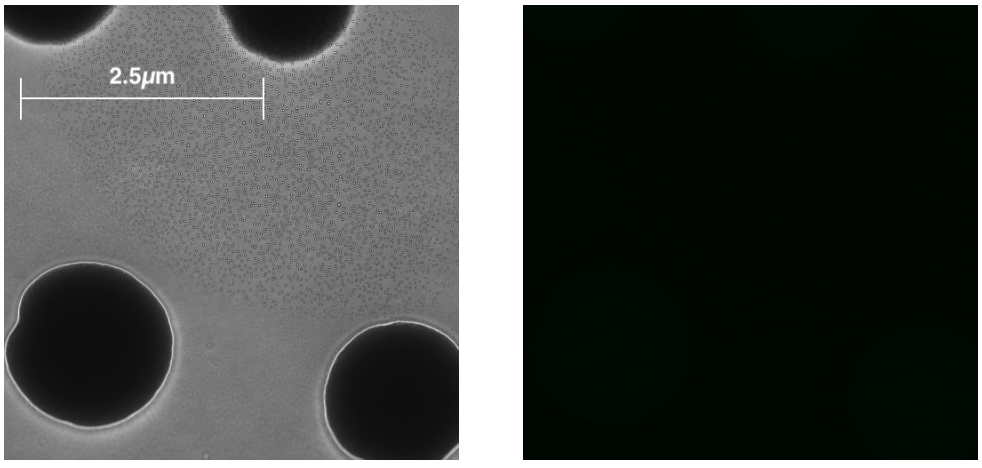


Figure 20: SC Agar (His, Trp) plates after 2 days in 30°C incubator with both strains mixed at different starting optical densities.

Unlike the co-cultures with V-shaped streaks, the cultures grown directly together did not show signs of fluorescence even after 10 days. The cells were imaged via fluorescence microscopy on the 10th day to check for a weak GFP signal, but none of the starting OD's produced a significant fluorescence signal that could be picked up by the microscope. Fig. 21 shows an example plate in the white and GFP channels.



(a) White channel.

(b) GFP channel.

Figure 21: Microscopy images of producer H5 mixed with sensor yWS1544 at starting optical density of 0.005 shows only very slight autofluorescence.

4 Discussion

Actual project SUC2 Knock-out Design Change

SUC2 Knock-out Design Modification

The high number of repetitions of the SUC2 knock-out yeast transformation suggested that there was something wrong with the initial CRISPR gRNA design. The deletion region included the SUC2 gene and 500bp downstream of the gene to include hanging terminator sequences. There

are two other genes 2000bp downstream of SUC2. The first CDS, YIL161W, is annotated on the SGD as a "putative protein of unknown function" and upon advice from experienced lab members on the design of CRISPR knock-outs was considered to be non-essential. The second gene, further downstream to SUC2 and YIL161W, is POT1 coding for "3-ketoacyl-CoA thiolase with broad chain length specificity". A gap of ca. 500bp was left between the POT1 gene and the end of the deletion region. However, the first ca. 250bp of the YIL161W gene were included with the deletion, which may have been the cause of difficulties with the knock-out. While both essential and non-essential genes should be susceptible to CRISPR edits and subsequent homology-directed repair (HDR) [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5951413/], as double-strand breaks are otherwise lethal to yeast and only employ non-homologous end-joining when HDR is blocked, previous experiences from other lab members support that there may be difficulties when other genes are within the donor regions. The terminator of YIL161W would remain on the genome and could cause interruptions of other gene mechanisms in POT1 or in the transcription of genes upstream of the deletion, such as YIL163C, a protein of unknown function.

In the modification of the knock-out design, the entire YIL163C gene was included in the deletion, while the entirety of the YIL161W gene was left un-deleted, leaving ca. 60bp of the tail end of the SUC2 gene remaining in the genome. As an alternative, a simple HR-directed gene deletion was also planned, whereby the upstream and downstream donor sequences were placed on a plasmid with antibiotic selection and transformed into yeast. Though the efficiency with this approach would be lower than with a double-strand break present, the CRISPR cassettes would not be required for this. Another approach would be to utilise the naturally occurring BsaI site within SUC2 as the cut site to further stimulate this repair mechanism, though the cut would not be a double-strand break. Thanks to positive results from the CRISPR-based knock-out design however, this approach was not necessary.

Efficiency of edits

While the knock-out worked for a number of samples, especially those with Nourseothricin as the selection marker, the efficiency of the knock-out was quite low and only one of the three producer strains and the sensor strain had the SUC2 gene successfully knocked out. An improvement over the second design may be to use the same gRNA targets, but lengthen upstream and downstream donors to 500bp rather than 50bp, as in the initial design, and carry these donors on a plasmid rather than as a gene fragment. The CRISPR targets could also be computationally evaluated for optimisation. Another reason for inefficiency would be the choice of selection markers, since results showed some natural increased resistance of the producer strains to recommended doses of zeocin. While the top-spreading method of applying antibiotic to plates should theoretically have higher efficiency than having the antibiotic mixed into the agar due to a more concentrated, toxic surface, inhomogeneities in this method may be the cause of some of the decreased selection levels, as recognisable from the ring-like edge colonies on the transformant plates in 8. Though efforts to parameterise the antibiotic response of the yeast strains used in this project were made, a more optimal concentration may increase the likelihood of successful transformations.

SUC2 Expression Characterisation

The plasmids for characterising the optimal expression levels of SUC2 (Table 18 unfortunately could not be assembled during the project for several reasons. Several attempts were made to assemble the plasmids with a SUC2 fragment that included a BsaI site, leading to highly error-prone Golden Gate assemblies. Though the BsaI site was not removed from the SUC2 fragment despite many trials of PCR amplification, as per the point mutation described in Section A.13. Though one or two of the colonies picked with the BsaI-containing SUC2 gene showed the right fragment length, sequencing results identified them as mis-assemblies. Impurities in the yeast genome or the SUC2 fragment templates may have caused low binding rates of the primers and thereby low amplification. The PCR reaction could be further optimised by adding different levels of DMSO and trying different polymerases or master mixes. The YTK overhang primers had a binding region of length 30 and 34 and overhang of length 23 and 26, as well as secondary structure homodimerisation energies of -5.31 kcal and -4.44 kcal at 55°C, none of which should hinder the binding of the primer to the gene. A primer re-design could still be attempted at the expense

of including (non-coding) regions of DNA outside of the gene. The gel purification of the DNA yielded low concentrations of each fragment, so other methods of isolating the PCR product could be used, such as running a gel with only a fraction of the total PCR product as validation then doing a PCR clean on the bulk of the PCR product. Isolating this gene effectively would then enable the construction of both the YTK promoter library characterising constitutive invertase expression and the melatonin-inducible invertase plasmids.

SUC2 Knock-out Melatonin Sensitivity

The Hill curve estimated previously in [11] (see Fig. 5) was replicated for the working sensor strain and achieved a similar 200x fold change over background GFP fluorescence, as shown in Fig. 14. The un-transformed sensor yWS1544 (blue) in this curve has a matching dynamic range to the previously determined curve, while the sensor strain with SUC2 knocked out is more sensitive to melatonin by ca. 1.5-2 orders of magnitude - the knock-out curve is shifted left of the original strain by ca. 2 log[Melatonin]. While further repeats would be needed to confirm the increase in sensitivity of the knock-out strain and improve the confidence of the estimated Hill curve, the reason could be reduced cell burden from lack of invertase production and therefore higher capacity for the production of GFP. If the lack of invertase is causing a decrease in the glucose content immediately surrounding the yeast, which is unlikely as the media used in this experiment had a final concentration of ca. 2% glucose, the increase in sensitivity of the melatonin GPCR could also stem from lower competition with the glucose-sensing GPCR signalling cascade, which shares several signalling molecules with the pheromone sensing pathway [32]. In any case, the melatonin-sensing ability of the knock-out strain was not hampered by the invertase gene knock-out thanks to the afore-mentioned flow cytometry data of exogenously added melatonin, as well as flow cytometry (FC) data from co-growth of the sensor and producer strains.

Co-growth of producer and sensor strains

The significantly increasing fluorescence levels in Fig. 17 upon higher ratios of producer to sensor cells confirm that the intercellular communication can result in a desired engineered response and that the rates of melatonin production are sufficient for sensing. While the producer strain H2 (low melatonin production) engenders a negligible increase in fluorescence as expected due to its melatonin production rates lying too low on the sensor activation curve (Fig. 14), the producer strain garnering the highest fluorescence levels is actually the medium-level producer H5 rather than the high-level producer H4. This could be due to culture or cell health conditions and would require further repeats to confirm. As the OD of the sensor was kept constant to vary the OD of the repeater strain, an optimum ratio of strains may be found by varying both parameters. This FC data also confirms that the working stock of sensor yWS1544 was not contaminated throughout the project, as a fresh sensor denoted yWS1544 R produced highly similar fluorescence levels. Further repetitions of this co-growth under different media and growth time conditions would provide confirmation of the stability of the growth environment and of the co-dependent behavior of the two strains, but initial flow cytometry data confirms that the melatonin sensing and producing range is compatible between the sensor and producers.

Co-growth of producer and sensor strains on plates

Halo assay

While confirmation of melatonin-GPCR activation by the producer in liquid culture bodes well for the reward mechanism, conditions on plates bring different challenges. The diffusion of melatonin to be measured via the halo experiment did not result in easily noticeable fluorescence despite the high melatonin concentration used to coat the blotting paper except in a few plates imaged via fluorescence microscopy. The two highest optical densities of OD 0.1 and 0.05, which should have shown the highest fluorescence signal, did not show anything beyond autofluorescence under the microscope (Fig. 15). This may be due to the rigidity of the blotting paper used, resulting in decreased melatonin diffusivity to the agar when the melatonin was concentrated too much on the top surface of the blotting paper rather than permeating the paper and thus gaining access to the agar below. The melatonin may also have only diffused across the surface of the agar rather than penetrating into the agar that the yeast cells were embedded in. The optical density of 0.01 had

larger colonies than other plates shaped like crescents, a very unnatural shape which is most likely due to the constraints placed on the cells by the surrounding rigid agar media they are embedded in. This may also be due to contamination, but this is unlikely as there would be no fluorescence signal from natural contaminant bacteria. Due to the low cell density of the plates with fluorescent colonies, the diffusion distance of melatonin across the plate could not be quantified.

Plating at different starting densities

Despite flow cytometry data confirming the cells have compatible ranges of melatonin sensing and production in liquid culture, there was no identifiable fluorescence on plates where the sensor and producer were mixed together in water then streaked together. Even high starting concentrations of OD 0.1 for both strains did not appear to fluoresce under the miscroscope or blue light even after 10 days of growth (Fig. 21 shows OD 0.005 as example). While unlikely, this could be due to contamination of the sensor strain used in this experiment, but is most likely due to insufficient levels of melatonin on the solid media to significantly activate the sensor strain, which responds non-linearly to sensed melatonin. Plating the cells with water may have washed the cells as well of melatonin from overnight growth. Improving the diffusion of melatonin through a surface coating (eg by adding water in time intervals) or, more practically, trialing higher optical densities could address the issue, especially in light of the results obtained from the V-shape co-streaking of cells.

V-shape co-growth

Streaking the sensor and producer strains directly from overnight cultures into a V-shape showed that melatonin production from sensor cells can sufficiently activate sensor cells to produce a response on solid media. The microscopy images from Fig. 19 reinforce the results from plain blue light in Fig. 18, as plates show the sensor streak with noticeable levels of fluorescence diffusing a significant way down the streak. Microscopy image Fig. 19b especially shows the non-fluorescent producer strain directly next to the weakly fluorescing sensor strain. The main takeaways from this experiment are that the co-growth on plates is fairly compatible between the strains at the right optical densities and that melatonin diffusion is sufficient to reach sensor strains if sufficient rates of production are achieved. This last point highlights the advantage offered by the the non-linear response of the sensor strain, as this presents an additional layer of selectivity for high-producing strains.

One central challenge remaining is the parameterisation of the reward and how well the producer strains respond to it. Modelling results show that after sufficient time, the overall final cell count for each strain corresponds in hierarchy to the level of melatonin production of each strain, despite the probability of division in areas with no glucose set to zero. This indicates that the chicken and egg problem of which strain activates the other first is able to be overcome. In practice, one way to achieve an initialisation of the reward cascade would be to plate producer strains without washing them of their residual secreted melatonin. There may also be an optimum amount of time after which to pick colonies for finding the highest producer, as diffusion of invertase and melatonin across the plate serve to de-localise the reward effect and confound the growth rate of the different producers. As seen from the V-shape plates however (Fig. 18), the activation of the sensors can be highly localised depending on the production rate and the length of growth time.

5 Conclusion

In general, the results found in this project support the idea that based on localisation, one strain can select for specific behavior from a library of strains. However, the extra work involved in cloning the sensor and producer strains before the reward may be activated is not practical for high-throughput screening - a reward agnostic of the pre-existing features in the producers would be more versatile and could theoretically serve as a generalised screening tool, ideally only requiring knowledge of its sensor's dynamic range. A more generalisable rewarding-strain would also turn the necessary but in this case system-specific parameterisations of growth conditions into an endeavour yielding dividends for screening a variety of microbes. Ultimately the role of the sensor was akin to a computational module in a system performing what could be seen as a

sort function or a maximisation and effectively taking the place of an entire machine specialised in optimising efficiency, showing how the tools synthetic biology uses have matured to the scale of entire plug-and-play organisms and how much potential even simple synthetic organisms have when used in a targeted way.

A Wet Lab Methods

A.1 DNA Clean Concentrator PCR Wash

From DNA Clean Concentrator PCR Wash Zymo. To be performed after purification of PCR product.

Change centrifuge time to 1min and 13000 bpm instead of just 30s After a PCR you should be starting with 100ul Add 50ul water if not DNA Elution buffer

- For PCR fragment: In a 1.5mL microcentrifuge tube, add 5 volumes of DNA Binding Buffer to each volume of DNA sample (eg 500μL Buffer to 100μL DNA). Mix briefly by vortexing.
- Transfer mixture to Zymo-Spin Column in a Collection Tube.
- Wash by adding 200μL DNA Wash Buffer to the column Centrifuge for 1 minute at 13000rpm. Repeat this wash step.
- Add min. 6μL DNA Elution buffer or nuclease-free water directly to the column matrix and incubate at room temperature for ca. 1 minute. Transfer the column to a clean 1.5mL microcentrifuge tube and centrifuge for 30s-1min to elute the DNA.

A.2 Gel extraction and purification

A gel extraction was performed to isolate short (500bp) fragments of DNA.

Procedure (based on Zymo DNA Gel extraction and purification)

- Excise the DNA fragment from the gel using a razor blade and transfer to a 1.5mL microcentrifuge tube.
- Add 3 volumes of Agarose Dissolving Buffer (ADB) to each volume of gel sliced.
- Incubate at 65°C for 10-15 minutes until gel is melted (in heat block or otherwise).
- Transfer melted agarose to Zymo-Spin Column in a Collection Tube.
- Centrifuge for 30-60s and discard the flow-through.
- Add 200μL of DNA Wash Buffer to the column and centrifuge for 30s. Discard the flow-through. Repeat this wash step.
- Add min. 6μL DNA Elution buffer or nuclease-free water directly to the column matrix and incubate at room temperature for ca. 1 minute. Transfer the column to a clean 1.5mL microcentrifuge tube and centrifuge for 30s-1min to elute the DNA.

A.3 Gel preparation and run

All gels were prepared using either 1% or 0.8% agarose gel media. Typical run settings were 20-30 minutes at 3.00A and 120V.

A.4 Golden Gate assembly

Name	Amount	Concentration
BsmBI (or BsaI)	0.50μL	
T4 ligase	0.50μL	
Backbone	0.25μL	25 fmol
Fragment 1	0.50μL	50 fmol
Fragment 2	0.50μL	50 fmol
GG buffer	1 μL	
MQ (Water)	6.75μL (To make 10 μL)	
Total	10 μL	

Table 3: Recipe for Golden Gate protocol. Restriction enzyme of choice can be used.

A.5 Flow Cytometry

Flow cytometry data was collected with a Life Technologies flow cytometer in a 96-well plate with 200ul volume of each sample. Cells were first grown overnight in liquid shaking culture at 250rpm then re-cultured at 1:100 cell to liquid media for 4-6 hours. SC media containing histidine and tryptophan was used for higher selection of yeast strains. The geometric mean was then calculated using the flow cytometry software FloJo [33]. To quantify the GFP signal of each strain in response to externally added melatonin, the GFP curve was used raw to get the geometric mean. In the case of the sensor and producer strains growing together however, the fluorescence from both strains was merged, so to find the geometric mean the fluorescence signal was first gated at the point where the fluorescence from the control strains tailed off to zero.

A.6 Halo Assay

The halo assay was performed similarly to [34] and as previously performed in the lab by Will Shaw.

Preparation

- SC plates (His, Trp) (room temperature)
- 1.5% agar melted and then brought to 50°C in water bath
- 1x SC media (room temperature)
- 2x SC media (room temperature)
- Punch out discs of blotting paper with hole punch. Sterilise with ethanol and dry. Add melatonin to disc and dry (0.2M was used).

Assay

- Back dilute overnight culture 1:100 into fresh SC media and grow to OD 0.6 (takes ca. 4 hours). Check with OD reader.
- Pellet cells and resuspend at 2x the desired OD in fresh 2x SC complete media.
- Do next steps quickly as agar will set
- Add cell suspension 50/50 with 1.5% agar (kept at 50 °C in water bath) in a 50 mL falcon and vortex.
- Pour 5 mL evenly over a SC plate (use serological pipette for precision and ease).
- Once set, place paper disc with ligand on to top of agar, ideally in the center of the plate.
- Incubate at 30°C overnight.
- Image plate on fluorescence scanner (blue light or fluorescent microscope were used).

A.7 Heat Shock Transformation E. coli

A heat shock was performed to insert a Golden gate assembly or other plasmid into competent E. coli cells.

- Get competent cells out of -80°C fridge on ice.
- Add 50ul KCM to competent 200ul E. coli cells.
- Keep on ice.
- Wait 3-5mins.
- Total competent cell volume + KCM = 250ul
- Pipette the desired volume of cells per Golden Gate reaction tube (eg 125ul for 2 tubes) into PCR tubes with plasmids or PCR product.
- Put into RC Heat shock protocol in PCR machine. Run program: 10 minutes at 4°C, 1 minute at 42°C, 10 minutes at 4°C, 1-2hrs at 37°C.
- Wait until machines cools to 4°C before putting in samples (after run has started).

A.8 Media and Plate Preparation

SC Media and Agar All SC media and agar was prepared with supplemented Histidine and Tryptophan.

Synthetic Complete (SC) Media	**1L**	**500mL**
Bact-yeast nitrogen base without amino acids*	6.7 g	3.35 g
L-Tryptophan 1g/100ml	2mL	1mL
L-Histidine 1g/100ml	2mL	1mL
Uracil 2000mg/L	10ML	5mL
L-Leucine 1g/100ml	12.5ml	6.25mL
Agar (Optional)	20g	10g
Cautiondrop-out mix	1.4 g	0.7g
H2O	to 900 ml	to 450 ml
Glucose	100ml (20 g/100mL)	50ml

Table 4: SC media and agar recipe. All materials from Sigma Aldrich or Thermo Fischer.

YPD

Compound	**Amount**	**For 500ml**	**For 50ml**	**Label**
Bacto-yeast extract	10g	5g	0.5g	Fermtech Yeast extract
Bacto-peptone	20g	10g	1g	Fermtech Peptone (casein)
Glucose	20g	10g	1g	D-Glucose (anhydrous)
Distilled H2O	1000ml	500ml	50ml	

Table 5: YPD media preparation recipe. All materials from Sigma Aldrich or Thermo Fischer.

A.9 Microscopy

The ZEISS Microscopy Zen software was used to capture images and process data into jpeg format. The microscope used was a ZEISS cell microscope with standard white, GFP and mScarlet imaging channels. An exposure of 130fps was used for the GFP channel and an automatic exposure in white light. All images were taken at a magnification of 5μm.

A.10 PCR

Colony PCR

Typically 8 colonies were picked per plate, as well as 2 control colonies that were run in the PCR mix without primers as background control.

Gene Fragment Amplification PCR

For the amplification of gene fragments, the same PCR recipe was used as for colony PCR, except that 25μL of Polymerase Mix was used instead of 5μL per reaction.

Name	Volume (ul)	10x Volume (Master mix)
Polymerase Mix (eg 2x Phire Plant)	5	50
Primer Forward	2	20 (2ul + 18ul H2O)
Primer Reverse	2	20 (2ul + 18ul H2O)
Plasmid / Colony	1	/
Enzyme temperature (eg 37°C for BsmbI)	Overnight	

Table 6: Colony PCR 2x Phire Plant recipe.

Standard PCR Settings

Stage 1 (1x): 98°C for 30s

Stage 2 (30x):

- 98°C for 15s
- Annealing temperature (usually 55°C) for 30s
- 72°C for Extension time (2kb/min for Phire plant mix)

Stage 3 (1x): 72°C for 5mins followed by 4°C until samples are taken out.

A.11 Plating cells

Plating cells

Petri dishes were usually prepared the day of or the day before cell transformations. Agar media was heated in a microwave and added to petri dishes in volumes between 12-20mL. The plates were left to cool for at least 30 minutes prior to plating cells. Common E. coli antibiotics such as Chloramphenicol (Cam), Spectinomycin (Spec) and Kanamycin (Kan) were mixed into LB media in a 1:1000 ratio to total volume, while yeast antibiotics (Zeocin, Hygramycin, Nourseothricin) were top-streaked onto the directly before or during streaking cells onto the plate. The following concentrations and volumes of each antibiotic were used to streak the plate (assuming 20mL agar volume):

- **Zeocin** (in stock as 100mg/mL): 200μg/mL final concentration. 40uL of stock streaked.
- **Hygramycin** (in stock as 50mg/mL): 200μg/mL final concentration. 80uL of stock streaked.
- **Nourseothricin** (in stock as 500mg/mL): 100μg/mL final concentration. 40uL of stock streaked.

A.12 Yeast Transformation

The transformation procedure used in this project was the lithium acetate preparation yeast transformation protocol.

Materials

- 50% PEG-3350

- 1M LiOAc
- 0.1M LiOAc
- Salmon Sperm DNA (10 mg/mL)
- ddH_2O

Reagent	Number of transformations
PEG-LiOAc mix	Volume per transformation (μL)
PEG-3350 (50% w/v)	260
LiOAc 1.0 M	36
Total	**296**
DNA cocktail	Per transformation (μL)
Boiled ssDNA (10 mg/mL)	10
Plasmid DNA + ddH2O	54
Total	**64**

Table 7: Lithium acetate mix preparation to be made right before use for optimal results.

Procedure: Competent yeast cells

- A colony was picked from a plate or from frozen glycerol stocks, mixed into 3ml growth media (SC media or YPD) in 15ml culture tubes and put into a 30°C shaking incubator at 250rpm.
- Yeast grown up to OD600 0.75 shaking at 30°C (Takes 4-5 hours for WT in YPD) - 15mL growth media in 50mL Falcon tubes.
- Pellet at 2000 rcf for 10 min in large centrifuge.
- Prepare PEG-LiOAc mix while harvesting cells (see Table 7).
- Wash with 0.5x volume 100 mM LiOAc and repeat pellet step.
- Final suspension in 100 μL/transformation 100 mM LiOAc - typically get 6 transformations from staring 15 mL culture. For higher efficiency cells were split only into 3-4 transformations in initial experiments, then into 6 transformations later in the project.
- Aliquot 100 μL of cells into individual 1.5 mL tubes and pellet at 8000 rpm for 1 min at 20°C on small tabletop centrifuge and remove supernatant.
- Competent yeast cells can at this point be frozen after mixing with 30% glycerol stock (15% final concentration) or stored in the -4°C fridge for 3-5 days.

Procedure: Heat Shock

- Resuspend cell pellet with 64 μL of DNA cocktail.
- Mix cell suspension with 296 μL of PEG-LiOAc mixture.
- Incubate for 40 mins at 42°C.
- Centrifuge 8000 rpm for 1 min and remove supernatant.
- Depending on the selection marker: [Prototrophic/auxotrophy gene] Resuspend in 200μL of H_2O. [Eukaryotic antibiotic] Recover in 1.0 mL YPD for 2-3 hrs at 30°C.
- Plate desired volume on plates. Typically do 100ul.

A.13 Point mutation

To amplify SUC2 from the yeast genome and make it compatible with the YTK standard, a BsaI site in the middle of the gene had to be removed. A point mutation was added on to the

primers for amplifying the gene in two parts (see B.5 for primers). Initially, a PCR amplification reaction was prepared with the YTK-overhang and point mutation primers as pairs (paired up via upstream and downstream), then the entire product from this was run through a gel to purify the fragments, which were 1100bp and 500bp long. These fragments were then used with the YTK-overhang primers to amplify the full length of the SUC2 gene from the left and right SUC2 fragments.

A.14 Optical Density Measurements

Optical density (OD) measurements were taken at 600nm wavelengths and blanked beforehand. When samples were scarce, a sample was mixed 1:1 with blank media to make a 1ml cuvette sample.

A.15 Restriction digest

The following recipe was used for preparing overnight restriction digests (8 hours at digest temperature, then 10-20 minutes at 80°C):

Name	**Volume**	**11x**
Plasmid (eg pYTK001 (50fmol))	3ul	/
Buffer (eg 3.1 Buffer for BsmbI)	1ul	11ul
Restriction Enzyme (eg BsmbI)	0.3ul	3.3ul
H2O	5.7ul	62.7ul
Enzyme temperature (eg 37°C for BsmbI)	Overnight	

Table 8: Recipe for enzyme digest preparation.

The enzymes, temperatures and the buffers used with them are summarised here:

BsmbI

- Buffer: 3.1 Buffer
- Digest temperature: 37°C

NotI-HF

- Buffer: Cutsmart Buffer
- Digest temperature: 55°C

BsaI

- Buffer: Cutsmart Buffer
- Digest temperature: 37°C

A.16 Sequencing

For sending samples to sequencing, the companies Source Bioscience or Eurofins were used. The EMBOSS bioinformatics sequencing alignment calculator was then used to analyse accuracy.

A.17 V-shape co-streaking

The sensor and producer strains were streaked together in a V-shape onto the plate to measure the distance across which the sensor could be activated by the producer strains. For this assay, yeast strains were grown overnight in YPD or SC media. SC agar (His, Trp) petri dishes were prepared the day before or the day of. The streak lines were first drawn in equal lengths onto the bottom of the plate with ethanol-erasable marker, at a 90° angle in the first repetition and at a ca. 30°

angle in the second repetition. Cells were then streaked from the tip of the V shape towards the intersection point to prevent double-streaking one strain into the other.

A.18 YTK Part design and assembly

The Yeast Toolkit [31] is a modular, hierarchical framework for multi-gene assemblies and was used to assemble plasmids and design parts. The structure of the toolkit is summarised in Fig. 22. Individual gene fragments were mostly stored in an entry-level YTK vector for ease of use.

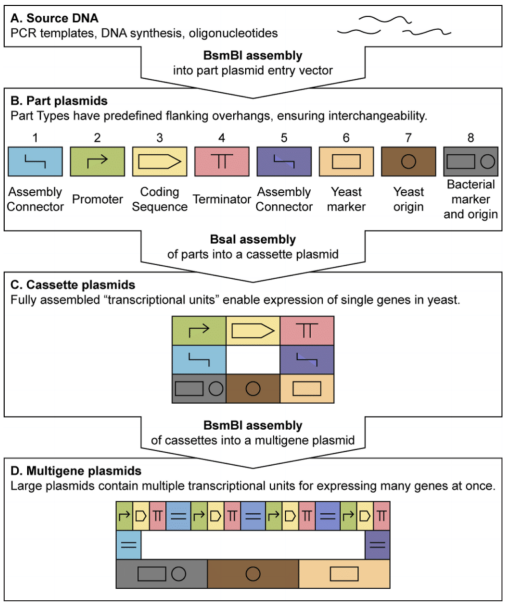


Figure 22: Reproduction of Figure 1 from [31] showing structure of the Yeast Toolkit (YTK).

B Primers

B.1 Primers for SUC2 Knock-out - first gRNA design

Label	Primer Sequence	Target
OG001	CAGTTGACCCTTTCTCATTCCCT	gRNA sequencing
OG002	TTTGGTCTCCCGCAGCTTTCCTTTTCCTT TTGGCGTTTTAGAGCTAGAAATAGCAAGTT	gRNA Up
OG003	TTTGGTCTCCTTTCGCTTGAGTGCGCAAG CCCGGAATCG	gRNA Up
OG004	TTTGGTCTCGGAAAAACAGAAGGGTTTTA GAGCTAGAAATAGCAAGTTAAAATAAGGC	gRNA Dn
OG005	TTTGGTCTCGGGATTGCGCAAGCCCGGAATCG	gRNA Dn
OG006	CCGCAGCAAAACGGTCGAATGT	gRNA sequencing
OG007	cgagggagcttccaggggaaa	Donor sequencing
OG008	ATGAGCCGTCTCCtcggGCGGCCGCTCCCC GCATTTTTATTACTCT	Donor Up
OG009	ATGAGCCGTCTCCTCCCATATACGTTAGTG AAAAGAAAAGC	Donor Up
OG010	ATGAGCCGTCTCGGGGAGCTAATAAAAAG AATACCCTACACTACTCA	Donor Dn
OG011	ATGAGCCGTCTCGggtcGCGGCCGCTTTTAA TACTTGATAATAGTTAATATTCTCCCTT	Donor Dn
OG012	gcagggcggggcgtaatttgat	Donor sequencing
OG101	CGCTCCCCCAGCAAAGCTCAAA	SUC2 sequencing
OG102	CCAAGAGGTGGTGCAATCGCGT	SUC2 sequencing

Table 9: Primers used in assembling the first design for the knock-out gRNA and donor targets.

B.2 Primers for SUC2 Knock-out - second gRNA design

Label	Source plasmid	Sequence
OG301	pOG017	TCCGGGCTTGCGCATTCCAAGCTAAAAAGT TTGAGTTTTAGAGCTAGAAATAGCAAGT
OG302	pOG017	CGATTCCGGGCTTGCGCAGGATCTGTGAAC ATGACCACGT
OG303	pOG017	CAGGATCTGTGAACATGACCACGTTTTAGA GCTAGAAATAGCAAGT
OG304	pOG017	CGATTCCGGGCTTGCGCAATCCactagtgcac
OG305	pOG016	TTTGGTCTCCCGCATTCCAAGCTAAAAAGTT TGAGTTTTAGAGCTAGAAATAGCAAGT
OG306	pOG016	TTTGGTCTCCGTTCACAGATCCTGCGCAAG CCCGGAATCG
OG307	pOG016	TTTGGTCTCGGAACATGACCACGTTTTAGA GCTAGAAATAGCAAGT
OG308	pOG016	TTTGGTCTCGGGATTGCGCAAGCCCGGAATCG
OG309	pOG015	TTTGGTCTCCCGCATTCCAAGCTAAAAAG TTTGAGTTTTAGAGCTAGAAATAGCAAGT
OG310	pOG015	TTTGGTCTCCGTTCACAGATCCTGCGCAA GCCCGGAATCG
OG311	pOG015	TTTGGTCTCCGAACATGACCACGTTTTAGA GCTAGAAATAGCAAGT
OG312	pOG015	TTTGGTCTCCGGATTGCGCAAGCCCGGAATCG
OG313	Donor Up 2	TGTAAAAATTCAGAATTCGCAACAACCTAT AATTGAGTTAAGTGCCTTTCCACTGGTGTC
OG314	Donor Dn 2	CTTCCCTTACTTGGAACTTGTCAATGTAGAA CAAATTATCGACACCAGTGGAAAGGCACT
OG315	Sequencing	CCGAATAATCTACGAAATATCATGGAGCTTCAGG

Table 10: Primers used in the redesign of the SUC2 knock-out.

B.3 Primers for SUC2 Knock-in Characterisation

Label	Source plasmid	Sequence
OG210C	SUC2 FWD C	TTTGGTCTCGtatgATGCTTTTGCA AGCTTTCCTTTTCCTTTTG
OG210G	SUC2 FWD G	TTTGGTCTCCtatgATGCTTTTGCA AGCTTTCCTTTTCCTTTTG
OG211	SUC2 REV 1	TTTGGTCTCGggatCTATTTTACTT CCCTTACTTGGAACTTGTCAATG
OG212	Backbone Amplic FWD	tacaagcaacgatctccaggaccatc
OG213	Backbone Amplic REV	ggttccggctgtcttgcttagttg

Table 11: Primers used for creating the library of YTK promoters with the SUC2 gene insert.

B.4 Primers for SUC2 Knock-out - CRISPR free

Label	Sequence	Name
OG403	TTTCGTCTCGctgCACTGGTGTCGAT AATTTGTTCTACATTGACAAG	Donor downstream
OG404	TTTCGTCTCGacACTACTTCTTTTTG AGAACACTTTTACGTAGAAGACAA	Donor downstream
OG405	TTTCGTCTCCGTgtgttacaaccaattaacca attctgattagaaaaactc	pYTK096
OG406	TTTCGTCTCCTAacggttatccacagaatca ggggataac	pYTK096
OG407	TTTCGTCTCGgtTACTGTTGCGCC AGGGGAAGT	Donor upstream
OG408	TTTCGTCTCGtGAAAGGCACTTAA CTCAATTATAGGTTGTTGCG	Donor upstream
OG409	TTTCGTCTCCTTCagcttgcctcgtccccgc	pWS3917
OG410	TTTCGTCTCCGcagtatagcgaccagcatt cacatacgatt	pWS3917
OG413	caacgcggccttttacggttcc	Sequencing
OG414	cctatggaactgcctcggtgagttt	Sequencing

Table 12: Primers for assembling plasmid to allow for CRISPR-free knock-out.

B.5 Primers for SUC2 Knock-in - SUC2 Gene point mutation

Label	Sequence	Name
OG401	gcatCGTCTCATcggGGTCTCATATGCTTTT GCAAGCTTTCCTTTTCCTTTTG	OG401 - SUC2 FWD
OG402	CCGTCTCAggtcGGTCTCTGGATCTACTATT TTACTTCCCTTACTTGGAACTTGTCAATG	OG402 - SUC2 REV
OG411	GGTCCCTGGTCGCGTTTTGC	OG411 - point mut FWD
OG412	GCAAAACGCGACCAGGGACC	OG412 - point mut REV

Table 13: Primers for inducing point mutation in SUC2 fragment and putting it into YTK format.

C Genetic Parts

C.1 Yeast parts

All yeast genomes used for part amplification were from standard wild-type yeast strains used in the lab (yWS2064).

All plasmids used as assembly backbones, antibiotic selection markers, or guides were sourced from within the lab from Will Shaw's part collection and the YTK part collection.

C.2 SUC2 Knock-out

Part Type	Name	ID	Start index (bp)	End index (bp)
	Total deletion region 1		37385	39681
	Total deletion region 2		36516	38925
CDS	SUC2		37385	38989
CDS	YIL161W (non-coding gene)		39433	39691
CDS	YIL163C (coding unknown)		36899	37252
gRNA	CRISPR target upstream 1	gRNA Up 1	37397	37416
gRNA	CRISPR target upstream 2	gRNA Up 2	36513	36532
gRNA	CRISPR target downstream 1	gRNA Dn 1	39656	39675
gRNA	CRISPR target downstream 2	gRNA Dn 2	38909	38928
Donor	SUC2 Upstream donor 1	SUC2 Up 1	36885	37384
Donor	SUC2 Upstream donor 2	SUC2 Up 2	36016	36515
Donor	SUC2 Downstream donor 1	SUC2 Dn 1	39691	40190
Donor	SUC2 Downstream donor 2	SUC2 Dn 2	38926	39425

Table 14: Annotations of genetic parts in the SUC2 deletion region.

Part Name	pOG001	pOG101	pOG102
Part Type	Donor plasmid	CRISPR vector	CRISPR vector
Backbone	pYTK001	pWS3353	pWS3917
Part Insert 1	SUC2 Up 1	gRNA Up 1	gRNA Up 1
Part Insert 2	SUC2 Dn 1	gRNA Dn 1	gRNA Dn 1
CRISPR Scaffold Template	N/A	pWS3178	pWS3178
Selection Marker	Cam	Spec, URA3	Spec, Zeocin

Table 15: Plasmids used for knockout in first gRNA design.

Part Name	pOG002	pOG003
Part Type	Donor plasmid	Donor plasmid
Backbone	pYTK096	pYTK096
Part Insert 1	SUC2 Up 2	SUC2 Up 2
Part Insert 2	SUC2 Dn 2	SUC2 Dn 2
Template Vector	pWS3916	pWS3917
Selection Marker	Kan, Hygramycin	Kan, Zeocin

Table 16: Donor plasmids used for knockout in second gRNA design.

Part Name	pOG015	pOG016	pOG017
Part Type	CRISPR vector	CRISPR vector	CRISPR vector
Backbone	pYTK001	pWS3353	pWS3917
Part Insert 1	gRNA Up 2	gRNA Up 2	gRNA Up 2
Part Insert 2	gRNA Dn 2	gRNA Dn 2	gRNA Dn 2
Template Vector	pWS3178	pWS3178	pWS3178
Selection Marker	Cam	Spec, URA3	Spec, Zeocin

Table 17: CRISPR (gRNA) vectors used for knockout in second gRNA design.

C.3 Inducible SUC2 Knock-in

Part Name	Backbone	YTK Promoter	CDS	Terminator
pOG004	pWS065 (His)	pYTK009	SUC2	pYTK056 - tTDH1
pOG005	pWS065 (His)	pYTK011	SUC2	pYTK056 - tTDH1
pOG006	pWS065 (His)	pYTK013	SUC2	pYTK056 - tTDH1
pOG007	pWS065 (His)	pYTK015	SUC2	pYTK056 - tTDH1
pOG008	pWS065 (His)	pYTK017	SUC2	pYTK056 - tTDH1
pOG009	pWS065 (His)	pYTK019	SUC2	pYTK056 - tTDH1
pOG010	pWS065 (His)	pYTK021	SUC2	pYTK056 - tTDH1
pOG011	pWS065 (His)	pYTK023	SUC2	pYTK056 - tTDH1
pOG012	pWS065 (His)	pYTK025	SUC2	pYTK056 - tTDH1
pOG013	pWS065 (His)	pYTK027	SUC2	pYTK056 - tTDH1

Table 18: Library of YTK promoters on SUC2 characterisation plasmids. 10 promoters selected from YTK for integration on the His auxotrophic marker (variable parts highlighted yellow).

Part Name	Backbone	2a	2b	3	4
pOG100	pWS065 (His)	Lex UAS - 1x	pRNR2m	SUC2	tENO1
pOG101	pWS065 (His)	Lex UAS - 2x	pPHO5m	SUC2	tENO1
pOG102	pWS065 (His)	Lex UAS - 6x	pLEU2m	SUC2	tENO1

Table 19: Cassettes for three GPCR-inducible promoters (highlighted yellow) for different expression ranges of SUC2 assembled with YTK.

D Code for Model

```
# !pip install perlin_noise
# !pip install noise
# !pip install array2gif

#@title Imports
# import pkg_resources
# pkg_resources.require("cupy==9.0.0")   # For interp

from perlin_noise import PerlinNoise
from noise import snoise2

import time
from copy import deepcopy
from datetime import datetime
from pathlib import Path
import glob
from PIL import Image
import os

import numpy as np
# import cupy as np
# from numba import jit
from numba import vectorize
import matplotlib.pyplot as plt
import matplotlib.colors as pltcolors
import seaborn as sns
import pandas as pd
import skimage.filters.rank
import skimage.morphology
import imageio

from google.colab import drive
from google.colab import files
drive.mount("/content/gdrive")

plt.style.use('seaborn')
%matplotlib inline


#@title Arbitrary HP's

grid_size = 200
num_dimensions = 2
grid_dimensions = [grid_size] * num_dimensions

initial_producer_density = 0.0001 * 2
initial_sensor_density = initial_producer_density

# Names
sensor = 'sensor'
producer_A = 'H2_(low)'
producer_B = 'H5_(medium)'
producer_C = 'H4_(high)'
```

```
initial_producer_density = 0.002
initial_sensor_density = initial_producer_density
cell_type_info = {
    sensor: [
             initial_sensor_density,     # Cell Density
             (90 * 60),                  # Doubling time (sec)
             0],                         # Producer strength
    producer_A: [
                 initial_producer_density,
                 (90 * 60),
                 0.0000000148310251434],
    producer_B: [
                 initial_producer_density,
                 (90 * 60),
                 0.000000595353313081],
    producer_C: [
                 initial_producer_density,
                 (90 * 60),
                 0.0000211020342857]
}
cell_type_info["null"] = [
                          1 - sum([i[0] for i in cell_type_info.values()]),
                          0,
                          0]

perlin_noise_octaves = {16: 0.25}   # 4: 0.2, 8: 0.5,



# @title Class: Base Grid
class Base_grid():
    def __init__(self):
        self.gif_directory = ''
        self.output_dir = './'
        self.np_dtype = np.float32
        self.date = datetime.now().strftime("%Y_%m_%d-%I%M%S")
        self.root_save_folder = '/content/gdrive/MyDrive/FYP_colab_outputs'

    # Decorators
    @staticmethod
    def timer(func):
        '''Decorator that reports the execution time.'''

        def wrap(*args, **kwargs):
            start = time.time()
            result = func(*args, **kwargs)
            end = time.time()

            print("Function_{}_took".format(func.__name__), end-start)
            return result
        return wrap

    # Helpers
    def animator(self, gif_name='cell_grid'):
        """ Animate numpy array """

        self.gif_save_name = '{}/{}_{}.gif'.format(self.output_dir,
            gif_name, self.date, self.cell_seed_iter)
        frame_rate = 5      # fps

        frames = []
        imgs = glob.glob("./{}/*.png".format(self.gif_directory))
        imgs.sort()

        for i in imgs:
            new_frame = Image.open(i)
            frames.append(new_frame)

        # Save into a GIF file that loops forever
        frames[0].save(self.gif_save_name, format='GIF',
                       append_images=frames[1:],
                       save_all=True,
                       duration=(len(frames)/frame_rate), loop=1)

    def exclude_keys(self, d, excluded_keys, return_type='values'):
        if return_type == 'values':
            return [x for k, x in d.items() if k not in excluded_keys]
        elif return_type == 'keys':
            return [x for x in d.keys() if x not in excluded_keys]
        else:
            return {k: x for k, x in d.items() if k not in excluded_keys}

    def play_gif(self, fname=None, url=None):
        from IPython.display import Image as pyImage
        if url:
            !wget url
        if fname==None:
            fname = self.gif_save_name
        pyImage(open(fname,'rb').read())

    def reset(self):
        Path(self.gif_directory).mkdir(parents=True, exist_ok=True)
        Path(self.output_dir).mkdir(parents=True, exist_ok=True)

        rem_files = glob.glob('./{}/*'.format(self.gif_directory))
        for f in rem_files:
            os.remove(f)

    def visualise_grid(self, grid, title_text='', opt_text='',
                       opt_cbar_text='Co-growth', show=True, save_name='',
                       quantised=False, opt_cbar_tickers=''):
        plt.style.use('default')

        grid_max = np.max(grid)
        num_colors = grid_max+1

        fig = plt.figure()
        ax = fig.add_subplot()
        ax.set_title(opt_text)
        cmap = plt.get_cmap('viridis', num_colors) if quantised else plt.get_cmap('viridis')
```

```
        fig.subplots_adjust(right=0.85)
        plt.title(title_text)
        im = ax.imshow(grid.copy(), cmap=cmap, vmin=0, vmax=grid_max)
        cbar = fig.colorbar(im)
        if quantised:
            bounds = np.array(range(0, num_colors)) * (1-1/grid_max)
            cbar.set_ticks(np.array(bounds)+0.5)
            tick_texts = cbar.ax.set_yticklabels(opt_cbar_tickers)
            tick_texts[0].set_verticalalignment('top')
            tick_texts[-1].set_verticalalignment('bottom')
        cbar.ax.set_xlabel(opt_cbar_text, labelpad=20)

        if show:
            print('Visualising...', opt_text)
            plt.show()
        else:
            plt.savefig('./{}/{}'.format(self.gif_directory, save_name))
            plt.close(fig)

    def visualise_grids(self, grids, opt_text=''):
        assert type(grids) == dict, "To visualise multiple grids, make sure they are passed as dict"

        for molec, grid in grids.items():
            self.visualise_grid(grid, opt_text=molec, opt_cbar_text=molec)

    # Grid Functions
    def get_cell_counts(self, grid):
        total_counts = {x: 0 for x in self.all_identities.keys()}
        for i, key in enumerate(total_counts.keys()):
            total_counts[key] = np.sum(np.where( grid == i ))
        return total_counts

    def get_diffuse_grid(self, grid_dimensions, noise_type="normal", noise_seed=1):
        if noise_type == "perlin":
            grid = np.array(self.get_perlin_texture(grid_dimensions, noise_seed), dtype=self.np_dtype)
        else:
            grid = np.array(self.get_normal_texture(grid_dimensions), dtype=self.np_dtype)
        return grid

    def get_empty_grid(self, grid_dimensions):
        return np.zeros(grid_dimensions, dtype=self.np_dtype)

    def get_quant_grid(self, grid_dimensions, densities, noise_type="perlin", noise_seed=1):
        """ Initialise the grid with each cell type in certain spatial distributions,
        eg Perlin noise or normal distributions. """

        get_texture = self.get_perlin_texture if noise_type == "perlin" else self.get_normal_texture
        texture = get_texture(grid_dimensions, noise_seed)

        # Scale to 1
        final_density = np.sum(np.array([i for i in densities.values()]))
        if not (final_density == 1): print('Initial densities do not add to 1')

        grid = self.get_empty_grid(grid_dimensions)
        cumul_density = 0
        for i, density in enumerate(densities.values()):
            # Normalise to 1
            density = density / final_density
            # print('index ' + str(i+1) + ' density: ' + str(density) + ' cumul_density: ' + str(cumul_density))
            grid += ((texture >= cumul_density) & (texture <= (cumul_density+density))) * int(i)
            cumul_density += density

        return np.array(grid, dtype=np.uint8)

    def get_normal_texture(self, grid_dimensions, new_seed=0):
        np.random.seed(new_seed)
        if self.dimensionality == 2:
            texture = np.random.rand(grid_dimensions[0], grid_dimensions[1])
        elif self.dimensionality == 3:
            texture = np.random.rand(grid_dimensions[0], grid_dimensions[1], grid_dimensions[2])
        texture = np.interp(texture, np.array([texture.min(), texture.max()]), np.array([0, 1]))
        return texture

    def get_perlin_texture(self, grid_dimensions, new_seed=0):
        noises = []
        for octave in self.perlin_noise_octaves.keys():
            noise = PerlinNoise(octaves=octave, seed=new_seed)
            noises.append(noise)

        disc_noises = []
        if self.dimensionality == 3:
            xpix, ypix, zpix = grid_dimensions
            for bias, noise in zip(self.perlin_noise_octaves.values(), noises):
                disc_noise = [[[bias * noise([i/xpix, j/ypix, k/zpix])
                    for j in range(xpix)] for i in range(ypix)] for k in range(zpix)]
                disc_noises.append(disc_noise)
        elif self.dimensionality == 2:
            xpix, ypix = grid_dimensions
            for (octave, bias), noise in zip(self.perlin_noise_octaves.items(), noises):
                if self.use_snoise:
                    disc_noise = [[bias * snoise2(i/xpix, j/ypix, octave)
                        for j in range(xpix)] for i in range(ypix)]
                else:
                    disc_noise = [[bias * noise([i/xpix, j/ypix])
                        for j in range(xpix)] for i in range(ypix)]
                disc_noises.append(disc_noise)

        texture = disc_noises[0]
        if len(disc_noises) > 1:
            for disc_noise in disc_noises[1:]:
                np.add(texture, disc_noise)
        texture = np.array(texture, dtype=self.np_dtype)

        return np.interp(texture, (texture.min(), texture.max()), (0, 1))
```

```
# @title Grid Functions
np_dtype=np.float32
def get_diffuse_grid(grid_dimensions, noise_type="normal", noise_seed=1):
    if noise_type == "perlin":
        grid = np.zeros(grid_dimensions, dtype=np_dtype)
    else:
        grid = np.zeros(grid_dimensions, dtype=np_dtype)
    return grid

def get_empty_grid(grid_dimensions):
    return np.zeros(grid_dimensions, dtype=np_dtype)


molecule_info = {
    'glucose': [
                1,                  # Diffusivity
                get_empty_grid(grid_dimensions),   # Grid
                0.0001,                                 # Degradation rate
                0.01],
    'invertase': [
                    1,
                    get_empty_grid(grid_dimensions),
                    0.001,
                    0],
    'melatonin': [
                    1,
                    get_empty_grid(grid_dimensions),
                    0.00001,
                    0],
    'sucrose': [
                1,
                get_diffuse_grid(grid_dimensions, noise_seed=2),
                0.0001,
                0]
}

molecules_grid = {k: i[1] for k, i in molecule_info.items()}

x = np.zeros(grid_dimensions)
top = molecules_grid['sucrose'][0:-2,1:-1]



class Grid_model(Base_grid):
    def __init__(self, grid_dimensions, cell_type_info, init_noise_type="normal",
        perlin_noise_octaves={4: 1, 8: 0.5, 16: 0.25}):
        super(Grid_model, self).__init__()

        self.grid_dimensions = np.array(grid_dimensions, dtype=self.np_dtype)
        if np.size(self.grid_dimensions) == 3:
            self.dimensionality = 3
        elif np.size(self.grid_dimensions) == 2:
            self.dimensionality = 2
        else:
            print('Invalid grid dimensionality (choose 2 or 3)')
        self.init_noise_type = init_noise_type
        self.perlin_noise_octaves = perlin_noise_octaves
        self.use_snoise = True      # Faster than perlin noise library
        self.animate = True
        self.cell_seed = [0,1,2,3,4,5,6]
        self.cell_seed_iter = 0
        self.time_steps_taken = 0

        # File & dir names
        self.gif_directory = 'Grid_plots'
        self.output_dir = self.root_save_folder  # 'Outputs'
        self.gif_save_name = None
        self.fig_name_cell_counts = '{}/total_cell_counts_{}_{}.png'.format(
            self.root_save_folder, self.date, self.cell_seed_iter)
        self.summary_text_fname = "{}/summary_text_{}_{}.txt".format(
            self.output_dir, self.date, self.cell_seed_iter)

        # Identities
        self.all_identities = {key: i for i, key in enumerate(cell_type_info)}
        self.null_identity = self.all_identities['null']
        self.sensor_identity = self.all_identities['sensor']
        self.cell_identities = self.exclude_keys(self.all_identities, ['null'], 'dict')
        self.producer_names = self.exclude_keys(self.all_identities, ['null', 'sensor'], 'keys')
        self.cell_densities = {k: i[0] for k, i in cell_type_info.items()}
        self.producer_strengths = {k: i[2] for k, i in cell_type_info.items()}

        self.total_cell_counts = {k: [[] for k in range(len(self.cell_seed))]
            for k in self.all_identities.keys()}

        # Sucrose -> glucose: from https://pubs.acs.org/doi/10.1021/ja802206s
        self.k_invertase_cat = 1 * np.power(10, 4)              # Unit (s-1)
        self.Km = 0.02                                          # Unit (M)

        # The poly should fit to:
        # Baseline MTNR1A Strain: poly1d([2.46487513e+02, 5.14351152e+03, 3.20442125e+04, 9.38276439e+04, 3.98583791e+05])
        # MTNR1A with SUC2 Knockout: poly1d([4.07399750e+02, 1.12997859e+04, 1.03845445e+05, 3.88466713e+05, 7.73390194e+05])
        # Their average: poly1d([3.26943631e+02, 8.22164870e+03, 6.79448286e+04, 2.41147179e+05, 5.85986993e+05])
        self.f_melatonin_knockout = np.poly1d(
            [3.26943631e+02, 8.22164870e+03, 6.79448286e+04, 2.41147179e+05, 5.85986993e+05])

        # Growth
        self.use_normalised_growth = True           # Re-parameterise growth curve to fit [0,1]
        self.use_simple_growth = False                # Squish x-axis to fit desired simulation time
        self.time_stationary = 150000
        self.desired_simulation_time = 100
        self.time_dilation = self.time_stationary / self.desired_simulation_time  # Subsampling: Realistic parameters take ca

        self.gluc_bump_percentage = 1
        self.initial_growth = 0.01

        # Modelling curve:
        # Values taken from S. Cerevisiae in
        # https://eprints.ucm.es/id/eprint/32734/1/Sil%C3%B3niz%20Letters%20in%20apllied%20microbiologyr1_LAM_12314_review-1d
        log_mew_max = 0.007
        A = 233.16 # - self.initial_growth
```

```
        if self.use_normalised_growth:  # Simplify parameters
            time_squish = self.time_stationary / self.desired_simulation_time
            self.log_mew_max = {k: 1 for k in self.cell_identities}
            self.log_asymptotic_A = {k: A for k in self.cell_identities}
            self.doubling_times = self.exclude_keys({k: i[1] / time_squish
                for k, i in cell_type_info.items()}, ['null'], 'dict')
        else:
            self.log_mew_max = {k: log_mew_max for k in self.cell_identities}
            self.log_asymptotic_A = {k: A for k in self.cell_identities}
            self.doubling_times = self.exclude_keys({k: i[1] for k, i in cell_type_info.items()}, ['null'], 'dict')

        self.log_lam = self.get_log_lam()
        self.collapse_p_div = True if np.average(
            list(self.doubling_times.values())) == list(self.doubling_times.values())[0] else False

        # Reset
        self.reset()

    def initialise_world(self, cell_seed):
        # Grids
        self.cell_grid = self.get_quant_grid(
            grid_dimensions, self.cell_densities, self.init_noise_type, noise_seed=cell_seed)
        print('Starting Cell counts: ', self.get_cell_counts(self.cell_grid))

        # Molecules
        self.infinite_sucrose = True    # Basically ignore sucrose

        # Diffusivities in (cm^2 s^-1)
        # All from old source: https://pubs-acs-org.iclibezp1.cc.ic.ac.uk/doi/pdf/10.1021/bi00910a019
        # Conflicting but close constants in Table 5 Glucose (3.6) https://www.sciencedirect.com/science/article/pii/014102299
        # and https://www.sciencedirect.com/science/article/pii/0141022993900546
        self.glucose_diffusivity = 0.00006  # 0.60 * np.power(10, 5)
        self.invertase_diffusivity = 0.00001  # 1 * np.power(10, 5)
        self.melatonin_diffusivity = 0.00001  # 1 * np.power(10, 5)
        self.sucrose_diffusivity = 0.000042  # 0.42 * np.power(10, 5)

        self.molecule_info = {
            'glucose': [
                        self.glucose_diffusivity,                # Diffusivity
                        self.get_empty_grid(grid_dimensions),    # Grid
                        0.0001,                                  # Degradation rate (not used)
                        0.0000016],                              # Consumption rate M/s from https://www.ncbi.nlm.nih.gov/pmc/
            'invertase': [
                          self.invertase_diffusivity,
                          self.get_empty_grid(grid_dimensions),
                          0.001,
                          0],
            'melatonin': [
                          self.melatonin_diffusivity,
                          self.get_empty_grid(grid_dimensions),
                          0.00001,
                          0],
            'sucrose': [
                        self.sucrose_diffusivity,
                        self.get_diffuse_grid(grid_dimensions, noise_seed=cell_seed) * np.power(10, 8),
# Basically infinite sucrose
                        0.0001,
                        0]
        }

        # Physical properties
        self.dx2 = self.dy2 = 0.0004 # 4 * np.power(10, 4) # 4 microns in cm because of diffusivity units

        self.diffusivities = {k: i[0] for k, i in self.molecule_info.items()}
        self.molecules_grid = {k: i[1] for k, i in self.molecule_info.items()}
        self.degradation = {k: i[2] for k, i in self.molecule_info.items()}
        self.consumption = {k: i[3] for k, i in self.molecule_info.items()}

        self.dt = np.min([(self.dx2 * self.dy2 / (2 * D * (self.dx2 + self.dy2)))
            for D in self.diffusivities.values()])
        self.dt = self.dt / 1.5

        # Probabilities
        self.p_starvation_init = np.average(
            [self.log_phase_step_deriv(
                cell_type, self.time_steps_taken)/self.log_mew_max[cell_type] for cell_type in self.cell_identities.keys()])
        self.p_starvation = np.ones(grid_dimensions) * self.p_starvation_init
        self.p_division_init = 0.001
        self.p_division = {k: np.ones(grid_dimensions) * self.p_division_init for k in self.cell_identities.keys()}

    # Growth equations
    def get_log_lam(self):
        """ Calculate the lag time lambda for parameterizing the yeast growth curve """
        log_lam = {k: None for k in self.cell_identities.keys()}
        for cell_type in self.cell_identities.keys():
            u = self.log_mew_max[cell_type]
            A = self.log_asymptotic_A[cell_type]
            t2 = self.doubling_times[cell_type]

            lam = A * (np.log( (-1) * np.log( self.initial_growth / A )) - 1) / (u * np.exp(1)) + t2
            log_lam[cell_type] = lam
        return log_lam

    def log_phase_step(self, cell_type, t):
        """ Based on Gompertz kinetics https://www.ncbi.nlm.nih.gov/pmc/articles/PMC184525/?page=2
        https://eprints.ucm.es/id/eprint/32734/1/Sil%C3%B3niz%20Letters%20in%20apllied%20microbiologyr1_LAM_12314_review-1def
        mew_max: ( _max ) Maximum specific growth rate. Also the inflexion point on the Gompertz growth curve
        asymptotic_A: Asymptotic value
        lam: (Greek lambda) Lag time. Intercept of x-axis (time) and linear extension of log curve
        """
        t = t if self.use_normalised_growth else t * self.time_dilation
        B = ( (self.log_mew_max[cell_type] * np.exp(1)) / self.log_asymptotic_A[cell_type]
            * (self.log_lam[cell_type] - t))
        if self.use_simple_growth:
            return (self.initial_growth + self.log_asymptotic_A[cell_type]
                * np.exp( (-1) * np.exp( (B + 1)))) / self.time_dilation
        else:
            return self.initial_growth + self.log_asymptotic_A[cell_type] * np.exp( (-1) * np.exp( (B + 1)))

    def log_phase_step_deriv(self, cell_type, t):
        """ Derivative of growth curve. Used to update growth. """
```

```
        u = self.log_mew_max[cell_type]
        A = self.log_asymptotic_A[cell_type]
        lam = self.log_lam[cell_type]
        t = t if self.use_normalised_growth else t * self.time_dilation

        B = (u * np.exp(1) * (lam - t) / A) + 1
        if self.use_simple_growth:
            return u * np.exp( (-1)*np.exp(B + 1) + B + 2) / self.time_dilation
        else:
            return u * np.exp( (-1)*np.exp(B + 1) + B + 2)

    def calculate_next_growth(self, cell_type, t): #, current_growth, phase, doubling_time, derivative=0):
        return self.log_phase_step(cell_type, t)

    @Base_grid.timer
    def test_calculate_next_growth(self, time_steps = 100000):
        current_growth = 0
        time_dilation = self.time_dilation if (self.use_normalised_growth or self.use_simple_growth) else 1

        history = {k: [np.zeros(time_steps), np.zeros(time_steps)] for k in self.cell_identities.keys()}
        for t in range(time_steps):
            for cell_type in self.cell_identities.keys():
                ng = self.calculate_next_growth(cell_type, t)
                dg = self.log_phase_step_deriv(cell_type, t)
                history[cell_type][0][t] = ng
                history[cell_type][1][t] = dg

        for cell_type in self.cell_identities.keys():
            plt.plot(np.arange(0, time_steps) * time_dilation, history[cell_type][0])
            plt.plot(np.arange(0, time_steps) * time_dilation, history[cell_type][1])
        plt.xlabel('time')
        plt.ylabel('growth')
        plt.legend([k for k in self.cell_identities.keys()])
        plt.show()

    def get_growth_phase(self, phase):
        for k, v in self.phase_index.items():
            if (phase >= v[0]) and (phase <= v[1]):
                return k
        print("Phase could not be determined")

    def get_invertase(self, melatonin_array):
        """ Function translating melatonin sensed by sensor cells into invertase produced.
        Curve based on my own data of melatonin sensing (basically the same as Will Shaw's curve),
        relative levels of expression of inducible promoter, and GFP expression levels
        from sensed melatonin.
        """
        # Convert to log. Prevent -inf
        log_mel = np.log10(melatonin_array + 0.000000000001)
        invertase = self.f_melatonin_knockout(log_mel)
        invertase[ np.where(invertase < 0) ] = 0
        return invertase

    def plot_cell_counts(self, tot_time, seed_iter=0, all_seeds=False):
        plt.style.use('seaborn')

        if all_seeds:
            x = pd.DataFrame()

            cell_type = list(self.cell_identities.keys())[0]
            time_line = [(np.arange(tot_time) * self.dt * self.time_dilation) for k in range(len(self.cell_seed))]
            time_line = np.array(time_line)
            time_line = time_line.flatten()
            x['Time'] = time_line
            seeds = [np.ones(
                len(self.total_cell_counts[cell_type][0])) * i for i in range(
                    len(self.total_cell_counts[cell_type]))]
            seeds = np.array(seeds)
            seeds = seeds.flatten()
            x['Seeds'] = seeds

            for cell_type in self.total_cell_counts.keys():
                if cell_type == 'null':
                    continue
                b = np.array(self.total_cell_counts[cell_type])
                b = b.flatten()
                x[cell_type] = b

            for cell_type in self.total_cell_counts.keys():
                if cell_type == 'null':
                    continue
                sns.lineplot(data=x, x="Time", y=cell_type, label=cell_type)

            plt.title("Cell counts over {}s and {} repeats".format(
                int(tot_time * self.dt * self.time_dilation), len(self.cell_seed)))

        else:
            for cell_type in self.all_identities.keys():
                if cell_type == 'null':
                    continue
                plt.plot((np.arange(
                    tot_time) * self.dt * self.time_dilation),
                    self.total_cell_counts[cell_type][seed_iter], label=str(cell_type))
            plt.title("Cell counts over {}s".format(int(tot_time * self.dt * self.time_dilation)))

        plt.xlabel('Time (s)')
        plt.ylabel('Cell count')
        plt.legend(loc='upper left')
        plt.savefig(self.fig_name_cell_counts)
        plt.show()

    def summarise(self, time_steps=1, seed=0, seed_iter=-1):
        # Generate gif
        if self.animate:
            self.animator('cell_grid_seed{}'.format(seed))

        # Visualise final cell environment
        self.visualise_grids(self.molecules_grid)
        self.visualise_grid(
            self.cell_grid, 'Growth_Simulation',
            quantised=True, opt_cbar_tickers=list(self.all_identities.keys()))
```

```
        self.plot_cell_counts(time_steps, seed_iter)

        for cell_type in self.cell_identities.keys():
            self.visualise_grid(
                self.p_division[cell_type], 'Probability_of_division_{}'.format(cell_type))

        total_end_counts = self.get_cell_counts(self.cell_grid)

        # Save summary text & params
        summary_text = ['Total_number:_{}'.format(total_end_counts),
                        'Identities:_'.format(self.all_identities),
                        'Total_time_simulated_(where_dt={}):_{}s'.format(
                            self.dt, time_steps * self.dt * self.time_dilation),
                        'Glucose-based_percentage_growth_increase:_{}'.format(self.gluc_bump_percentage)]
        text_file = open(self.summary_text_fname, "w")
        for s in summary_text:
            n = text_file.write(s + '\n')
            print(s)
        text_file.close()

    @Base_grid.timer
    def simulate(self, time_steps=10):

        for s_iter, s in enumerate(self.cell_seed):
            self.cell_seed_iter = s_iter
            self.update_fnames()
            self.initialise_world(s)

            for t in range(time_steps):
                self.simulate_growth(return_array=True)
                self.update_cell_count()
                # print('np.shape(self.cell_grid)', np.shape(self.cell_grid))
                self.visualise_grid(
                    grid=self.cell_grid,
                    title_text='Time:_{}s_Seed:_{}'.format(str(int(t * self.dt * self.time_dilation)), s),
                    opt_text='', opt_cbar_text='Strain', show=False, save_name='cells_{}'.format(str(t).zfill(5)),
                    quantised=True, opt_cbar_tickers=list(self.all_identities.keys()))
                self.time_steps_taken += 1

                if np.mod(self.time_steps_taken, 1000) == 0:
                    # Play an audio beep. Any audio URL will do.
                    from google.colab import output
                    output.eval_js('new_Audio("https://upload.wikimedia.org/wikipedia/commons/9/91/Sound4.wav").play()')

            self.summarise(time_steps, s, s_iter)

        self.plot_cell_counts(time_steps, all_seeds=True)

    def simulate_growth(self, return_array=False):
        # Cells
        self.take_flux_step()            # Includes time dilation
        self.take_growth_step_3()        # Includes time dilation through probabilities

        # Molecules
        self.take_diffusion_step()       # Includes time dilation
        self.update_probabilities()

        return self.cell_grid

    @Base_grid.timer
    def simulate_diffusion(self, time_steps=10):
        """ Just for testing. """
        for t in range(time_steps * self.time_dilation):
            self.take_diffusion_step()
            self.time_steps_taken += 1

    def take_diffusion_step(self):
        for dil in range(int(self.time_dilation)):
            for molec in self.molecules_grid.keys():
                if self.infinite_sucrose and (molec == 'sucrose'):
                    continue
                diffusivity = self.diffusivities[molec]
                self.molecules_grid[molec] = self.update_grid_heat(self.molecules_grid[molec], diffusivity)

    def take_flux_step(self):
        """ Simulate metabolic flux of relevant compounds """
        # Melatonin production
        for producer_name in self.producer_names:
            producer_id = self.all_identities[producer_name]
            producer_rate = self.producer_strengths[producer_name]
            self.molecules_grid['melatonin'][ np.where(self.cell_grid == producer_id) ] += producer_rate * self.dt * self.tim

        # Invertase production
        invertase_update = np.where( self.cell_grid == self.sensor_identity )
        self.invertase_secretion = self.get_invertase(self.molecules_grid['melatonin'] / self.time_dilation)
        self.molecules_grid['invertase'][invertase_update] += self.invertase_secretion[invertase_update] * self.dt * self.tim

        # Sucrose -> glucose: Michaelis-Menten
        glucose_Km_denom = self.Km + self.molecules_grid['sucrose']
        dP = self.k_invertase_cat * self.molecules_grid['invertase'] * (self.molecules_grid['sucrose'] / glucose_Km_denom) *
        self.molecules_grid['glucose'] += dP
        if not self.infinite_sucrose:
            self.molecules_grid['sucrose'] -= dP

        # Cells 'eat'
        for cell_type, cell_id in self.cell_identities.items():
            cell_locs = np.where(self.cell_grid == cell_id)
            self.molecules_grid['glucose'][cell_locs] -= self.consumption['glucose'] * self.dt * self.time_dilation * self.mo

    def take_growth_step_3(self):
        """ Determine cells to grow and die """
        # Define "next to" - this may be a square, diamond, etc
        neighborhood = skimage.morphology.disk(radius=1)

        # Make free neighbors into growth areas
        for cell_key, cell_id in self.cell_identities.items():
            img_raw = deepcopy(self.cell_grid)
            img_raw[img_raw != cell_id] = self.null_identity

            min = (skimage.filters.rank.minimum(img_raw, neighborhood) == cell_id)
```

```
            max = (skimage.filters.rank.maximum(img_raw, neighborhood) == self.null_identity)
            neighbors = min & max

            chosen = np.random.rand(grid_dimensions[0], grid_dimensions[1])
            neighbors[self.p_division[cell_key] < chosen] = False

            self.cell_grid[neighbors] = cell_id

        # Death
        dead_grid = np.random.rand(grid_dimensions[0], grid_dimensions[1])
        self.cell_grid[self.p_starvation > dead_grid] = self.null_identity

    def update_cell_count(self):
        current_cell_counts = self.get_cell_counts(self.cell_grid)
        for cell_type in self.total_cell_counts.keys():
            self.total_cell_counts[cell_type][self.cell_seed_iter].append(
                current_cell_counts[cell_type])

    def update_fnames(self):
        self.fig_name_cell_counts = '{}/total_cell_counts_{}_{}.png'.format(
            self.root_save_folder, self.date, self.cell_seed_iter)
        self.summary_text_fname = "{}/summary_text_{}_{}.txt".format(
            self.output_dir, self.date, self.cell_seed_iter)

    # @jit(nopython=True, parallel=True)
    def update_grid_heat(self, grid, diffusivity=1):
        """ Update a grid with the heat equation diffusion. dU/dt = diffusivity * DEL(U).
        See https://scipython.com/book/chapter-7-matplotlib/examples/the-two-dimensional-diffusion-equation/
        for inspiration.

        The dx^2 and dy^2 components are like width and height of each cell. """

        top = grid[0:-2,1:-1]
        left = grid[1:-1,0:-2]
        bottom = grid[2:,1:-1]
        right = grid[1:-1,2:]
        center = grid[1:-1,1:-1]

        grid[1:-1, 1:-1] = center + diffusivity * self.dt * (
            (top - 2* center + bottom)/self.dy2
            + (left - 2*center + right)/self.dx2 )
        return grid

    def update_growth_phase(self):
        """ Project the growth phase of each cell in the next time step """
        # Exponential growth: k = ln2/T
        for cell_type in self.growths.keys():
            new_growth_phase = self.calculate_next_growth(cell_type, self.time_steps_taken) # current_growth, phase, doubling.
            self.growths[cell_type].append(new_growth_phase)

    def update_probabilities(self):

        # self.update_growth_phase()

        # Shared
        use_glucose = False if int(np.max(self.molecules_grid['glucose'])) == 0 else True

        # Death
        div_max = np.max(self.p_starvation) * self.gluc_bump_percentage
        if use_glucose:
            # glucose_bump = np.interp(self.molecules_grid['glucose'], 0 , div_max ) * self.k_glucose_division
            glucose_bump = self.molecules_grid['glucose'] * (div_max/np.max(
                self.molecules_grid['glucose']))
        else:
            glucose_bump = np.zeros(np.shape(self.molecules_grid['glucose']))
        self.p_starvation = self.p_starvation_init - glucose_bump
        # Throttle
        self.p_starvation[ np.where(self.p_starvation > 1) ] = 1
        self.p_starvation[ np.where(self.p_starvation < 0) ] = 0

        # Life
        for cell_type in self.cell_identities.keys():
            # Growth rate affects probability of division
            self.p_division_growth = self.log_phase_step_deriv(
                cell_type, self.time_steps_taken)/self.log_mew_max[cell_type]

            # Glucose  content imposed on probability of division as 5% (eg relative power of glucose to up division)
            div_max = np.max(self.p_division[cell_type]) * self.gluc_bump_percentage
            if use_glucose:
                glucose_bump = self.molecules_grid['glucose'] * (div_max/np.max(
                    self.molecules_grid['glucose']))
            else:
                glucose_bump = np.zeros(np.shape(self.molecules_grid['glucose']))
            self.p_division[cell_type] = self.p_division_init + self.p_division_growth + glucose_bump

            # Zero glucose means no growth
            gluc_locs = np.where(self.molecules_grid['glucose'] == 0)
            self.p_division[cell_type][gluc_locs] = 0

            # Throttle
            self.p_division[cell_type][ np.where(self.p_division[cell_type] > 1) ] = 1
            self.p_division[cell_type][ np.where(self.p_division[cell_type] < 0) ] = 0
            p_division = deepcopy(self.p_division)

            if self.collapse_p_div:  # Same matrix for all cell types
                for ct in self.cell_identities.keys():
                    self.p_division[ct] = self.p_division[cell_type]
                break


# Simulation
Grid = Grid_model(grid_dimensions, cell_type_info)
Grid.simulate(time_steps=1000)


# Play gif
from IPython.display import Image as pyImage

fname = Grid.gif_save_name
pyImage(open(fname,'rb').read())
```

References

[1] G. O. Gowers, S. M. Chee, D. Bell, L. Suckling, M. Kern, D. Tew, D. W. McClymont, T. Ellis, Improved betulinic acid biosynthesis using synthetic yeast chromosome recombination and semi-automated rapid lc-ms screening, Nature Communications 11 (2020) 1–7. doi:10.1038/s41467-020-14708-z.
URL https://doi.org/10.1038/s41467-020-14708-z

[2] B. A. Blount, G. O. Gowers, J. C. Ho, R. Ledesma-Amaro, D. Jovicevic, R. M. McKiernan, Z. X. Xie, B. Z. Li, Y. J. Yuan, T. Ellis, Rapid host strain improvement by in vivo rearrangement of a synthetic yeast chromosome, Nature Communications 9 (2018) 1–10. doi:10.1038/s41467-018-03143-w.
URL www.nature.com/naturecommunications

[3] K. Stephens, M. Pozo, C. Y. Tsao, P. Hauk, W. E. Bentley, Bacterial co-culture with cell signaling translator and growth controller modules for autonomously regulated culture composition, Nature Communications 10 (2019) 1–11. doi:10.1038/s41467-019-12027-6.
URL https://doi.org/10.1038/s41467-019-12027-6

[4] K. Brenner, L. You, F. H. Arnold, Engineering microbial consortia: a new frontier in synthetic biology (9 2008). doi:10.1016/j.tibtech.2008.05.004.

[5] L. Goers, P. Freemont, K. M. Polizzi, Co-culture systems and technologies: Taking synthetic biology to the next level (7 2014). doi:10.1098/rsif.2014.0065.
URL /pmc/articles/PMC4032528/?report=abstract https://www.ncbi.nlm.nih.gov/pmc/articles/PMC40

[6] W. Bacchus, M. Fussenegger, Engineering of synthetic intercellular communication systems (3 2013). doi:10.1016/j.ymben.2012.12.001.

[7] P. Du, H. Zhao, H. Zhang, R. Wang, J. Huang, Y. Tian, X. Luo, X. Luo, M. Wang, Y. Xiang, L. Qian, Y. Chen, Y. Tao, C. Lou, De novo design of an intercellular signaling toolbox for multi-channel cell–cell communication and biological computation, Nature Communications 11 (2020) 1–11. doi:10.1038/s41467-020-17993-w.
URL https://doi.org/10.1038/s41467-020-17993-w

[8] G. Perrino, A. Hadjimitsis, R. Ledesma-Amaro, G.-B. Stan, Control engineering and synthetic biology: working in synergy for the analysis and control of microbial systems, Current Opinion in Microbiology 62 (2021) 68–75. doi:10.1016/j.mib.2021.05.004.
URL https://linkinghub.elsevier.com/retrieve/pii/S1369527421000655

[9] J. Tasoff, M. T. Mee, H. H. Wang, An economic framework of microbial trade, PLoS ONE 10 (2015) e0132907. doi:10.1371/journal.pone.0132907.
URL www.nih.gov/].

[10] S. Billerbeck, J. Brisbois, N. Agmon, M. Jimenez, J. Temple, M. Shen, J. D. Boeke, V. W. Cornish, A scalable peptide-gpcr language for engineering multicellular communication, Nature Communications 9 (2018) 1–12. doi:10.1038/s41467-018-07610-2.
URL www.nature.com/naturecommunications

[11] W. Shaw, Refactoring yeast signalling pathways for tuneable extracellular biosensing (2020).

[12] A. M. Ehrenworth, T. Claiborne, P. Peralta-Yahya, Medium-throughput screen of microbially produced serotonin via a g-protein-coupled receptor-based sensor, Biochemistry 56 (2017) 5471–5475. doi:10.1021/acs.biochem.7b00605.
URL https://pubs.acs.org/doi/pdf/10.1021/acs.biochem.7b00605

[13] E. Leonard, P. K. Ajikumar, K. Thayer, W. H. Xiao, J. D. Mo, B. Tidor, G. Stephanopoulos, K. L. Prather, Combining metabolic and protein engineering of a terpenoid biosynthetic pathway for overproduction and selectivity control, Proceedings of the National Academy of Sciences of the United States of America 107 (2010) 13654–13659. doi:10.1073/pnas.1006138107.
URL www.pnas.org/cgi/doi/10.1073/pnas.1006138107

[14] A. Casini, F.-Y. Chang, R. Eluere, A. M. King, E. M. Young, Q. M. Dudley, A. Karim, K. Pratt, C. Bristol, A. Forget, A. Ghodasara, R. Warden-Rothman, R. Gan, A. Cristofaro, Amin, E. Borujeni, M.-H. Ryu, J. Li, Y.-C. Kwon, H. Wang, E. Tatsis, C. Rodriguez-Lopez, S. O'connor, M. H. Medema, M. A. Fischbach, M. C. Jewett, C. Voigt, D. B. Gordon, A pressure test to make 10 molecules in 90 days: External evaluation of methods to engineer biology (2018). doi:10.1021/jacs.7b13292.
URL https://pubs.acs.org/sharingguidelines

[15] C. Tuerk, L. Gold, Systematic evolution of ligands by exponential enrichment: Rna ligands to bacteriophage t4 dna polymerase, Science 249 (1990) 505–510. doi:10.1126/science.2200121.
URL http://science.sciencemag.org/

[16] D. Q. Zheng, J. Chen, K. Zhang, K. H. Gao, O. Li, P. M. Wang, X. Y. Zhang, F. G. Du, P. Y. Sun, A. M. Qu, S. Wu, X. C. Wu, Genomic structural variations contribute to trait improvement during whole-genome shuffling of yeast, Applied Microbiology and Biotechnology 98 (2014) 3059–3070. doi:10.1007/s00253-013-5423-7.
URL http://www.ncbi.nlm.nih.gov/geo;

[17] J. Yang, B. Kim, G. Y. Kim, G. Y. Jung, S. W. Seo, Synthetic biology for evolutionary engineering: From perturbation of genotype to acquisition of desired phenotype (5 2019). doi:10.1186/s13068-019-1460-5.
URL https://doi.org/10.1186/s13068-019-1460-5

[18] J. Zhang, M. K. Jensen, J. D. Keasling, Development of biosensors and their application in metabolic engineering (10 2015). doi:10.1016/j.cbpa.2015.05.013.

[19] M. R. Wu, L. Nissim, D. Stupp, E. Pery, A. Binder-Nissim, K. Weisinger, C. Enghuus, S. R. Palacios, M. Humphrey, Z. Zhang, E. M. Novoa, M. Kellis, R. Weiss, S. D. Rabkin, Y. Tabach, T. K. Lu, A high-throughput screening and computation platform for identifying synthetic promoters with enhanced cell-state specificity (specs), Nature Communications 10 (2019) 1–10. doi:10.1038/s41467-019-10912-8.
URL https://doi.org/10.1038/s41467-019-10912-8

[20] J. K. Michener, C. D. Smolke, High-throughput enzyme evolution in saccharomyces cerevisiae using a synthetic rna switch, Metabolic Engineering 14 (2012) 306–316. doi:10.1016/j.ymben.2012.04.004.

[21] S. Binder, G. Schendzielorz, N. Stäbler, K. Krumbach, K. Hoffmann, M. Bott, L. Eggeling, A high-throughput approach to identify genomic variants of bacterial metabolite producers at the single-cell level, Genome Biology 13 (2012) R40. doi:10.1186/gb-2012-13-5-r40.
URL http://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-5-r40

[22] A. Meyer, R. Pellaux, S. Potot, K. Becker, H. P. Hohmann, S. Panke, M. Held, Optimization of a whole-cell biocatalyst by employing genetically encoded product sensors inside nanolitre reactors, Nature Chemistry 7 (2015) 673–678. doi:10.1038/nchem.2301.
URL www.nature.com/naturechemistry

[23] S. Castaño-Cerezo, M. Fournié, P. Urban, J.-L. Faulon, G. Truan, Development of a biosensor for detection of benzoic acid derivatives in saccharomyces cerevisiae, Frontiers in Bioengineering and Biotechnology 7 (2020) 372. doi:10.3389/fbioe.2019.00372.
URL https://www.frontiersin.org/article/10.3389/fbioe.2019.00372/full

[24] P. W. M.-I. d. S. P. B. E. C. F. C. A. M. E.-M. Rivas, E. Gil de Prado, J. Peinado, A simple mathematical model that describes the growth of the area and the number of total and viable cells in yeast colonies, Letters in Applied Microbiology (2014). doi:10.1111/lam.12314.
URL https://eprints.ucm.es

[25] R. Salari, Investigation of the best saccharomyces cerevisiae growth condition, Electronic physician (2017). doi:10.19082/3592.

[26] E. J. Schantz, M. A. Lauffer, Diffusion measurements in agar gel, Biochemistry (1962). doi:10.1021/bi00910a019.

[27] M. Zakhartsev, M. Reuss, Cell size and morphological properties of yeast saccharomyces cerevisiae in relation to growth temperature, FEMS Yeast Research 18 (6), foy052 (04 2018). arXiv:https://academic.oup.com/femsyr/article-pdf/18/6/foy052/25134882/foy052.pdf, doi:10.1093/femsyr/foy052.
URL https://doi.org/10.1093/femsyr/foy052

[28] R. Wolfenden, Y. Yuan, Rates of spontaneous cleavage of glucose, fructose, sucrose, and trehalose in water, and the catalytic proficiencies of invertase and trehalas, Journal of the American Chemical Society 130 (2008) 7548–7549. doi:10.1021/ja802206s.
URL http://pubs.acs.org.

[29] S. M. Germann, S. A. B. Jacobsen, K. Schneider, S. J. Harrison, N. B. Jensen, X. Chen, S. G. Stahlhut, I. Borodina, H. Luo, J. Zhu, J. Maury, J. Forster, Glucose-based microbial production of the hormone melatonin in yeast saccharomyces cerevisiae, Biotechnology Journal 11 (2016) 717–724. doi:10.1002/biot.201500143.
URL www.biotechnology-journal.com

[30] A. Akhmetov, J. Laurent, J. Gollihar, E. Gardner, R. Garge, A. Ellington, A. Kachroo, E. Marcotte, Single-step precision genome editing in yeast using crispr-cas9, BIO-PROTOCOL 8 (2018). doi:10.21769/bioprotoc.2765.
URL /pmc/articles/PMC5951413/ /pmc/articles/PMC5951413/?report=abstract https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5951413/

[31] B. C. Michael E. Lee, William C. DeLoache, J. E. Dueber, A highly characterized yeast toolkit for modular, multipart assembly, American Chemical Society (2015). doi:10.1021/sb500366v.

[32] M. Versele, K. Lemaire, J. M. Thevelein, Sex and sugar in yeast: Two distinct gpcr systems (2001). doi:10.1093/embo-reports/kve132.
URL /pmc/articles/PMC1083946/ /pmc/articles/PMC1083946/?report=abstract https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1083946/

[33] D. Ashland, OR: Becton, Company, Flowjo software (for mac) version 10 (2019).

[34] T. Achstetter, Regulation of alpha-factor production in saccharomyces cerevisiae: a-factor pheromone-induced expression of the mf alpha 1 and ste13 genes., Molecular and cellular biology (1989). doi:10.1128/mcb.9.10.4507-4514.1989.