Genomic Instability at Single Cell Resolution

Ian T. Dowsett

A dissertation

Submitted in partial fulfillment of the

Requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Alan Herr, Chair

Matt Kaeberlein

Scott Kennedy

Program Authorized to Offer Degree:

Pathology

University of Washington

Abstract

Genomic Instability at Single Cell Resolution

Ian T. Dowsett

Chair of the Supervisory Committee:
Alan Herr
Department of Laboratory Medicine and Pathology

All organisms maintain the integrity of their genome through highly precise DNA replication and repair. Errors in these mechanisms can lead to genetic instability that results in cellular dysfunction or malignancy. Modern sequencing technology has enabled the development of methods to interrogate these processes during individual cell divisions. Our lab previously devised a single cell resolution approach that suggested that *Saccharomyces cerevisiae* cells with abrogated replicative fidelity exhibit multiple mutator states, a phenomenon termed ‘mutator volatility’, but the data was also consistent with a hypothesis in which replication errors segregated unequally during cell division. The research reported in my dissertation uses an expanded approach in chapter two to confirm that mutator volatility exists in strong mutators. It also shows that unequal inheritance of replication errors occurs due to the sequential processes of semiconservative DNA replication and chromosome segregation. Using computational modeling, I show that unequal segregation may dramatically expand heterogeneity in

the mutation burden in human tumors. In chapter three, I model the strongest human mutator allele in cancer (*POLE-P286R*) and find an even greater volatility. Although this mutator phenotype depends in yeast on the S-phase checkpoint, the volatility does not. In the fourth chapter, I present a novel single cell resolution method to detect loss of heterozygosity (LOH) in yeast and then use this method to investigate if LOH increases with replicative age. These findings highlight the numerous insights that may be gleaned by studying the processes that give rise to genome instability at the level of single cells.

Chapter One
Genetic Entropy – Examining the Causes and Consequences of Genomic Instability at Single Cell Resolution

Ian T. Dowsett and Alan J. Herr*

Author Information
Department of Laboratory Medicine and Pathology, University of Washington, Seattle, WA 98195-7705, USA
*Corresponding author, alanherr@uw.edu

Introduction

Physical entropy describes the tendency of matter to become increasingly disordered as a function of thermodynamics. This fundamental principle underlies not only the increasing randomness of the Universe as it expands, but also the inherent stability of macromolecules in our cells. Left unchecked, damage due to physical entropy would pose an existential threat. Cells resist entropy by using energy obtained from the environment to establish and maintain order through effector molecules encoded in their genomes, which perform a myriad of metabolic and structural roles. Critical to the defense against entropy are proteins that repair or eliminate damaged macromolecules. Life also counters entropy by simply replicating so that individual molecules or cells need not last forever. But mistakes occurring during replication and damage repair pose another threat, one of “genetic entropy” in which permanent changes accumulate in the genome. Microbial experiments show that while changes to genetic information provide fodder for evolution, an excessive mutation accumulation can drive loss of replicative fitness and even extinction of cell populations (Herr *et al.* 2014). In multicellular organisms, genetic entropy of somatic cells have been hypothesized, along with a host of other hallmarks of aging, to contribute to organismal dysfunction and aging over time (López-Otín *et al.* 2013).

In this chapter, we trace the historical foundations of the theory that genetic entropy contributes to aging. We discuss the sources of instability that generate this information loss in the nuclear and mitochondrial genomes, the epigenome and even during the expression of RNA and protein. We explore the innate mechanisms of overcoming these issues employed by eukaryotic cells and systems at the organism level. We review what is known of how genetic entropy impacts health and lifespan and chart a path forward to describing the finer details of these causes and consequences for the individual cells that compose an aging multicellular organism through the use of single cell sequencing.

Historical Foundations of the Genetic Entropy Theory of Aging

With few exceptions, organisms maintain a very low mutation rate. As proposed by Drake, this may reflect a balanced compromise struck between two extremes (Drake 1991). A mutation rate too high negatively impacts fitness of the offspring through deleterious mutation accumulation, while reducing the mutation rate below some threshold may diminish fitness by exacting too high a cost in energy expenditure and replication speed. An alternative hypothesis, put forward by Lynch, is that the lower limit of mutation rate reflects the power of genetic drift to overcome selection for alleles that would further lower the mutation rate (Lynch 2010). Whatever the fundamental cause, the downward pressure on mutation rate is clear from experiments in unicellular models. Under conditions requiring rapid evolutionary change, cells emerge with higher mutation rates due to so-called 'mutator' alleles (Mao *et al.* 1997; Miller *et al.* 1999; Giraud *et al.* 2001; Notley-McRobb *et al.* 2002). In such conditions the risk of deleterious mutations, in the short-term, is outweighed by the advantage of rare beneficial mutations that enable progeny to survive environmental challenges. However, following adaptation, lower mutation rates are at a selective advantage and mutators give way to non-mutators in the population (Giraud *et al.* 2001). Attenuation of strong mutator phenotypes also occurs in diploid organisms (Herr *et al.* 2014; Tracy *et al.* 2020).

Even with low mutation rates, mutation accumulation poses an intractable problem for asexually reproducing species. As H. J. Muller explained in his much-cited work "The Relation of Recombination to Mutational Advance" (Muller 1964), asexually reproducing organisms carry no fewer mutations than their parent and with each succeeding generation, the number of genetic differences with the original parent grows. While purifying selection continually acts on the population to maintain fitness, the drift from the parental genotype is inexorable. Sexually reproducing species bypass this so-called "Muller's Ratchet" through the process of recombination following fertilization (Muller 1964). Recombination gives progeny a chance to inherit fewer deleterious mutations than their parents, which can then be favored by selection. That is not to say that mutation accumulation within sexually reproducing species does not occur. As Peter Medawar observed, evolution cannot easily eliminate mutations that act beyond the point of an organism's peak fecundity, termed a 'selection shadow' (Medawar 1951). In what has become known as the Evolutionary Theory of Aging, Medawar theorized that these late-acting deleterious alleles lead to age-dependent dysfunction. Charles Williams advanced the related idea of "antagonist pleiotropy", which argues that some of these alleles may be beneficial early in life but cause harm to late-life function (Williams 1957). The Evolutionary Theory of Aging predicts two classes of genetic changes that influence aging. The first class, present in all members of a species, would be "fixed" in the homozygous state. Some of these variants explain why the rates of aging vary between species. The second class of mutations would have arisen after speciation and may be rare or common, depending on when they arose, the strength of selection for or against them, and genetic drift. In most cases these variants will be heterozygous and rare, since they are constantly arising. Sequencing studies reveal that the potential size of this second class of mutations is profound. The average variation between any two unrelated humans is 10,000 non-synonymous changes in

protein coding sequence. Included in these are some 250-300 potentially deleterious alleles and 50-100 genetic disease variants (Durbin *et al.* 2010). Most deleterious mutations are assumed to be recessive. This assumption has been tested using the budding yeast, *Saccharomyces cerevisiae*. Competition experiments with all 6200 heterozygous diploid yeast knock-out strains indicate that 10-20% of null mutations confer a semi-dominant reduction in replicative fitness in at least one growth condition (Delneri *et al.* 2007). Thus, a subset of the inherited burden of heterozygous deleterious SNVs may exert a direct influence on organismal function as imagined by Medawar. But genetic entropy is not confined to the germline. Such pre-existing deleterious variants may also contribute to aging by interacting with the ongoing genetic entropy occurring in somatic tissues over time.

It was Szilard who first modeled the synergy between inherited mutations (what he termed “faults”) and somatic mutagenesis. He posited that random inactivation or loss of entire chromosomes would allow these deleterious traits to be expressed and kill cells (Szilard 1959). In his model, reserves of undamaged cells buffered the individual from these lethal events for a time until they became exhausted, manifesting as aging. A modern day adaptation of this idea can be found in aging theories that incorporate stem-cell exhaustion (López-Otín *et al.* 2013). A related model, the Somatic Mutation Theory of Aging (Morley 1995), focused entirely on the increase in mutations in somatic cells as a driver of aging, not only in the nuclear genome but also in the mitochondrial genome, whose relative importance was only beginning to be appreciated at the time. Much has been learned since these theories were proposed about the changes that occur to the genome as a function of aging. The genetic entropy model of aging we propose here is a broad information-based perspective of aging. In our view, any loss of information, whether in the germline or somatic cells is relevant and cumulative. This includes loss of heterozygosity (LOH), either in the form of tracts generated by homologous recombination or aneuploidy, as well as chromosome rearrangements or translocations and even copy number variants that phenocopy LOH. Also, included under this purview are base-substitution mutations, small indels, and other DNA sequence alterations that inactivate functional genes. Genetic entropy includes changes to the epigenetic states that silence gene expression and even errors in RNA and protein synthesis, as discussed below.

Sources of Genetic Entropy

The earliest evidence that DNA damage could have biological consequences, though not discussed in those terms, can be traced back to British Surgeon Percivall Pott’s 1775 writings on the scrotal cancers peculiar to chimney sweeps, the first description of an occupational cancer (Brown and Thornton 1957). More than a century later, in 1914, by perturbing fertilized sea urchin eggs, Theodor Boveri developed the theory that damage to the nucleus of the cell resulted in ‘aberrant mitoses’ that led to malignancy, a prelude to what would become known as DNA damage and mutation (Boveri 2008). Though the elevation of site-specific cancer risk due to exogenous DNA damage is widely accepted within the oncology community, the importance of such

damaging agents on rates of aging remains unsettled. It is certain that some tissues are more impacted by exogenous sources of damage than others. For instance, genetic damage due to UV-radiation, often in the form of cyclobutane pyrimidine dimers and 6-4 photoproducts, is over-represented in skin cells in sun-exposed regions (Martincorena *et al.* 2015). Similarly, the cells of the gut are uniquely challenged by exogenous DNA damaging agents found in food such as polycyclic aromatic hydrocarbons (PAHs) and heterocyclic amines (HCAs), which have been proposed to elevate spontaneous colorectal cancer risk (Helmus *et al.* 2013). Although exogenous DNA damage may contribute to aging in some tissues, it does not appear to be essential for the process. Tissues not obviously exposed to exogenous damaging agents still age. Far more relevant to aging may be endogenous sources of damage arising from the pervasive challenge of physical entropy.

As catalogued by Tomas Lindahl, an estimated 15,000 lesions arise per human cell per day due to the endogenous processes of hydrolysis, oxidation, and methylation (Lindahl 1993). In nearly each case, repair mechanisms exist to suppress the potential catastrophic effects. The greatest risk for genetic entropy are those forms of damage that induce mutagenic base changes. By far the most frequent DNA damaging events involve spontaneous hydrolysis of the N-glycosyl bond of purines (9,000/day). Pyrimidines are lost at a rate of 5% that of purines (450/day). Cells rapidly repair this damage through enzymes that participate in base excision repair (BER). Apurinic/apyrimidinic (AP) endonuclease initiates repair by cleaving the phosphodiester backbone on the 5' side of the missing base. A second phosphodiesterase cleaves the backbone 3' of the lesion and DNA polymerase and ligase then restore the missing base. Spontaneous hydrolysis also occurs in the form of deamination of the C4 position of cytosine (50/day), converting the base to uracil. The ability of uracil to template insertion of dA means this lesion poses a risk for C>T transition mutations. However, cells suppress this potential mutagenesis through uracil-DNA glycosylase, which efficiently removes the damaged base, creating an abasic site for AP endonuclease. Deamination of the C4 position also occurs at 5-methylcytosine residues, converting the base to thymine (5/day); however, this lesion is not recognized by BER. While MMR can correct the T-G mismatch, repair is less efficient, giving time for DNA replication to make the genetic change permanent. Even if the mismatch is detected, since MMR typically repairs the nascent strand, a permanent double-stranded mutation may still occur if the Thymine resides on the template strand. Oxidative damage from superoxide ($O_2{\cdot}^-$) and hydroxyl radicals ($OH\cdot$) arise primarily as by-products of respiration and induce a range of lesions in DNA. The most well known oxidative lesion is 8-oxoguanine, which results when a hydroxyl radical reacts with guanine in double-stranded DNA (500-1000/day). If not removed before the ensuing round of replication, 7,8-Dihydro-8-oxoguanine (8-oxoguanine) may template misinsertion of dATP, resulting in a G>T transversion. Oxidation of dGTP also occurs, which can be misincorporated opposite of dA. In this case, an A>C transversion is possible if not repaired. Cells deploy a coordinated set of defense mechanisms to prevent genetic entropy from 8-oxoguanine. First, cells detoxify free radicals beginning with conversion of superoxide to

hydrogen peroxide (H_2O_2) by superoxide dismutase. Catalase then converts H_2O_2 to H_2O and O_2 to limit the production of hydroxyl radicals through the breakdown of hydrogen peroxide by the Fenton reaction. Second, cells limit incorporation of 8-oxo-dGTP into DNA by rapidly degrading it to 8-oxo-dGMP through a nucleoside triphosphatase that specifically recognizes oxidized purines (MutT in *E. coli*). Third, in most organisms, adenosine paired with 8-oxo-dG in duplex DNA is excised by a specific DNA glycosylase, encoded by MutY gene in *E. coli*. A redundant pathway involves the Msh2-Msh6 MMR complex, which catalyzes the removal of dA opposite 8-oxo-dG. Finally, oxoguanine glycosylase (OGG) specifically excises 8-oxo-dG paired with cytosine, allowing an undamaged dGTP to be inserted. Oxidation of pyrimidines also occurs. The most relevant for genetic entropy is formation of deoxycytosine glycol (dCg), which occurs with similar frequency as 8-oxo-dG (Wagner *et al.* 1992). Deamination and/or dehydration of this unstable species yields three oxidative lesions observed in DNA: 5-OH-dC, 2′-deoxyuridine glycol (dUg), and 5-OH-dU. dUg and 5-OH-dU promote C>T transitions if not repaired (Wang *et al.* 1998). The final category of assault on DNA through physical entropy occurs through nonenzymatic methylation of DNA. Reaction of DNA with the common enzymatic cofactor, S-adenosylmethionine (SAM), produces substantial numbers of 7-methylguanine and 3-methyladenine adducts (~4000/day). While the former appears benign, the latter is cytotoxic since it blocks DNA replication. The cellular response involves 3-methyladenine-DNA glycosylase, which efficiently removes the damaged base, again creating an abasic substrate for AP-endonuclease. Thus, for nearly every assault of physical entropy, life has evolved a solution. Nevertheless, the main mutagenic forms of physical entropy emerge from Lindahl's work including oxidation of guanine to 8-oxoguanine, oxidation of cytidine to cytidine glycol, deamination of cytidine to uracil, and deamination of 5-methylcytidine to thymine.

DNA lesions represent only a fraction of events that contribute to genetic entropy. Duplication of the genome can lead to base-substitution errors, frameshifts, larger indels, and even aneuploidy. The fidelity of nuclear DNA replication depends on DNA polymerases (Pol) ε and δ, which copy the leading and lagging strands, respectively. Biochemical estimates for the nucleotide selectivity of their polymerase domains range between 10^{-4} to 10^{-5} misinsertion errors/nucleotide synthesized (Thomas *et al.* 1991; Hashimoto *et al.* 2003; McCulloch and Kunkel 2008). Error rates at specific sites in the genome may be higher. As extrapolated to the genome by Preston et al., this means that cells contend with between 10^5 to 10^6 misinsertion errors each division, far more than the number of lesions due to spontaneous damage (Preston *et al.* 2010). Cells stem this potential tidal wave of genetic entropy through the synergistic action of polymerase exonuclease 'proofreading' and mismatch repair (MMR), which cooperatively lower the replication error rate to less than a single error per 10^9 bps (Loeb 1991) Even with this low mutation rate, normal human cells sustain at least one DNA replication-derived mutation each division. These mutations begin arising from the first divisions of the embryo, leading to somatic mosaicism, and continue to accumulate throughout development. Most cell lineages will continue to accumulate genetic entropy

by this route in a manner dependent on their replicative index. The highest genetic entropy due to replication errors will be found in highly replicative tissues such as colonic epithelium and endometrial cells. In cells, such as neurons, that settle into a non-mitotic state the most important sources of genetic entropy will be DNA damage and repair.

Mutations in genes involved in preventing mutations amplify the likelihood of future errors. These 'mutator' alleles were first predicted by Larry Loeb in 1974 as a possible explanation as to why cancers seem to be more frequent than expected given the relative stability of DNA (Loeb *et al.* 1974).The first recognized mutator-driven familial cancer was confirmed with the linking of Lynch syndrome to defects in MMR genes (Fishel *et al.* 1993; Peltomaki *et al.* 1993). Since that time, a number of mutations in genes involved in DNA replication and repair have been found to give rise to mutator phenotypes. Mutator phenotypes may also arise from transient dysregulation of AID/APOBEC cytosine deaminase, which is normally constrained to convert cytosine to uracil at sites of somatic hypermutation. Sequencing of cancer genomes reveals that off-target AID/APOBEC activity occasionally occurs and introduces localized mutagenesis in a process known as Kataegis (Alexandrov *et al.* 2013; Petljak *et al.* 2019). Cells with sustained or transient mutator phenotypes may persist and propagate in the absence of malignancy, accelerating genetic entropy among subclonal populations of somatic cells.

The repair processes cells use to remedy physical entropy can themselves be an unintuitive source of genetic entropy. Single and double-strand breaks of DNA (SSBs, DSBs) represent a critical dysfunction in nuclear DNA that must be resolved for a cell to survive and undergo replication. These may arise due to ionizing radiation as well as unligated nicks arising from a prior round of DNA replication or incomplete repair (Maizels and Davis 2018). Two broad categories of recombinational repair have evolved to address this threat, yet neither is without risk. The first, non-homologous end joining (NHEJ), has generally been considered a mutagenic process (Moore and Haber 1996). Evidence in recent years suggests some subtypes of NHEJ are less error-prone than previously indicated (Bétermier *et al.* 2014). What has become termed "classical NHEJ", or C-NHEJ, mediated in a Ku- dependent manner, is relatively faithful (Chen *et al.* 2010). C-NHEJ competes with an alternative and error-prone NHEJ process, inhibited by the presence of the enzymes involved with C-NHEJ (Wang *et al.* 2006). It is clear that this repair process, though effective at quickly repairing DSBs, is a potential source of mutagenesis and therefore a likely potential contributor to genetic entropy. The second broad category of recombination, homology-directed repair (HDR), resolves DSBs using an undamaged template. During the S-phase of the cell cycle, this process most often uses a sister-chromatid, resulting in no loss of information. If HDR utilizes a homologous chromosome instead, the sequence found on the homologue replaces unique information held on the damaged strand, resulting in Loss of Heterozygosity (LOH). Any deleterious mutation carried on the homologue, whether germline or somatically derived, will be made homozygous. Thus, despite the potential for error free repair by HDR, the threat of genetic entropy remains. The rate at which such events

occur and the effects of these events on human health remain relatively unexplored, though we note that cancers frequently exhibit signs of a large number of LOH tracts (Cavenee *et al.* 1983; Ryland *et al.* 2015). In some cases, LOH serves as the critical 'second hit' to the remaining wildtype allele in patients with germline mutations in tumor suppressors such as BRCA1 (Merajver *et al.* 1995). A special subtype of HDR, break-induced replication (BIR), occurs during stalled replication, in the event of two collinear DSBs, or other cases where only a single end of the DSB is available. BIR proceeds in a manner similar to other subtypes of HDR, but can stretch in length over thousands of kilobases to the ends of a chromosome arm. Further, the synthesis of DNA during BIR has been shown to be far less accurate than that associated with normal cell replication (Deem *et al.* 2011). The breadth of the repair tract and the error prone process make BIR a particularly important potential source for genetic entropy in aging somatic tissue.

The mitochondrial genome is subject to genetic entropy just like the nuclear genome, yet with three crucial differences that merit a mention. The first such difference is significant copy number variation. A single human cell will typically contain only two copies of the nuclear genome, with some variation to this among cardiomyocytes, skeletal muscle and liver cells, yet any of these same cells may contain several thousand copies of the mitochondrial genome (Miller *et al.* 2003). This creates a unique means for the retention of mutation-derived genetic misinformation, as deleterious mutations in the mitochondrial genome are buffered by the numerous other extant copies, allowing the persistence of such mutations in the cell's lineage and the potential buildup of aberrant proteins encoded by these mutants. The second significant difference between the nuclear and mitochondrial genomes is the proximity of the latter to the electron transport chain, lack of protective histones, and consequential exposure to reactive oxygen species (ROS). ROS can inflict oxidative damage to DNA as well as to lipids and proteins, and mitochondria are the greatest producers of ROS in the cell (Finley and Haigis 2009). This renders the mitochondrial genome particularly vulnerable to such ongoing insults. The final notable difference between the nuclear and mitochondrial genome is the lack of recombination to bypass Muller's Rachet. Mitochondrial inheritance is famously uniparental suggesting that the mitochondrial genome should inevitably accumulate mutations over successive cellular generations. This effect is likely compounded further by the relatively high mutation rate of mitochondrial DNA, and the low rate of recombination (Denver *et al.* 2000).

Random changes to the epigenetic state of the nuclear genome over the course of a lifespan also contributes to genetic entropy. The epigenome, defined by the patterning of DNA and histone modifications that affect the structure and accessibility of genetic information, is a deeply complex and carefully maintained information structure. Much as DNA damage-induced mutations alter the original information structure of the genome, aberrant alterations to the epigenomes that affect chromatin state and transcription may similarly be interpreted as a form of damage. Changes to DNA methylation state have been observed to increase with aging. In fact, the accumulation of DNA methylation changes at CpG islands occurs so consistently with chronological age that these changes have even been interpreted as clock-like processes (Hannum *et*

al. 2013; Horvath 2013). DNA methylation clocks use a select set of CpG islands whose change in methylation state is suspected to correlate with chronological age. These changes to methylation state describe a type of epigenetic drift in which the broadly similar young epigenome of individuals increasingly diverge from each other as they age. Each clock that has been developed varies according to the set of CpG islands and the tissue and analysis types used in its construction in an attempt to best describe the degree of that divergence (Bell *et al.* 2019). Though the development of an epigenetic clock that can perform as a reliable predictive biomarker for human aging is still an area of active research (Jylhävä *et al.* 2017), there is strong evidence to support a correlation between epigenetic drift of this kind and aging (Chen *et al.* 2016). Other possible types of damage may include the loss of components that compose the epigenome. In general, tightly packed, histone wrapped heterochromatin DNA is transcriptionally silent and relatively protected from damage. Loss of these important proteins has been observed to accelerate in budding yeast during replicative aging and upregulation of these components extends lifespan (Feser *et al.* 2010; Hu *et al.* 2014). As research on the epigenome has developed over the past two decades it has become increasingly clear that the epigenome must be considered an important component of gene expression and therefore merits inclusion in the genetic entropy theory.

The correlative link between epigenetic noise and aging would also indicate a connection between information loss and aging. Early research on the topic revealed that divergence of the epigenome of human monozygotic twins appears early in life (Fraga *et al.* 2005). At its core, it should not be surprising that altering a feature that controls gene expression is capable of altering lifespan, if after all, lifespan is a product of factors that include the variable expression of genes. And indeed, targeting the known lifespan altering IGF1 pathway for instance via knockdown of the H3K27me3 demethylase UTX- 1 can increase the lifespan of affected *C. elegans* animals (Jin *et al.* 2011). More recently, hints have begun to emerge that indicate that centenarians have a more stable epigenome than other elderly individuals as revealed through sequencing of cell-free DNA (Teo *et al.* 2019). Recently, reversal of the previously alluded to ‘clock’ of DNA methylation by overexpression of several key genes has been shown to restore the regenerative capacity of retinal ganglion cells in adult mice and following injury (Lu *et al.* 2020). The reversible nature of epigenetic post-translational changes makes it an attractive target for researchers seeking to test if undoing age-associated damage in this context ameliorates some of the deleterious effects of aging.

The sources of genetic entropy discussed thus far have involved changes to DNA sequence or epigenetic status. The randomness of the events found in each cell or cell lineage imparts a uniqueness to their dysfunction. Not all cells need to experience the same defect for genetic entropy to contribute to cell attrition and thereby to aging. And yet, could there be a unified endpoint to genetic entropy? Information flows from DNA to RNA to protein and the idea of genetic entropy may occur at each level. Leslie Orgel first proposed in 1963 that the phenomenon of aging could be due to an “error catastrophe”, in which random errors in protein synthesis that affected the accuracy of translation led to an exponential cascade of additional errors that undermined cell

viability (Orgel 1963). When new information emerged that the accuracy of protein synthesis was greater than Orgel had first imagined, he corrected his theory, noting that error catastrophe did not seem inevitable with age (Orgel 1970). Could genetic entropy at the level of DNA and RNA prime a protein error catastrophe or a loss of proteostasis, a recognized hallmark of aging (López-Otín *et al.* 2013)? Experiments delineating the range and frequency of spontaneous errors in RNA synthesis across the genome suggest that random errors in RNA synthesis impact the proteome (Vermulst *et al.* 2015; Gout *et al.* 2017; Fritsch *et al.* 2021). In wildtype yeast cells the error rates vary depending on the RNA polymerase. For RNA Pol II transcripts, the average error rate is 3.9×10^{-6} errors/bp (Gout *et al.* 2017). Since the errors randomly affect transcripts they do not confer a null phenotype in any one gene. But increasing the noise in the form of mutant proteins induces proteostatic stress. A 10-fold elevation of Pol II transcriptional errors in budding yeast due to deletion of *rpb9* markedly shortens replicative lifespan (Gout *et al.* 2017) and saturates the protein quality control machinery so that unstable peptides that are normally degraded become stabilized (Vermulst *et al.* 2015). The energy investments needed to ensure accurate DNA replication, transcription, translation, and proteostasis are expensive. As first proposed by Thomas Kirkwood, in multicellular organisms, these investments in somatic cells are likely balanced with energy expenditures that maximize the success of the germline (Kirkwood 1977, 2005). If so, thresholds may exist for each form of genetic entropy that are not static, but dynamically influenced by the degree of genetic entropy existing at other levels.

A Threshold of Genetic Entropy for Aging

In Szilard's model for aging, loss of entire chromosomes led to lethal LOH events that drove the attrition of somatic cells (Szilard 1959). As detailed above, loss of genetic information takes many forms. We propose that genetic entropy — whether inherited mutations, somatic mutations, epigenetic changes, and even errors in transcription or translation — collectively leads to a loss of organismal homeostasis. Evidence exists for a threshold effect of mutagenic changes to DNA. For instance, long standing correlations exist between aging and somatic mutation accumulation. Both the well documented Ames dwarf mouse and calorically restricted mice exhibit a decreased mutation frequency and a long lifespan relative to their wildtype controls (Garcia *et al.* 2008). Correspondingly, mutation frequency increases in an accelerated senescence mouse model that experiences an abbreviated lifespan (Odagiri *et al.* 1998). Many of the progeroid type diseases specifically involve mechanisms that in some form disrupt the nucleus or genome stability and maintenance. Hutchinson-Gilford progeria syndrome, caused by mutations in the LMNA gene, results in the production of a toxic alternative splicing product, progerin, with deleterious effects on the nuclear membrane (Eriksson *et al.* 2003). Other syndromes are due to defects in the enzymes responsible for the repair of various common DNA lesions. These syndromes often present as segmental progeroid diseases, characterized by an acceleration of a subset of aged phenotypes, which suggests that the requirement for particular DNA repair pathways varies between tissues. Bloom and Werner syndromes are caused by defects in RecQ

helicases that participate in recombinational repair. Similarly, the segmental progeroid disease ataxia-telangiectasia (AT), an autosomal recessive disorder, is caused by a mutation in the AT-mutated (ATM) gene. The ATM protein serves the vital function of coordinating with ATM and Rad3-related protein (ATR) to respond to DSBs and coordinate repair(Lavin 2008), and loss of function of this gene also impairs G1-S phase checkpoint signaling that involves coordination with p53 (Khanna and Lavin 1993). As another example, the unimodal progeroid syndrome Xeroderma pigmentosum, characterized by UV hypersensitivity and early onset of a variety of age-associated skin neoplasms, is caused by one of a handful of dysfunctions in either the proteins involved in nucleotide excision repair processes or DNA polymerase η (Karass *et al.* 2014). These examples are all defects in some form of genomic integrity and all produce degenerative phenotypes characteristic of aging. However, using these progeria models to infer a mutagenic threshold for aging is problematic due to the difficulty in uncoupling the effects of genetic entropy from the consequences of unrepaired lethal DNA damage.

Several mouse models specifically increase mutagenesis through defects in DNA replication fidelity and provide a more direct test of where the mutagenic threshold may lie for genetic entropy-induced aging. Mice with defective mitochondrial DNA polymerase proofreading exhibit a strong segmental progeroid syndrome (Trifunovic *et al.* 2004). Though these mice retain mitochondrial DNA replication activity, the loss of exonuclease activity resulted in the rapid accumulation of mutations in the mitochondrial genome. At only 25 weeks of age, the mice display pathologies strikingly similar to those of animals of much more advanced age including cardiomyopathy, anaemia, kyphosis, alopecia and reduced overall body size. It's worth noting, however, that wildtype mice do not seem to accumulate enough mitochondrial single nucleotide variants (SNVs) in their lifetime for this mechanism to be the sole driver of natural aging in the species (Vermulst *et al.* 2007). Mice engineered with proofreading defects affecting the nuclear DNA polymerases, Polδ or Polε, produce cells with an elevated mutation rate. The mice display a striking increase in cancer incidence in a range of tissues, with median lifespans of 10 and 16 months, respectively, compared to 26 months for WT littermate controls (Goldsby *et al.* 2001, 2002; Albertson *et al.* 2009). Both mutator strains develop nodal and follicular lymphomas as well as intestinal and lung tumors. Non-overlapping tumor types are also observed suggesting distinct hotspots of mutagenesis or participation in different repair processes: Polδ proofreading-deficient mice develop T-cell lymphomas and squamous papilloma tail skin tumors, while Polε mutant mice frequently develop histiocytic sarcomas. Since cancer is a disease of aging, these mice can be classified as displaying a unimodal progeroid syndrome. The mice do not exhibit generalized signs of accelerated aging during their otherwise abbreviated lifespan. Whether their cancer mortality masks a modest acceleration of other age-related phenotypes remains undetermined. However, further increasing genetic entropy by crossing either of the two proofreading-defective mutants with an MMR-mutant resulted in a striking lethal phenotype in the developing double homozygous mutant embryos. Both genotypes began development at the expected Mendelian ratios, but $Pold1^{exo/exo}$ $MMR^{\Delta/\Delta}$ embryos ceased developing by day E9.5 and

Pole$^{exo/exo}$ *MMR*$^{\Delta/\Delta}$ embryos by day E16. These developmental failures are consistent with a threshold for mutation accumulation that causes a catastrophic loss of replicative capacity. Experiments with diploid yeast mutator strains have since established clear evidence for a threshold of replication errors for extinction of a dividing population of cells (Herr *et al.* 2014).

We recently explored the degree of genetic entropy required to specifically accelerate aging of individual cells with the yeast *S. cerevisiae* (Lee *et al.* 2019). Budding yeast cells divide asymmetrically into mother and daughter cells and the number of divisions a mother undergoes before senescence is defined as the replicative lifespan. Due to frequent mating between siblings and purifying selection, wild type diploid yeast carry few inherited deleterious heterozygous alleles. Moreover, new mutations occur too infrequently in yeast to explain replicative aging, with at most a single mutation in a complete replicative lifespan (Kaya *et al.* 2015). Since mutation accumulation does not normally drive yeast aging, we viewed yeast as an ideal model for exploring the level of mutagenesis required to compromise lifespan. We engineered diploid yeast strains with heterozygous mutations affecting MMR (*msh6Δ)* and polymerase proofreading (*POL/pol MSH6/msh6Δ)* and then randomly mated haploid spores to obtain zygotes with different genotypes, conferring mutation rates that varied over three orders of magnitude. Following isolation, the diploid strains were propagated briefly, clonally isolated to bottleneck the mutations, propagated again, and then analyzed for lifespan. Colonies from representative cells of each cohort were also sequenced to determine mutation load. In a separate experiment, we used a transient mutator phenotype to obtain strains with an increased mutation burden in the absence of an ongoing mutator phenotype.

For most strains, lifespan steadily decreased in a log-linear fashion as mutation burden increased. Some variation in replicative lifespan between independent isolates of the same genotype and mutation burden was observed, suggesting that specific mutations or combinations of mutations accelerated aging in each strain. Notably, 2/14 strains with mutation burdens of only 1 mutation/Mb displayed a 25% reduction in median lifespan, which could be traced to just a few inherited heterozygous SNVs, underscoring the power of even modest genetic entropy to impact lifespan. Active mutators with 20-30 mutations/Mb consistently exhibited reductions in median lifespan of between 25-35%. Similar reductions were observed in nonmutator strains with 12-99 mutations/Mb, confirming that genetic entropy apart from an active mutator is sufficient to drive aging. A dramatic departure from the above trend was observed with strains with the strongest mutator phenotype and highest mutation burdens (100 SNVs/Mb), which showed a uniform 90% reduction in median lifespan. To understand the basis of this apparent threshold, we measured the lifespan of the first daughters of freshly isolated zygotes with this genotype, before extensive mutation accumulation had occurred. Lifespan of the initial mutator mother cells was comparable to the parent strain. Later descendents of these mutator cells, however, displayed the same profound reduction in lifespan. The acceleration in aging likely derives from the strong mutator phenotype of these cells (~60 mutations/division) (Dowsett *et al.* 2021) synergizing with

the mounting mutation burden. Taken together our observations in yeast indicate that genetic entropy has an escalating impact on replicative potential. The reductions in lifespan do not always correlate perfectly with mutation burden. Chance, of course, plays an important part in the acquisition of deleterious alleles. Selection may also favor cell lineages with less deleterious mutation burdens during growth. Similar selective forces acting during development and homeostasis likely influence the mutation burden found in human cells in aging tissues.

It remains unclear if some progeroid models are in fact etiologically accelerating what happens during normal aging or if they simply represent independent pathways to a phenocopying an aged phenotype. It is also worth noting that no progeroid syndrome fully accelerates the onset of the complete panel of age-associated disorders. To determine whether genetic entropy truly plays a role in metazoan aging will require careful quantification of the degree of genetic and epigenetic changes in individual aged cells. Research has recently begun to experimentally define the degree and range of mutation burdens found in both replicative and post-mitotic aged human tissues (Lodato *et al.* 2017; Martincorena *et al.* 2018). Key to this endeavor is the adaptation of modern sequencing techniques to yield accurate single cell resolution data.

The precise degree to which any two cells with the same initial function will differ due to genetic entropy is a function of the number of divisions separating them and rate of information loss. Cellular compartment, organ system and location within the body will affect which sources of damage are likely to dominate the mutational landscape. This relationship is further complicated by any of the myriad of possible selective advantages or disadvantages that will affect the competitive survival of a cell's mitotic progeny within a tissue. Stem cells undergoing active replication throughout an individual's lifespan are subject to a different set of genetic information challenges than long-lived cells that are essentially post-mitotic following development, such as neurons or cardiomyocytes, that must remain functional for decades. With both classes of cells, understanding the degree of genetic entropy required for aging is important as is understanding how divergent genetic identity of individual cells within a tissue affects broader tissue function.

Single Cell Resolution Studies: Quantifying Genetic Entropy in Aging.

The degree to which mutations accumulate in aged tissues has been of great interest to researchers for several decades. The challenge from the beginning has been discerning the mutation load of individual cells. Early approaches began by selecting for mutations in a reporter gene, such as HPRT, to show that mutation frequency was higher in cells isolated from aged individuals (Morley et al 1982; Trainor, 1984; Curry et al, 1999). The requirement to establish cell lines *ex vivo* was cumbersome and limited the approach to cells that could easily be cultured. The development of transgenic mice carrying LacZ reporter arrays allowed researchers to explore the frequencies of point mutations and genome rearrangements with aging in vivo in a wide variety of tissues (Dollé *et al.* 2000; Dollé and Vijg 2002). While the reporter genes gave useful insights, it wasn't clear whether observations made with a large transgenic array were representative of the rest of the genome. Since then, the development of massively

parallel short-read sequencing has revolutionized our ability to understand genetic changes occurring across the genome. The Cancer Genome Atlas (TCGA) project primarily used exome sequencing to reveal the clonal mutation burden present in cancers. Since tumors begin with malignant transformation of a single cell, these results provide an indirect genome-wide picture of the genetic entropy present within individual cells during aging. It is clear from these studies that the somatic mutation burden in tumors increases with patient age (Milholland *et al.* 2015). The exomes of tumors from patients over 80 years old exhibit a six-fold increase in mutation frequency (median 2.21 mutations/MB) compared to the tumors derived from patients under 20 years of age (median 0.37 mutations/MB). The mutation burden of exomes may underestimate the total mutation burden of the tumors, yet even so the mutation burden found in aged tumors is within the minimum range that imparts a negative effect on cellular lifespan in yeast. Understanding aging by peering through the “cracked lens” of cancer genomes (Zuccala 2016), although useful, has limitations. The mutations represent not only those that arose during normal aging but also during selection for malignant phenotypes, where cells with more mutations are favored.

Traditional whole genome sequencing, in which a large number of cells are used for library construction, is ideal for studying populations of cells in which the vast majority of those cells share variants of interest and thus can be used to identify commonality. However, research questions involving highly heterogeneous populations, such as querying the variability of genetic entropy of somatic tissues in elderly individuals, are interested in questions of exceptionality - rare mutations present in as little as one cell within the population. Sequencing DNA at great depth could in theory reveal such rare mutations. However, standard Illumina sequencing, the most widely used approach, has a false positive rate of 10^{-2}. Errors due to PCR amplification or sequencing methodology can be dramatically decreased by incorporating random barcodes into the Illumina sequencing adapters used to clone the DNA libraries. The most successful of these approaches, Duplex Sequencing, begins by performing a targeted capture of a limited portion of the genome (Kennedy *et al.* 2014). Following library construction with dual random barcodes and several PCR amplification cycles, the library is sequenced at great depth so that multiple related molecules derived from the same cloned fragment are read. The Single strand consensus (SSC) of sequences with the same random index are determined computationally and then a duplex consensus is generated, lowering the artifactual error rate below 10^{-8}/bp sequenced. As a proof of principle, Kennedy et al., utilized this approach to show how aging increases genetic entropy in the mitochondria of human brains. Although Duplex Sequencing provides unparalleled sensitivity for reliably scoring low frequency mutations present in only a fraction of a population of molecules it fails to give a genome wide understanding of the mutation load in individual cells.

Finding genuine rare genetic variants possessed by only a single cell in a population necessitates single cell sequencing (SCS). Since being declared by Nature Methods as the Method of the Year in 2013 (“Method of the Year 2013” 2014), a host of SCS methods have been advanced to bypass the limits of traditional whole genome sequencing and achieve reliable single cell resolution genomic analysis. This task is complicated by two technological challenges which make SCS a more technically

demanding task than whole genome sequencing of populations. The first of these is the technical difficulty of isolating single cells. This can be accomplished by a variety of approaches, most commonly by flow cytometry assisted cell sorting or manual cell manipulation. The second challenge, that of whole genome amplification (WGA) is, in many SCS approaches, the most difficult and crucial step in the process. The difficulty lies in amplifying the genome of the cell without introducing and amplifying errors that will then incorrectly be scored as mutations, while simultaneously achieving high quality coverage of a substantial portion of the genome. Several variations of WGA have been developed and applied to single cell sequencing. Many of these approaches use inventive biochemistry techniques to sequester or by other means prevent newly synthesized products from replicating during the initial rounds of amplification, thus reducing the chance of error accumulation. Multiple displacement amplification (MDA), for instance, uses random hexamer primers and a φ29 polymerase with strong DNA displacement properties under isothermal conditions (Blanco *et al.* 1989), to produce a nest of branched genetically identical fragments using genomic DNA as the sole template. In an effort to further reduce the amount of nonlinear amplification observed in MDA, Zong *et al* developed a multiple annealing and looping based amplification cycles (MALBAC) approach (Zong *et al.* 2012). MALBAC uses primers with 27-base complementary ends to form stable loop structures that ensure only genomic DNA is copied in the first several rounds of thermal-cycling. Another set of approaches that have been adapted for use in SCS include variations of degenerate oligonucleotide-primed polymerase chain reaction (DOP-PCR) (Telenius *et al.* 1992). DOP-PCR approaches use primers containing a small random sequence as well as a larger fixed sequence and low annealing temperatures to drive non-selective amplification of the genome. In general, however, DOP-PCR approaches for SCS suffer from poor coverage and allele dropout (Huang *et al.* 2015).

A growing body of literature suggests that these new SCS approaches are gaining acceptance. An early SCS study utilized SCS to demonstrate that punctuated rather than gradual changes to copy number occur early in tumor evolution in triple-negative breast cancer tumors (Gao *et al.* 2016). This study used Fluorescence-Activated Cell Sorting (FACS) to isolate nuclei and DOP-PCR WGA to sequence a thousand individual cells from twelve patients to achieve these results. An aging focused SCS study by Jan Vijg's group utilized MDA to explore mutation accumulation in liver cells, which detoxify a wide range of chemicals from the bloodstream. Their findings support an exponential increase in mutations accumulation in single hepatocytes between young and old individuals with mutation burdens increasing from 0.2 mutations/MB in a five-month-old to 0.7 mutations/MB in cells derived from a 77-year-old (Brazhnik *et al.* 2020). Notably, they also observed several significant outliers across multiple ages at upwards of 10-fold higher than their age-matched counterparts, endorsing the likelihood of mutator phenotypes emerging in somatic populations during normal aging. Such findings would be impossible to make when studying populations of cells rather than the genome of a single cell due to the averaging of effects of numerous cell divisions. A final illustrative study utilizing SCS methodology focused on understanding the mutation burden of post-mitotic cells. As part of his classic 1993

review delineating the many sources of DNA damage and decay, Thomas Lindahl predicted that neurons and stem cells might quietly accumulate mutations over a lifetime, which would only reach a critical detrimental threshold well after reproductive maturity of the individual (Lindahl 1993). In 2017, Lodato et al were able to examine the mutation burden of single mature neurons using a MDA-based WGA approach (Lodato *et al.* 2017). They found an age-dependent linear increase in mutation burden in neurons and mutational signatures associated with such insults as oxidation and defects in DNA damage repair, with levels of mutation accumulation similar to what Vijg's group found in hepatocytes.

The above examples demonstrate the many opportunities for SCS to test decades old predictions. The limitation of these approaches remains the high cost of sequencing whole genomes. Strong models of mutation accumulation within malignant and aged tissues require high density distributions of mutation count data from hundreds of cells from the same tissue. Although sequencing costs continue to come down, such high resolution models will require new innovations that dramatically reduce costs even further. Within both types of cellular populations there is a great deal of potential for loss of homogeneity as genetic entropy accumulates. For a complete understanding of the contribution of genetic entropy to aging will also require ways of quantifying epigenetic drift within the same cell populations. A further area yet to explore is how much genetic entropy of individual cells is required to appreciably diminish overall tissue function.

As I have shown above, from prokaryotes to metazoans, life staves off the effects of genetic entropy, in part, through the highly precise and error-correcting mechanisms that maintain DNA replication fidelity. Based on a large body of research over the last few decades, we understand a myriad of ways that genetic entropy occurs, and the new SCS methods described above provide the tools to understand the extent to which this varies between cells of the same tissue. In the following chapters, I use a different inferential method of SCS to address several critical questions to genetic entropy and the ways DNA replication fidelity can be perturbed that can only be fully examined at single cell resolution. Our approach allows live yeast cells to amplify their own genome through normal colony formation. Inferential methods discard the need for highly precise WGA but necessitate another method of error-correcting. In our case, for instance, we use a colony derived from an immediate descendent to correct for genetic changes that might occur during colony outgrowth. We use our method to examine mutator volatility, the dramatic deviation of the mutation rate across individual divisions in strong mutators and an alternative hypothesis that mutation rate variation is a consequence of unequal sharing of mismatches between mother and daughter cells. We then apply these same techniques to examine if mutator volatility is a product of S-phase checkpoint signaling and corresponds to changes in intracellular dNTP levels. Lastly, we test a novel method of identifying loss of heterozygosity events at the single cell and genome wide levels in yeast and address the question of whether such events increase dramatically at the far reaches of replicative age. As discussed earlier, such events can catalyze the transition of silent heterozygous mutations or variants into deleterious homozygous alleles with potentially disastrous consequences at the cell, tissue, or organism levels.

Chapter One Bibliography

Albertson T. M., M. Ogawa, J. M. Bugni, L. E. Hays, Y. Chen, *et al.*, 2009 DNA polymerase ε and δ proofreading suppress discrete mutator and cancer phenotypes in mice. Proc National Acad Sci 106: 17101–17104. https://doi.org/10.1073/pnas.0907147106

Alexandrov L. B., S. Nik-Zainal, D. C. Wedge, S. A. J. R. Aparicio, S. Behjati, *et al.*, 2013 Signatures of mutational processes in human cancer. Nature 500: 415–421. https://doi.org/10.1038/nature12477

Bell C. G., R. Lowe, P. D. Adams, A. A. Baccarelli, S. Beck, *et al.*, 2019 DNA methylation aging clocks: challenges and recommendations. Genome Biol 20: 249. https://doi.org/10.1186/s13059-019-1824-y

Bétermier M., P. Bertrand, and B. S. Lopez, 2014 Is Non-Homologous End-Joining Really an Inherently Error-Prone Process? Plos Genet 10: e1004086. https://doi.org/10.1371/journal.pgen.1004086

Blanco L., A. Bernad, J. M. Lázaro, G. Martín, C. Garmendia, *et al.*, 1989 Highly efficient DNA synthesis by the phage phi 29 DNA polymerase. Symmetrical mode of DNA replication. J Biological Chem 264: 8935–40.

Boveri T., 2008 Concerning the Origin of Malignant Tumours by Theodor Boveri. Translated and annotated by Henry Harris. J Cell Sci 121: 1–84. https://doi.org/10.1242/jcs.025742

Brazhnik K., S. Sun, O. Alani, M. Kinkhabwala, A. W. Wolkoff, *et al.*, 2020 Single-cell analysis reveals different age-related somatic mutation profiles between stem and differentiated cells in human liver. Sci Adv 6: eaax2659. https://doi.org/10.1126/sciadv.aax2659

Brown J. R., and J. L. Thornton, 1957 Percivall Pott (1714-1788) and Chimney Sweepers' Cancer of the Scrotum. Brit J Ind Med 14: 68. https://doi.org/10.1136/oem.14.1.68

Cavenee W. K., T. P. Dryja, R. A. Phillips, W. F. Benedict, R. Godbout, *et al.*, 1983 Expression of recessive alleles by chromosomal mechanisms in retinoblastoma. Nature 305: 779–784. https://doi.org/10.1038/305779a0

Chen J.-M., D. N. Cooper, C. Férec, H. Kehrer-Sawatzki, and G. P. Patrinos, 2010 Genomic rearrangements in inherited disease and cancer. Semin Cancer Biol 20: 222–233. https://doi.org/10.1016/j.semcancer.2010.05.007

Chen B. H., R. E. Marioni, E. Colicino, M. J. Peters, C. K. Ward-Caviness, *et al.*, 2016 DNA methylation-based measures of biological age: meta-analysis predicting time to death. Aging 8: 1844–1865. https://doi.org/10.18632/aging.101020

Deem A., A. Keszthelyi, T. Blackgrove, A. Vayl, B. Coffey, *et al.*, 2011 Break-Induced Replication Is Highly Inaccurate. Plos Biol 9: e1000594. https://doi.org/10.1371/journal.pbio.1000594

Delneri D., D. C. Hoyle, K. Gkargkas, E. J. M. Cross, B. Rash, *et al.*, 2007 Identification and characterization of high-flux-control genes of yeast through competition analyses in continuous cultures. Nat Genet 40: ng.2007.49. https://doi.org/10.1038/ng.2007.49

Denver D. R., K. Morris, M. Lynch, L. L. Vassilieva, and W. K. Thomas, 2000 High Direct Estimate of the Mutation Rate in the Mitochondrial Genome of *Caenorhabditis elegans*. Science 289: 2342–2344. https://doi.org/10.1126/science.289.5488.2342

Dollé M. E. T., W. K. Snyder, J. A. Gossen, P. H. M. Lohman, and J. Vijg, 2000 Distinct spectra of somatic mutations accumulated with age in mouse heart and small intestine. Proc National Acad Sci 97: 8403–8408. https://doi.org/10.1073/pnas.97.15.8403

Dollé M. E. T., and J. Vijg, 2002 Genome Dynamics in Aging Mice. Genome Res 12: 1732–1738. https://doi.org/10.1101/gr.125502

Dowsett I. T., J. L. Sneeden, B. J. Olson, J. McKay-Fleisch, E. McAuley, *et al.*, 2021 Rate volatility and asymmetric segregation diversify mutation burden in cells with mutator alleles. Commun Biology 4: 21. https://doi.org/10.1038/s42003-020-01544-6

Drake J. W., 1991 A constant rate of spontaneous mutation in DNA-based microbes. Proc National Acad Sci 88: 7160–7164. https://doi.org/10.1073/pnas.88.16.7160

Durbin R. M., D. Altshuler, R. M. Durbin, G. R. Abecasis, D. R. Bentley, *et al.*, 2010 A map of human genome variation from population-scale sequencing. Nature 467: 1061–1073. https://doi.org/10.1038/nature09534

Eriksson M., W. T. Brown, L. B. Gordon, M. W. Glynn, J. Singer, *et al.*, 2003 Recurrent de novo point mutations in lamin A cause Hutchinson–Gilford progeria syndrome. Nature 423: 293–298. https://doi.org/10.1038/nature01629

Feser J., D. Truong, C. Das, J. J. Carson, J. Kieft, *et al.*, 2010 Elevated Histone Expression Promotes Life Span Extension. Mol Cell 39: 724–735. https://doi.org/10.1016/j.molcel.2010.08.015

Finley L. W. S., and M. C. Haigis, 2009 The coordination of nuclear and mitochondrial communication during aging and calorie restriction. Ageing Res Rev 8: 173–188. https://doi.org/10.1016/j.arr.2009.03.003

Fishel R., M. K. Lescoe, M. R. S. Rao, N. G. Copeland, N. A. Jenkins, *et al.*, 1993 The human mutator gene homolog MSH2 and its association with hereditary nonpolyposis colon cancer. Cell 75: 1027–1038. https://doi.org/10.1016/0092-8674(93)90546-3

Fraga M. F., E. Ballestar, M. F. Paz, S. Ropero, F. Setien, *et al.*, 2005 Epigenetic differences arise during the lifetime of monozygotic twins. P Natl Acad Sci Usa 102: 10604–10609. https://doi.org/10.1073/pnas.0500398102

Fritsch C., J.-F. Gout, S. Haroon, A. Towheed, C. Chung, *et al.*, 2021 Genome-wide surveillance of transcription errors in response to genotoxic stress. Proc National Acad Sci 118: e2004077118. https://doi.org/10.1073/pnas.2004077118

Gao R., A. Davis, T. O. McDonald, E. Sei, X. Shi, *et al.*, 2016 Punctuated copy number evolution and clonal stasis in triple-negative breast cancer. Nat Genet 48: 1119–1130. https://doi.org/10.1038/ng.3641

Garcia A. M., R. A. Busuttil, R. B. Calder, M. E. T. Dollé, V. Diaz, *et al.*, 2008 Effect of Ames dwarfism and caloric restriction on spontaneous DNA mutation frequency in different mouse tissues. Mech Ageing Dev 129: 528–533. https://doi.org/10.1016/j.mad.2008.04.013

Giraud A., I. Matic, O. Tenaillon, A. Clara, M. Radman, *et al.*, 2001 Costs and benefits of high mutation rates: adaptive evolution of bacteria in the mouse gut. Science 291: 2606--2608. https://doi.org/10.1126/science.1056421

Goldsby R. E., N. A. Lawrence, L. E. Hays, E. A. Olmsted, X. Chen, *et al.*, 2001 Defective DNA polymerase-δ proofreading causes cancer susceptibility in mice. Nat Med 7: 638–639. https://doi.org/10.1038/88963

Goldsby R. E., L. E. Hays, X. Chen, E. A. Olmsted, W. B. Slayton, *et al.*, 2002 High incidence of epithelial cancers in mice deficient for DNA polymerase δ proofreading. Proc National Acad Sci 99: 15560–15565. https://doi.org/10.1073/pnas.232340999

Gout J.-F., W. Li, C. Fritsch, A. Li, S. Haroon, *et al.*, 2017 The landscape of transcription errors in eukaryotic cells. Sci Adv 3: e1701484. https://doi.org/10.1126/sciadv.1701484

Hannum G., J. Guinney, L. Zhao, L. Zhang, G. Hughes, *et al.*, 2013 Genome-wide Methylation Profiles Reveal Quantitative Views of Human Aging Rates. Mol Cell 49: 359–367. https://doi.org/10.1016/j.molcel.2012.10.016

Hashimoto K., K. Shimizu, N. Nakashima, and A. Sugino, 2003 Fidelity of DNA Polymerase δ Holoenzyme from Saccharomyces cerevisiae : The Sliding Clamp Proliferating Cell Nuclear Antigen Decreases Its Fidelity †. Biochemistry-us 42: 14207–14213. https://doi.org/10.1021/bi0348359

Helmus D. S., C. L. Thompson, S. Zelenskiy, T. C. Tucker, and L. Li, 2013 Red Meat-Derived Heterocyclic Amines Increase Risk of Colon Cancer: A Population-Based Case-Control Study. Nutrition Cancer 65: 1141–1150. https://doi.org/10.1080/01635581.2013.834945

Herr A. J., S. R. Kennedy, G. M. Knowels, E. M. Schultz, and B. D. Preston, 2014 DNA Replication Error-Induced Extinction of Diploid Yeast. Genetics 196: 677–691. https://doi.org/10.1534/genetics.113.160960

Horvath S., 2013 DNA methylation age of human tissues and cell types. Genome Biol 14: 3156. https://doi.org/10.1186/gb-2013-14-10-r115

Hu Z., K. Chen, Z. Xia, M. Chavez, S. Pal, *et al.*, 2014 Nucleosome loss leads to global transcriptional up-regulation and genomic instability during yeast aging. Gene Dev 28: 396–408. https://doi.org/10.1101/gad.233221.113

Huang L., F. Ma, A. Chapman, S. Lu, and X. S. Xie, 2015 Single-Cell Whole-Genome Amplification and Sequencing: Methodology and Applications. Annu Rev Genom Hum G 16: 1–24. https://doi.org/10.1146/annurev-genom-090413-025352

Jin C., J. Li, C. D. Green, X. Yu, X. Tang, *et al.*, 2011 Histone Demethylase UTX-1 Regulates C. elegans Life Span by Targeting the Insulin/IGF-1 Signaling Pathway. Cell Metab 14: 161–172. https://doi.org/10.1016/j.cmet.2011.07.001

Jylhävä J., N. L. Pedersen, and S. Hägg, 2017 Biological Age Predictors. Ebiomedicine 21: 29–36. https://doi.org/10.1016/j.ebiom.2017.03.046

Karass M., M. M. Naguib, N. Elawabdeh, C. A. Cundiff, J. Thomason, *et al.*, 2014 Xeroderma Pigmentosa: Three New Cases with an In Depth Review of the Genetic and Clinical Characteristics of the Disease. Fetal Pediatr Pathol 34: 120–127. https://doi.org/10.3109/15513815.2014.982336

Kaya A., A. V. Lobanov, and V. N. Gladyshev, 2015 Evidence that mutation accumulation does not cause aging in Saccharomyces cerevisiae. Aging Cell 14: 366–371. https://doi.org/10.1111/acel.12290

Kennedy S. R., M. W. Schmitt, E. J. Fox, B. F. Kohrn, J. J. Salk, *et al.*, 2014 Detecting ultralow-frequency mutations by Duplex Sequencing. Nat Protoc 9: nprot.2014.170. https://doi.org/10.1038/nprot.2014.170

Khanna K. K., and M. F. Lavin, 1993 Ionizing radiation and UV induction of p53 protein by different pathways in ataxia-telangiectasia cells. Oncogene 8: 3307–12.

Kirkwood T. B. L., 1977 Evolution of ageing. Nature 270: 301–304. https://doi.org/10.1038/270301a0

Kirkwood T. B. L., 2005 Understanding the Odd Science of Aging. Cell 120: 437–447. https://doi.org/10.1016/j.cell.2005.01.027

Lavin M. F., 2008 Ataxia-telangiectasia: from a rare disorder to a paradigm for cell signalling and cancer. Nat Rev Mol Cell Bio 9: 759–769. https://doi.org/10.1038/nrm2514

Lee M. B., I. T. Dowsett, D. T. Carr, B. M. Wasko, S. G. Stanton, *et al.*, 2019 Defining the impact of mutation accumulation on replicative lifespan in yeast using cancer-associated mutator phenotypes. Proc National Acad Sci 201815966. https://doi.org/10.1073/pnas.1815966116

Lindahl T., 1993 Instability and decay of the primary structure of DNA. Nature 362: 709–715. https://doi.org/10.1038/362709a0

Lodato M. A., R. E. Rodin, C. L. Bohrson, M. E. Coulter, A. R. Barton, *et al.*, 2017 Aging and neurodegeneration are associated with increased mutations in single human neurons. Science 359: eaao4426. https://doi.org/10.1126/science.aao4426

Loeb L. A., C. F. Springgate, and N. Battula, 1974 Errors in DNA replication as a basis of malignant changes. Cancer Res 34: 2311–21.

Loeb L. A., 1991 Mutator phenotype may be required for multistage carcinogenesis. Cancer Res 51: 3075--3079.

López-Otín C., M. A. Blasco, L. Partridge, M. Serrano, and G. Kroemer, 2013 The Hallmarks of Aging. Cell 153: 1194--1217. https://doi.org/10.1016/j.cell.2013.05.039

Lu Y., B. Brommer, X. Tian, A. Krishnan, M. Meer, *et al.*, 2020 Reprogramming to recover youthful epigenetic information and restore vision. Nature 588: 124–129. https://doi.org/10.1038/s41586-020-2975-4

Lynch M., 2010 Evolution of the mutation rate. Trends Genet 26: 345--352. https://doi.org/10.1016/j.tig.2010.05.003

Maizels N., and L. Davis, 2018 Initiation of homologous recombination at DNA nicks. Nucleic Acids Res 46: gky588. https://doi.org/10.1093/nar/gky588

Mao E. F., L. Lane, J. Lee, and J. H. Miller, 1997 Proliferation of mutators in A cell population. J Bacteriol 179: 417–422. https://doi.org/10.1128/jb.179.2.417-422.1997

Martincorena I., A. Roshan, M. Gerstung, P. Ellis, P. V. Loo, *et al.*, 2015 High burden and pervasive positive selection of somatic mutations in normal human skin. Science 348: 880–886. https://doi.org/10.1126/science.aaa6806

Martincorena I., J. C. Fowler, A. Wabik, A. R. J. Lawson, F. Abascal, *et al.*, 2018 Somatic mutant clones colonize the human esophagus with age. Science 362: eaau3879. https://doi.org/10.1126/science.aau3879

McCulloch S. D., and T. A. Kunkel, 2008 The fidelity of DNA synthesis by eukaryotic replicative and translesion synthesis polymerases. Cell Res 18: 148–161. https://doi.org/10.1038/cr.2008.4

Medawar P., 1951 An Unsolved Problem of Biology

Merajver S. D., T. S. Frank, J. Xu, T. M. Pham, K. A. Calzone, *et al.*, 1995 Germline BRCA1 mutations and loss of the wild-type allele in tumors from families with early onset breast and ovarian cancer. Clin Cancer Res Official J Am Assoc Cancer Res 1: 539–44.

Method of the Year 2013, 2014 Nat Methods 11: 1–1. https://doi.org/10.1038/nmeth.2801

Milholland B., A. Auton, Y. Suh, and J. Vijg, 2015 Age-related somatic mutations in the cancer genome. Oncotarget 6: 24627–24635. https://doi.org/10.18632/oncotarget.5685

Miller J. H., A. Suthar, J. Tai, A. Yeung, C. Truong, *et al.*, 1999 Direct Selection for Mutators in Escherichia coli. J Bacteriol 181: 1576–1584. https://doi.org/10.1128/jb.181.5.1576-1584.1999

Miller F. J., F. L. Rosenfeldt, C. Zhang, A. W. Linnane, and P. Nagley, 2003 Precise determination of mitochondrial DNA copy number in human skeletal and cardiac muscle by a

PCR-based assay: lack of change of copy number with age. Nucleic Acids Res 31: e61–e61. https://doi.org/10.1093/nar/gng060

Moore J. K., and J. E. Haber, 1996 Cell cycle and genetic requirements of two pathways of nonhomologous end-joining repair of double-strand breaks in Saccharomyces cerevisiae. Mol Cell Biol 16: 2164–2173. https://doi.org/10.1128/mcb.16.5.2164

Morley A. A., 1995 The somatic mutation theory of ageing. Mutat Res Dnaging 338: 19–23. https://doi.org/10.1016/0921-8734(95)00007-s

Muller H. J., 1964 The relation of recombination to mutational advance. Mutat Res Fundam Mol Mech Mutagen 1: 2–9. https://doi.org/10.1016/0027-5107(64)90047-8

Notley-McRobb L., S. Seeto, and T. Ferenci, 2002 Enrichment and elimination of mutY mutators in Escherichia coli populations. Genetics 162: 1055–62.

Odagiri Y., H. Uchida, M. Hosokawa, K. Takemoto, A. A. Morley, *et al.*, 1998 Accelerated accumulation of somatic mutations in the senescence-accelerated mouse. Nat Genet 19: 116–117. https://doi.org/10.1038/468

Orgel L. E., 1963 The Maintenance of The Accuracy of Protein Synthesis and Its Relevance To Ageing. Proc National Acad Sci 49: 517–521. https://doi.org/10.1073/pnas.49.4.517

Orgel L. E., 1970 The maintenance of the accuracy of protein synthesis and its relevance to ageing: a correction. Proc National Acad Sci 67: 1476–1476. https://doi.org/10.1073/pnas.67.3.1476

Peltomaki P., L. Aaltonen, P. Sistonen, L. Pylkkanen, J. Mecklin, *et al.*, 1993 Genetic mapping of a locus predisposing to human colorectal cancer. Science 260: 810--812. https://doi.org/10.1126/science.8484120

Petljak M., L. B. Alexandrov, J. S. Brammeld, S. Price, D. C. Wedge, *et al.*, 2019 Characterizing Mutational Signatures in Human Cancer Cell Lines Reveals Episodic APOBEC Mutagenesis. Cell 176: 1282-1294.e20. https://doi.org/10.1016/j.cell.2019.02.012

Preston B. D., T. M. Albertson, and A. J. Herr, 2010 DNA replication fidelity and cancer. Semin Cancer Biol 20: 281–293. https://doi.org/10.1016/j.semcancer.2010.10.009

Ryland G. L., A. O. C. S. Group, M. A. Doyle, D. Goode, S. E. Boyle, *et al.*, 2015 Loss of heterozygosity: what is it good for? Bmc Med Genomics 8: 45. https://doi.org/10.1186/s12920-015-0123-z

Szilard L., 1959 On the Nature of the Aging Process. Proc National Acad Sci 45: 30–45. https://doi.org/10.1073/pnas.45.1.30

Telenius H., N. P. Carter, C. E. Bebb, M. Nordenskjo¨ld, B. A. J. Ponder, *et al.*, 1992 Degenerate oligonucleotide-primed PCR: General amplification of target DNA by a single degenerate primer. Genomics 13: 718–725. https://doi.org/10.1016/0888-7543(92)90147-k

Teo Y. V., M. Capri, C. Morsiani, G. Pizza, A. M. C. Faria, *et al.*, 2019 Cell-free DNA as a biomarker of aging. Aging Cell 18: e12890. https://doi.org/10.1111/acel.12890

Thomas D. C., J. D. Roberts, R. D. Sabatino, T. W. Myers, C. K. Tan, *et al.*, 1991 Fidelity of mammalian DNA replication and replicative DNA polymerases. Biochemistry-us 30: 11751–11759. https://doi.org/10.1021/bi00115a003

Tracy M. A., M. B. Lee, B. L. Hearn, I. T. Dowsett, L. C. Thurber, *et al.*, 2020 Spontaneous Polyploids and Antimutators Compete During the Evolution of Saccharomyces cerevisiae Mutator Cells. Genetics 215: 959–974. https://doi.org/10.1534/genetics.120.303333

Trifunovic A., A. Wredenberg, M. Falkenberg, J. N. Spelbrink, A. T. Rovio, *et al.*, 2004 Premature ageing in mice expressing defective mitochondrial DNA polymerase. Nature 429: 417–423. https://doi.org/10.1038/nature02517

Vermulst M., J. H. Bielas, G. C. Kujoth, W. C. Ladiges, P. S. Rabinovitch, *et al.*, 2007 Mitochondrial point mutations do not limit the natural lifespan of mice. Nat Genet 39: 540–543. https://doi.org/10.1038/ng1988

Vermulst M., A. S. Denney, M. J. Lang, C.-W. Hung, S. Moore, *et al.*, 2015 Transcription errors induce proteotoxic stress and shorten cellular lifespan. Nat Commun 6: 8065. https://doi.org/10.1038/ncomms9065

Wagner J. R., C. C. Hu, and B. N. Ames, 1992 Endogenous oxidative damage of deoxycytidine in DNA. Proc National Acad Sci 89: 3380–3384. https://doi.org/10.1073/pnas.89.8.3380

Wang D., D. A. Kreutzer, and J. M. Essigmann, 1998 Mutagenicity and repair of oxidative DNA damage: insights from studies using defined lesions. Mutat Res Fundam Mol Mech Mutagen 400: 99–115. https://doi.org/10.1016/s0027-5107(98)00066-9

Wang M., W. Wu, W. Wu, B. Rosidi, L. Zhang, *et al.*, 2006 PARP-1 and Ku compete for repair of DNA double strand breaks by distinct NHEJ pathways. Nucleic Acids Res 34: 6170–6182. https://doi.org/10.1093/nar/gkl840

Williams G. C., 1957 Pleiotropy, Natural Selection, and the Evolution of Senescence. Evolution 11: 398. https://doi.org/10.2307/2406060

Zong C., S. Lu, A. R. Chapman, and X. S. Xie, 2012 Genome-Wide Detection of Single-Nucleotide and Copy-Number Variations of a Single Human Cell. Science 338: 1622–1626. https://doi.org/10.1126/science.1229164

Zuccala E., 2016 Through the cracked lens of cancer genomes. Nat Rev Genet 17: 7–7. https://doi.org/10.1038/nrg.2015.13

Chapter Two
Rate volatility and asymmetric segregation diversify mutation burden in cells with mutator alleles.

Ian T. Dowsett[a], Jessica L. Sneeden [a], Branden J. Olson [b, c], Jill McKay-Fleisch[a], Emma McAuley[a], Scott R. Kennedy[a], Alan J. Herr[a,*]

Author Information
[a]Department of Laboratory Medicine and Pathology, University of Washington, Seattle, WA 98195-7705, USA
[b]Department of Statistics, University of Washington, Seattle, WA 98195-7705, USA
[c]Computational Biology Program, Fred Hutchinson Cancer Research Center, Seattle, WA 98109, USA

Abstract

Mutations that compromise mismatch repair (MMR) or DNA polymerase ε or δ exonuclease domains produce mutator phenotypes capable of fueling cancer evolution. Here, we investigate how combined defects in these pathways expands genetic heterogeneity in cells of the budding yeast, *Saccharomyces cerevisiae*, using a single-cell resolution approach that tallies all mutations arising from individual divisions. The distribution of replication errors present in mother cells after the initial S-phase was broader than expected for a single uniform mutation rate across all cell divisions, consistent with volatility of the mutator phenotype. The number of mismatches that then segregated to the mother and daughter cells co-varied, suggesting that each division is governed by a different underlying genome-wide mutation rate. The distribution of mutations that individual cells inherit after the second S-phase is further broadened by the sequential actions of semiconservative replication and mitotic segregation of chromosomes. Modeling suggests that this asymmetric segregation may diversify mutation burden in mutator-driven tumors.

Introduction

All tumors contain genetically divergent cells spawned by the evolutionary processes of mutation and selection. In some tumors, genetic heterogeneity arises from a “mutator phenotype”(Beckman and Loeb, 2017) due to mismatch repair (MMR) defects(Lynch et al., 2009) or heterozygous exonuclease domain mutations (EDM)

affecting the leading or lagging strand DNA polymerases (pol), Polε or Polδ (Barbari and Shcherbakova, 2017; Church et al., 2013; Hodel et al., 2018; Levine et al., 2013; Muzny et al., 2012; Palles et al., 2013; Yoshida et al., 2011). Since MMR corrects polymerase errors, when MMR and EDM mutations occur together they produce a dramatic increase in the number of unrepaired polymerase errors. The resulting tumors rapidly evolve and possess “ultramutated” genomes. Yet a full understanding of the relative contributions of mutagenesis and selection to the rise of heterogeneity within these tumors remains elusive, since cells with more mutations tend to adapt more readily.

A key unanswered question is whether the mutation rate is constant within populations of cells with mutator alleles (mutator cells). The two most common ways of measuring mutation rates are fluctuation analysis(Foster, 2006) and mutation accumulation lines(Lynch et al., 2008). Both assume a uniform mutation rate and report the average of hundreds or thousands of cell divisions. However, in recent years, evidence has emerged that mutagenic processes may vary from one division to the next. Kataegis and chromothripsis, for instance, sharply increase mutation burden in a single cell division(Alexandrov et al., 2013; Nik-Zainal et al., 2012; Zhang et al., 2015). Indirect evidence for highly mutagenic sub-populations of cells also comes from studies of yeast exposed to 6-hydroxylaminopurine or AID/APOBEC cytosine deaminase. Selected mutants in mutation rate assays had substantially higher mutation burdens than non-selected isolates from the same population(Lada et al., 2013). Episodic APOBEC mutagenesis also occurs in human cell cultures propagated for prolonged periods(Petljak et al., 2019). Moreover, limited single-cell propagation of human cancer cell lines coupled to whole genome sequencing revealed broader than expected variation in mutation rate in closely related subclones(Brody et al., 2018). Observations such as these challenge the assumption that mutation rate is constant and beg higher resolution studies of mutator cells.

The asymmetrically dividing budding yeast, *S. cerevisiae*, is ideal for studying mutator phenotypes with high resolution. It encodes many of the same DNA replication and mismatch repair genes found in humans. Yeast “daughter” cells can be separated from their larger “mother” cell at each division by micromanipulation and then moved to defined locations on an agar plate, forming a “single cell lineage”. Whole genome sequencing (WGS) of cultures derived from these cells permits the number of new mutations that arose in the mother cell at each division to be counted. Moreover, the small size of the genome (12 megabases) makes it cost effective to score enough cell divisions to see whether the distribution of mutation counts conforms to that expected from a single underlying mutation rate.

We previously pioneered this approach with haploid mutator mother cells deficient in Polε proofreading and MMR (*pol2-4 msh6Δ*)(Kennedy et al., 2015). A single underlying mutation rate could not explain the distribution of mutation counts from 87 divisions. However, the distribution did fit a model with two underlying mutation rates that differed by 10-fold (0.4 and 4 mutations/genome/division). This led to a hypothesis

of “mutator volatility” in which cells assumed one of two mutator states as they passed through the cell cycle(Kennedy et al., 2015). But since we only scored mutations retained by the mother, we could not exclude an alternative hypothesis: that polymerase errors sporadically segregated asymmetrically between mother and daughter cells, either as mismatches at the initial division or as permanent double-stranded mutations following the next round of synthesis. Here, to distinguish between these two hypotheses, we sought to score all replication errors that arose in individual cell divisions using more extensive single cell lineages. Examination of the distribution of the full replication error counts from individual divisions provided a way to test the mutator volatility hypothesis apart from the confounding influence of segregation. At the same time, sequencing complete lineages gave us the means to determine whether replication errors segregate asymmetrically. The full replication error counts from two different mutator genotypes produced unimodal distributions that were significantly overdispersed relative to that expected from a single underlying mutation rate. Our data suggest that mutator volatility in these cells derives from continuous variation in the underlying mutation rate. Moreover, we found that asymmetric segregation due to the normal process of semiconservative DNA replication and mitotic segregation of chromosomes further expands the distribution of new mutations in individual mutator cells.

Results

Evidence for Mutator Volatility

To confidently score replication errors arising on all nascent DNA strands from each division, we devised a scheme that ensured that all mutations were observed in at least two members of a single cell lineage. After moving each daughter by micromanipulation from the founding mother cell, we isolated a sublineage of three additional cells to help score the number of errors segregated to that daughter. These cells included the first and second granddaughter (born to the daughter cell) as well as the first great-granddaughter cell derived from the first granddaughter (Fig. 2.1a). Errors segregated to the daughter as mismatches in the first division segregate as double-stranded mutations in the next division when the daughter produces the first granddaughter. Mutations retained by the daughter after that segregation event will be inherited by the second granddaughter, forming what we call the “Da” segregant group. Mutations segregated to the first granddaughter will be inherited by the great-granddaughter, forming the “Db” segregant group. In theory, the Da and Db segregant groups represent half of the errors made by the mother cell during a given division. The remaining errors, retained initially by the mother as mismatches, segregate between the mother and her next daughter as double-stranded mutations in the next division. The mutations segregated to that daughter will be uniquely present in the next sublineage, forming the “Ma” segregant group. Mutations retained by the mother will be found in all later sublineages, defining the “Mb” segregant group. After colony formation and WGS, a full error count for a given division can be determined by simply summing the number of mutations in the Da, Db, Ma, and Mb segregant groups. With a complete set of sublineages from the same mother cell, the full replication error counts from several sequential cell divisions can be determined from the nested data (Fig. 2.1b). By

requiring that all errors be observed in at least two members of the lineage, this approach eliminates false positives due to sequencing errors or clonal sweeps within the cultures.

We initially began our experiments with the *pol2-4 msh6Δ* haploid strain used in the previous study(Kennedy et al., 2015). We found evidence for a more limited mutator volatility but were concerned that lethality within some sublineages may have introduced a bias (see Supplementary Information and Supplementary Fig. 2.1). To improve viability and the mutational signal, we switched to using diploid yeast with a 10-fold higher mutation rate due to homozygous mutations affecting Polδ proofreading and base-base mismatch repair (*pol3-01/pol3-01 msh6Δ/msh6Δ*)(Kennedy et al., 2015; Luria and Delbrück, 1943). To obtain *pol3-01/pol3-01 msh6Δ/msh6Δ* cells, we mated *pol3-01 msh6Δ* haploids, freshly dissected from sporulated *POL3/pol3-01 MSH6/msh6Δ* diploids. We isolated the newly formed zygotes and then used the first or second diploid daughters as founding mother cells for the isolation of single-cell lineages, noting the time and placement of each cell. Following colony formation, and WGS, we scored 13,801 mutations from 50 divisions obtained from 7 different lineages (Fig. 2.1c, Supplementary Table 1, Supplementary Fig. 2.2). The mutations were distributed across the genome and displayed a spectrum consistent with combined proofreading and MMR deficiency (Supplementary Fig. 2.3). We only scored mutations at genomic sites confidently called in all members of a lineage and carefully vetted the resulting variant lists. Having complete lineage information allowed us to assign when the mutations arose using the logic described above. In addition, we visually inspected the variant sites in all genomes from a given lineage using the Integrative Genomics Viewer, which allowed us to detect discrepancies in the lineage order or whether mutations had been incorrectly assigned (see Methods). We tallied the full replication error counts from each division and determined whether the distribution could be explained by a single underlying mutation rate.

Mutagenesis has been modeled for more than 70 years(Jackson and Loeb, 1998; Luria and Delbrück, 1943; Youn and Simon, 2013) with the Poisson distribution, which is a discrete probability distribution of the number of expected independent events occurring within a defined interval, assuming a constant rate (λ). A simple test of whether a distribution matches a single Poisson is to calculate the index of dispersion ($\hat{D}$), which is equal to the variance of the distribution divided by the mean (σ^2/μ). The variance of Poisson distributions always equals the mean, which results in a $\hat{D}$ of 1. The *pol3-01/pol3-01 msh6Δ/msh6Δ* mother cells committed an average of 276 (±37.7, standard deviation (σ)) replication errors per division. This corresponds to a $\hat{D}$ of 5.15 ($37.7^2/276$), which suggests that the distribution does not conform to a single Poisson (Fig. 2.1c). Two alternative explanations failed to account for the overdispersion. For instance, we did not observe any relationship between the mother's replicative age and the number of errors made by Polδ (Spearman's rank correlation coefficient: 0.007209, p = 0.9604)(Supplementary Fig. 2.4), nor did the number of mutations correlate with the size of the scored genome, which differed between lineages due to variation in sequencing depth and the number of members in each lineage (Spearman's rank correlation coefficient: -0.0416, p = 0.7743)(Supplementary Fig. 2.4). Instead, the broad

distribution of full replication error counts, free from the confounder of segregation, is consistent with mutator volatility.

To better understand the nature of mutator volatility in *pol3-01/pol3-01 msh6Δ/msh6Δ* cells, we used finite mixture modeling, which employs a maximum likelihood framework to identify mixtures of two or more Poisson distributions that better fit the data. We also modeled the data as a negative binomial (nb), which is a discrete distribution with separate rate (μ) and shape parameters (θ) commonly used to interpret overdispersed count data. The rate parameters λ and μ, for the Poisson and nb distributions, both define the mean number of events. Since these models derive from different distributions, they cannot be directly compared using standard statistical tests. Non-nested models such as these can be evaluated with Akaike Information Criteria (AIC), which uses maximum likelihood to estimate the loss of information of each model relative to the observed distribution. To prevent overfitting, AIC penalizes models with more parameters. Lower AIC values correspond to a more parsimonious fit; however, interpreting the difference in the magnitude of raw AIC values is not intuitive. Thus, we transformed the raw AIC values to "Akaike weighted values", which conveys their relative likelihood (Fig. 2.1b)(Burnham and Anderson, 2004; Wagenmakers and Farrell, 2004). We found that the negative binomial model was the most likely (relative likelihood of 0.9999), followed by the two-Poisson-mixture model (2.2×10^{-6}), and the single Poisson (4.3×10^{-28}) (Fig. 2.1c). Similar results were obtained using Bayesian Information Criteria (BIC), which imposes stronger penalties for overfitting. Thus, mutator volatility in *pol3-01/pol3-01 msh6Δ/msh6Δ* cells is more complex than just two distinct mutator states.

Mutation Rate Varies Between Divisions

The superiority of the negative binomial model suggests that the mutator phenotype may vary continuously. This rationale derives from the ability to describe a negative binomial as a gamma-Poisson distribution (Fig. 2.2a). The gamma function is a continuous, rather than discrete, distribution. Here, it takes the same shape parameter (θ) as the negative binomial and serves as a conjugate-prior to define variation in the rate parameter λ of a mixture of Poisson distributions. The variation in λ that creates a negative binomial occurs between replication events at the same site, or a collection of sites such as a chromosome or genome. Having complete lineage information provided an opportunity to test whether λ varies at a chromosomal or genome-wide level. The distributions of mismatches segregated to mother (Mm) or daughter cells (Dm) across all divisions were the same and fit a negative binomial (Fig. 2.2b). If λ varied widely during the replication of individual replicons (the units of DNA replication on a chromosome), this could introduce asymmetry in the number of errors on sister chromatids, which would then propagate to the daughter and mother cells (Fig. 2.2c). Consequently, Dm and Mm from the same division would be free to vary within the observed negative binomial distribution. Alternatively, if the genome-wide value for λ varies between cell divisions, a single mutation rate would govern mismatch formation for both the mother and daughter genomes (Fig. 2.2d). Dm and Mm would co-vary within the constraints of the corresponding Poisson distribution. To distinguish between these two hypotheses, we first compared the correlation of mismatches segregated to

mother and daughter cells to simulated data generated under the constraints of the two models. While no correlation was seen between Dm and Mm in the simulated data from the “replicon variant” model (R^2=0.001), similar correlations were observed for both the simulated data from the “division variant” model (R^2=0.47) and the actual data (R^2=0.37). This correspondence in the number of mismatches segregated to mother and daughter cells extended down to the level of chromosomes (Fig. 2.2e). The R^2 values are lower than typically seen with strong correlations, but as our modeling shows, this is expected since both X and Y values are randomly drawn from a Poisson distribution. As a second test of the hypotheses, we also performed 10,000 simulations of how each model would affect the distribution of full replication error counts from 50 divisions (Fig. 2.2f). With the replicon variant model, the simulated index of dispersion (3.28 ± 0.66, σ) was substantially less than observed with the actual data ($\hat{D}$=5.15), while the division variant model produced a good match (5.54 ± 1.12, σ). Together, these analyses strongly suggest that the source of mutator volatility is variation in the genome-wide mutation rate from one division to the next.

Asymmetric Inheritance of New Mutations

With this support for the mutator volatility hypothesis, we turned our attention to the question of asymmetric inheritance of new mutations. Individual cells averaged 69 (±18, σ) new mutations/diploid genome/division (*n* = 200) (Fig. 2.3a) with an index of dispersion of 4.8. A negative binomial fit the distribution most closely (relative likelihood = 0.82), followed by a four-Poisson mixture model (relative likelihood = 0.18). A close examination of mutations arising from the same division revealed a striking asymmetric pattern of inheritance. When pairs of segregant groups were compared (e.g. Da vs Db or Ma vs Mb), half of the time one segregant group inherited all of the mutations for a given chromosome while the other received none (Fig. 2.3b,c). This pattern is explained by the sequential actions of semiconservative DNA replication and mitotic segregation of chromosomes (Fig. 2.3d). At the end of the first S-phase, due to semiconservative replication, all errors arising due to the Poisson process of polymerase error formation reside on one of the two strands of each sister chromatid. These strands segregate equally between mother and daughter cells. The next round of replication produces two new duplexes per cell, only one of which contains double-stranded mutations. At metaphase, cells receive either all or none of the new mutations for that chromosome from the previous division. This binomial process occurs twice for every chromosome number in diploid cells. Consequently, for each chromosome number, cells receive 0%, ~50%, or 100% of the mutations in a given division with a “Mendelian” ratio of 1:2:1 (Fig. 2.3c) (actual ratio, 876:1490:834). Thus, we can describe how polymerase errors arise in an individual division and later become permanent as a compound Poisson-binomial process.

To determine the contribution of the Poisson-binomial process to the overdispersion of mutation counts, we simulated mutagenesis in *pol3-01/pol3-01 msh6Δ/msh6Δ* cells assuming a constant error rate. Given that we observed an average of 138 mismatches per diploid mother or daughter cell (Fig. 2.2b), the average rate of

error formation was 69 errors/haploid genome/division. Since cells only inherit, on average, half of the polymerase errors, the observed mutation rate in *pol3-01/pol3-01 msh6Δ/msh6Δ* cells was 34.5 mutations/haploid genome/division. To model the Poisson-binomial process we simulated mutagenesis on each chromosome by setting λ equal to 69 errors/haploid genome and then, to mimic segregation, multiplied the number of mutations apportioned to each chromosome by a randomly chosen 1 or 0, before summing the total mutations (Fig. 2.3e). For comparison, we simulated mutation accumulation assuming a simple Poisson process in which mutations accumulated with a rate of 34.5 mutations per haploid genome (Fig. 2.3e). With 1000 simulations of 200 cell cohorts, the Poisson-binomial model produced a broader index of dispersion ($\hat{D}$ = 3.58±0.49, σ) than the Poisson model ($\hat{D}$ = 1.0±0.1, σ) (Fig. 2.3f), but narrower than the observed data ($\hat{D}$ = 4.8). However, substituting the constant mutation rate with the gamma-distributed set of λ values from Fig. 2.2c yielded simulated data with an equivalent dispersion ($\hat{D}$ = 4.80±0.49, σ) (Fig. 2.3f). Thus, the combination of mutator volatility and asymmetric segregation of mutations — a gamma-Poisson-binomial process —accounts for the observed distribution of mutations in individual *pol3-01/pol3-01 msh6Δ/msh6Δ* cells.

To understand the potential implications of our findings for mutator-driven cancers, we first focused on how the Poisson-binomial process would influence the heterogeneity of mutation burden within a dividing population of tumor cells. Assuming a constant mutation rate comparable to *pol3-01/pol3-01 msh6Δ/msh6Δ* yeast, the expected distribution of simulated mutation counts in human cells after one division ($\hat{D}$ = 50) was far broader than in yeast (Fig. 3g) and persisted through 30 simulated divisions (Fig. 2.3h,i). Adding a comparable level of volatility to the mutator phenotype further increased the simulated dispersion ($\hat{D}$ = 82) (Fig. 2.3g). Using the Poisson-binomial model, we simulated a range of mutator phenotypes observed in cancer cells and found a linear relationship between mutation rate and predicted index of dispersion. For instance, mutation accumulation in HCT116, the well-known MLH1 mutant colon cancer cell line, increases from 48 to 190 mutations/haploid genome/division upon introduction of a heterozygous *POLE* proofreading-deficient allele(Hodel et al., 2018). In these cells, the predicted index of dispersion expanded from 3.4 to 10.8 (Fig. 2.3j). Even greater heterogeneity may arise in human cancers when more potent *POLE* mutator alleles occur in combination with MMR deficiency(Church et al., 2013; Kane and Shcherbakova, 2014; Levine et al., 2013; Li et al., 2018). Thus, the fundamental Poisson-binomial process of asymmetric segregation of new mutations has the potential to dramatically expand the diversity of mutation burdens present among a population of human mutator cells.

Discussion

Genetic heterogeneity progressively increases in a dividing population of cells as an unavoidable consequence of errors made during DNA synthesis. Here, for the first

time, we describe the fate of polymerase errors made on all nascent DNA strands synthesized in individual cell divisions. We developed this single cell resolution approach in order to understand previous observations that the distribution of new mutations in individual mutator cells was broader than expected. To explain the phenomenon, we proposed two hypotheses: (1) that mutator phenotypes are volatile and (2) that polymerase errors arise with a constant rate but segregate asymmetrically on the way to becoming double-stranded mutations. The design of our single cell pedigrees ensured at least two independent biological observations for each mutation, which allowed us to confidently assign more than 13,000 mutations to fifty divisions. From the resulting mutation count data, we found strong evidence that both mutator volatility and asymmetric segregation of new mutations significantly expand genetic heterogeneity in *pol3-01/pol3-01 msh6Δ/msh6Δ* yeast.

Historically, mutagenesis has been modeled with the Poisson distribution, which describes the probability of the number of independent events per unit time given a constant rate. The observed distribution of full replication error counts of mutator cells, free from the influence of segregation, best fit a negative binomial and not a single Poisson (Fig. 2.1c). Negative binomials are equivalent to a continuous mixture of Poisson distributions whose rates vary according to a gamma distribution (Fig. 2.2a). This suggests that mutator volatility may create a continuum of mutation rates rather than discrete mutator states. We explored the idea that mutation rate varies from one division to the next by simulating the number of mismatches segregated to mother and daughter cells (Fig. 2.2d,e) and the dispersion of full replication error counts expected from small cohorts of cells (Fig. 2.2f). Both simulations closely matched the observed data, consistent with the hypothesis that mutator volatility derives from continuous variation in mutation rate between divisions. Two caveats are worth noting. First, this model of volatility was developed from a relatively small sample size (n = 50). Substantially increasing the number of scored divisions may reveal that the underlying distribution is derived from discrete mutator states rather than a continuum of rates. Second, in cells with different mutator alleles the underlying distribution may vary substantially depending on how the mutator alleles interact with the currently undefined source of volatility, and mutator phenotypes affecting other processes besides DNA replication may have different sources of volatility. Mutator polymerases do not operate as a closed system. They interface with a myriad of other replication components and metabolites, such as dNTPs, that influence their fidelity(Mertz et al., 2015; Williams et al., 2015). Variation in the timing and duration of perturbations to these interactions may produce volatility. The observed overall mutation rate that cells exhibit represents a composite of mutation rates at all sites within the genome. Conceivably, the change in replication fidelity could be localized to certain parts of the genome in a given division. But if so, our data suggests, that the nascent strands from each pair of sister chromatids in the affected region must be equally influenced by the change in rate (Fig. 2.2c,f).

The asymmetric inheritance of new mutations observed in mutator cells results from the fundamental processes of semi-conservative replication and mitotic

segregation of chromosomes acting in concert. Current models of mutation accumulation generally ignore the potential for this synergy to expand genetic heterogeneity, although there are exceptions. John Cairns proposed a far more extreme asymmetric inheritance of mutations in the “Immortal Strand Hypothesis” in which stem cells always segregated away newer DNA duplexes with double-stranded mutations(Cairns, 1975). In keeping with this hypothesis, a recent computational analysis of human somatic variants argued that the high variance of mutation burden in adult stem cells with age supports a preferential inheritance of ancestral strands(Werner and Sottoriva, 2018). A second study from the field of evolutionary biology examined the potential influence of disparate mutagenesis of leading and lagging strand synthesis to promote variable evolutionary trajectories from the same cell population(Furusawa, 2014). Our findings here demonstrate that, in the context of a mutator phenotype, the *normal* process of semi-conservative replication and mitotic segregation of chromosomes has the potential to create unequal sharing of mutations. We find no evidence that daughter or mother cells preferentially inherit new mutations (Supplementary Fig. 2.5). For every cell that inherits disproportionately more mutations there will be another cell with fewer mutations. The predicted impact of this process on the variation in mutation burden is larger in human cells than in yeast due to the vast differences in chromosome length, and the correspondingly larger number of errors per chromosome. However, with longer chromosomes comes an increased likelihood that sister chromatid exchanges (SCEs) may mitigate the asymmetry. SCEs clearly to do not homogenize mutation burden in diploid mutator yeast cells since half of cells either received all or none of the new mutations for a given chromosome (Fig. 2.3c). A high frequency of SCEs would have left few chromosomes with 0 mutations. This finding is in keeping with recent evidence from a sensitive Next Generation Sequencing methodology (Strand-seq) that SCE occurs with a rate of 0.26 events/division in yeast(Claussin et al., 2017). Strand-seq experiments of normal human fibroblasts and lymphoblasts indicate the SCEs occur with a rate of 5 events/cell division(van Wietmarschen and Lansdorp, 2016). At this rate, most chromatid pairs in mutator cells would be free of SCEs even after the two divisions it takes for polymerase errors to become double-stranded. Of course, the frequency of SCEs may increase in some cancer cells, especially those with certain intrinsic DNA repair defects(van Wietmarschen and Lansdorp, 2016). Performing single cell lineage analysis of human mutator cells in future studies should address both the prevalence of SCEs and the asymmetric inheritance of mutations.

Our simulation of a mutator-driven tumor rapidly generated substantial intra-tumoral genetic heterogeneity during expansion (colored lines, Fig. 2.3i) compared to a population in which mutations accumulated by a simple Poisson process (black line, Fig. 2.3j). The associated variability in mutation load may be relevant to cancer evolution. Early during tumorigenesis the subpopulation of cells that inherit disproportionately more mutations may adapt more readily. With elevated mutation rates, polyclonal adaptation is almost certain. The unifying feature of these adapted cells is a high mutation burden. As mutation burden mounts and mutator cells contend

with increasingly strong negative selection pressure due to immune surveillance and negative epistatic interactions(Herr et al., 2014; Shlien et al., 2015), adapted cells that inherit fewer new mutations due to asymmetric inheritance may be at a relative fitness advantage. In this context, selectively increasing mutation rate in mutator cancer cells could represent a novel therapy(Williams et al., 2015). If, as a means of treatment, the mutation rate of cancer cells is only transiently elevated to induce extinction, this subpopulation may persist. Sustained elevation of mutation rate over many divisions of mutator cells may be required to drive their extinction.

Methods

Yeast strains and culture conditions.

The diploid strains AH2801 (*POL2/URA3::pol2-4 MSH6/msh6Δ::LEU2*)(Kennedy et al., 2015) and AH2601 (*POL3/URA3::pol3-01 MSH6/msh6Δ::LEU2*)(Lee et al., 2019) are derived from AH0401, a BY4743 derivative engineered to be heterozygous at the *CAN1* locus (*CAN1::natMX/can1Δ::HIS3*) to facilitate forward mutation rate assays(Herr et al., 2014). We followed standard procedures for yeast propagation and tetrad dissection(Sherman, 2002). For general propagation, we grew liquid YPD cultures (1% wt/vol yeast extract, 2% wt/vol peptone, 2% wt/vol dextrose) at 30°C. For sporulation, we diluted overnight YPD cultures 1:100 in 3 mls of YPD and grew until the culture reached 1-2 x 10^7 cells/ml. We recovered the cells by centrifugation, resuspended and pelleted the cells once in 1 ml H_2O, and then resumed growth at 22-25°C in 2 mls of sporulation media (1% potassium acetate, 0.1% yeast extract, 0.05% dextrose) for five days. For rich solid media, we used synthetic complete (SC) [6.7 g Difco yeast nitrogen base without amino acids, 2% wt/vol dextrose, 2 g/L SC amino acid Mix (SCM) (Bufferad)] supplemented with 2% wt/vol agar. For plates lacking leucine and uracil (SC-Leu-Ura), SCM was substituted for SCM-Leu-Ura (Bufferad). Archival frozen stocks were stored in 23% glycerol at -80°C.

Single cell lineage isolation

To isolate *pol2-4 msh6Δ* lineages we dissected AH2801 tetrads on SC-Leu-Ura selective media and chose one germinating spore per plate to serve as the founding mother cell. To obtain *pol3-01::URA3/pol3-01::URA3 msh6∆::LEU2/msh6∆::LEU2* cells for pedigree analysis we first dissected *POL3/ pol3-01::URA3 MSH6/ msh6∆::LEU2* tetrads on SC-Leu-Ura plates. After two divisions, double mutant haploid cells from different tetrads were placed next to each other to allow mating. Upon isolation of a zygote, the first or second daughter was used as the founding mother (M) for the lineage. Mothers were placed at an isolated location and we separated daughter cells (designated Dn, Dn+1, etc.) from the mother as they were generated and moved them to select areas 5 mm apart on the plate. We repeated the procedure to obtain each daughter's first daughter (GD.1, Fig. 2.1b), second daughter (GD.2), and first granddaughter (GGD, born to GD.1). This strategy was repeated for each daughter up to either the 20th division or the end of the mother's replicative lifespan, whichever occurred first. In a typical experiment, we pre-punched the agar with the dissecting needle at each drop-off location so that we would always put the cell in a defined place, making it easy to later find the cell for inspection and manipulations. We isolated lineages over the span of a week by performing rounds of dissections every 90-120

minutes. Only a few cells on a plate were moved in any one round, and then, only one cell at a time. We noted the timing of each round of bud dissections. We incubated plates at 30°C between dissections. At the end of the day, plates were wrapped in parafilm and stored overnight at 4°C. When plate dissections were concluded, we incubated each plate an additional 48 hours at 30°C to allow colonies to fully develop. Prior to sequencing, the *pol3-01/pol3-01 msh6Δ/msh6Δ* and *pol2-4 msh6Δ* genotypes were confirmed by allele-specific PCR assays(Lee et al., 2019).

Genome sequencing

Each colony in a pedigree was used to inoculate overnight 5 ml liquid YPD cultures for whole genome sequencing(Lee et al., 2019). Glycerol stocks were made and genomic DNA extracted with the ZR Fungal/bacterial purification kit (Zymo Research). DNA was sheered into 500 to 1000 bp fragments by sonication. After end-repair, Illumina sequencing libraries were made by ligating on dsDNA adapters and indexing by quantitative PCR. The samples were then sequenced on the HiSeq 2500 or Nextseq platforms. We performed sequencing alignments and variant calling using a custom pipeline (eex_yeast_pileline.sh) that runs in the Unix command-line. Reads were aligned to a repeat-masked S288C yeast genome(Kennedy et al., 2015) using the Burrows-Wheeler Aligner (0.7.17)(Li and Durbin, 2009). We removed discordant and split-read groups using Samblaster (0.1.24)(Faust and Hall, 2014). We used Picard tools (2.21.9) AddorReplaceReadGroups to add information to the header used for later steps in the analysis. We then indexed the BAM files with Samtools (1.8)(Li and Durbin, 2009). To minimize false variant calls, we sequentially processed the BAM files with functions from the Genome Analysis Toolkit (GATK3)(DePristo et al., 2011) including RealignerTargetCreator, IndelRealigner, LeftAlignIndels, BaseRecalibrator, and PrintReads. We made a pileup file with Samtools and used VarScan (v2.3.9) mpileup2snp to call single nucleotide variants(Koboldt et al., 2012). We limited our analysis to single nucleotide variants, which are by far the most abundant polymerase error type in these cells. We used the Varscan2 tool to identify variants present in our colonies with the following parameters. For *pol2-4 msh6Δ* haploid lineages we used a variant frequency cut-off of 0.8 with a minimum read-depth of 18 (daughter and GD.1 positions) or 10 (for GD.2 and GGD positions). Since these are haploid cells, new variants should be present in 100% of reads. Setting the cut-off at 0.8 accommodates sites with low read depth and one sequencing error. For *pol3-01/pol3-01 msh6Δ/msh6Δ* diploids, we used a minimum read-depth of 18 for all strains and a variant frequency cut-off of 0.22. With a read depth of 18, clonal heterozygous variants in diploid cells have a false negative rate of 6.1×10^{-5}. With 1000 mutations we have a 6% chance of having 1 false negative in a genome. We filtered the above results to remove variants present in the parental strains as well as recurrent sequencing artifacts. A small number of variants (<0.1%) could be reliably scored with the above parameters but fell below a quality threshold for a subset of genomes. These were manually curated for inclusion. We detected these by visually inspecting the BAM files for all strains in a single cell lineage at the same time using the Integrative Genome Viewer (IGV)(Robinson et al., 2011).

Scoring of mutations and detection of assignment errors

We used a custom Python script (JLSLineageCaller) to determine the number of shared variants within each lineage. The program first determines all genomic positions with 18-fold read-depth in all members of the lineage and then filters the called variant lists for mutations at positions within the shared genome. Pairwise comparisons are done between certain strains to identify shared mutations at different branch points in the lineage, resulting in a data-frame of comparisons that allows all mutations arising in a lineage to be sorted and examined in Microsoft Excel. The mutation counts for division *n* were determined by summing the number of new mutations identified at branchpoints Da (GDn.1 vs GGDn.1), Db (Dn vs GDn.2), Ma (Dn+1 vs GDn+1.1), and Mb (Dn+2 vs Dn+3). Da mutations are only found in the daughter (Dn) and her second daughter (GDn.2). Likewise, Db mutations are only found in GDn.1 and her first daughter GGDn.1. Mismatches retained by the mother after the first division become double-stranded mutations in the next division and are either passed on to her next daughter (Dn+1) or are retained by the mother and passed on to all future offspring. The mutations inherited by Dn+1 that form the Ma segregant group are only found in this branch of the lineage. Finally, the mutations retained by the mother, the Mb segregant group, first appear in Dn+2 and her offspring, but also show up in all subsequent daughters (Dn+3, Dn+4, etc) and their offspring. Any deviation from this pattern of inheritance indicates an “assignment error” has occurred and that a cell was inadvertently placed in the wrong position in the lineage. In the Supplementary Information we describe two such cases. The divisions encompassing these strains were censored from the analysis. Below we describe how these errors arise and are detected to illustrate the reliability of the method.

One possible assignment error could occur at dissection when the daughter and mother cells both divide before the next round of dissection. On the basis of size, the first daughter (Dn) can be easily distinguished from the mother, the second daughter (Dn+1), and her own daughter (GDn.1). Usually Dn+1 and GDn.1 can also be distinguished because Dn+1 buds before GDn.1. However, in rare cases Dn+1 and GDn.1 are adjacent and similarly sized. If Dn+1 is moved in place of GDn.1, we will have a sublineage consisting of Dn, Dn+1, GDn.2, and GDn+1.1 (instead of Dn, GDn.1, GDn.2, and GGDn.1). Every sublineage should normally contain subsets of mutations from different divisions (Da and Db mutations from the “n” division; Ma mutations from the “n-1” division; and Mb mutations from the “n-2” division). In this sublineage, the Ma segregant group mutation count will be 0, since there are no *new* mutations that will be shared by these four colonies. However, a substantial subset of the mutations assigned to the Db segregant group will also be found in later sublineages indicating that they are *not* Db mutations but Mb mutations from a later division. The other half of what appear to be Db mutations will in fact be Ma mutations from a different division. Added confirmation of the dissection error comes from the analysis of the next sublineage, which will consist of GDn.1 (not Dn+1 as it should be), GGDn.1, GGDn.2, GGGDn.1 (great-great-great granddaughter 1). There will be 0 Mb mutations in this sublineage since all of these cells are directly descended from Dn. These problematic cell divisions would be censored because we lack key lineage members necessary to obtain a full replication error count. Another type of assignment errors could occur during dissections to isolate the sublineages. For instance, if Dn divides twice in the interval before the

next round of dissection we would have to distinguish between GDn.1 and GDn.2. This is usually easy to do because, as above, GDn.1 would be forming a bud while GDn.2 would be unbudded. If we inadvertently reversed those two cells, we would have a sublineage consisting of Dn, GDn.2, GDn.1, and a great granddaughter born to the second granddaughter. When calling the Da segregant group we would be calling shared mutations between Dn and GDn.1 (and not between Gn and GDn.2). We would quickly see that these are, in fact, Ma segregant group mutations because they would also be present as a subset of Db mutations obtained in the comparison between GDn.2 and her offspring.

The most difficult potential assignment errors to detect would occur in the Da and Db segregant groups. For example, if GDn.1 divided twice, producing GGDn.1 and GGDn.2, and we selected GGDn.2 instead of GGDn.1, the mutation count for the Db segregant group would be derived from two divisions instead of one. Again, this is unlikely, because GGDn.1 would begin budding long before GGDn.2. But we lack an obvious distortion to the pattern of mutation inheritance to flag this as an error. We don't think this is a common problem given the correspondence between mismatches segregated to the mother (Mm) and daughter (Dm) cells illustrated in Fig. 2.2d,e. As described above, we regard the Ma and Mb segregant groups as highly reliable because dissection errors lead to obvious perturbations in the pattern of mutation inheritance. In favor of the reliability of the Da/Db data, an XY scatter plot of mutation counts observed in pairs of Ma/Mb segregant groups corresponds very well to that observed with pairs of Da/Db segregant groups (Supplementary Fig. 2.5). Both sets also correspond with what would be expected based on simulated data. (The simulation assumed a gamma-Poisson distribution as in Fig. 2.2). Interestingly, there are two Ma/Mb (47,36) and Da/Db (52, 20) segregant pairs in the lower left-hand quadrant that appear as outliers. Both pairs are derived from Division 15 (Supplementary Data 1), leading to the conclusion that the mutation rate in that division was inherently low. The highest Da/Db outlier (51,120), derived from Division 8, is also associated with a Ma/Mb pair with high mutation counts (120,65), leading to the conclusion that this division had a high mutation rate.

Statistical modeling

We grouped the mutation counts from the above branch points into Da, Db, Ma, and Mb segregant groups to determine their distributions. We also joined all segregant groups into one larger group to examine the distribution of mutation counts across all cell divisions. To determine the distributions of mismatches segregated to the daughter (Dm) and mother (Mm) cells, we first summed the Da and Db or Ma and Mb mutation counts from each division. We also combined these two sets into one group to view the distribution of mismatches across all cell divisions. To determine the distribution of total polymerase errors per division, we summed all mutations from individual divisions (Da+Db+Ma+Mb). We considered two common approaches for modeling over dispersed count data: the Poisson mixture distribution and the negative binomial distribution.

A *K*-component Poisson mixture distribution, which we denote PM(*K*), has a probability mass function (pmf) given by

(1)

$$f_{\mathrm{PM}}(\mathrm{x}; K, \mathbf{p}_K, \boldsymbol{\lambda}_K) := \sum_{k=1}^{K} p_k f_{Poisson}(x; \lambda_k)$$

where $\mathbf{p}_K$ = (p_1,…,p_K) is a vector of mixture proportions, $\boldsymbol{\lambda}_K$= (λ_1,…, λ_K) is a vector of Poisson means, and f is the pmf of a Poisson(λ_k):

(2)

$$f_{Poisson}(x; \lambda_k) := \frac{\lambda_k^x e^{-\lambda_k}}{x!}.$$

From this formulation, we see that the full density of the distribution is decomposed as a sum of the scaled Poisson densities. In (1), p_k represents the prior probability that a given count measurement will be generated from the *k*th Poisson component distribution, parameterized by λ_k. Since a given count measurement could have been generated from any of these *K* components, we average over their densities based on their prior probabilities to get the full density of that count.

The negative binomial distribution can be specified by the following probability mass function:

(3)

$$f_{NegBinom}(x; \mu, \theta) := \frac{\Gamma(x+\theta)}{x!\Gamma(\theta)} \left(\frac{\theta}{\theta+\mu}\right)^{\theta} \left(\frac{\theta}{\theta+\mu}\right)^{x}$$

where μ is the rate parameter and θ is the shape or dispersion parameter. As θ tends towards zero, the variance increases. As $\theta \rightarrow \infty$, the negative binomial reduces to a Poisson distribution.

We implemented these principles using a single R script (FMM.R, see Appendix or Github link below).To fit Poisson mixture models we used the flexmix R package in R v3.5.3(Leisch, 2004). To fit negative binomial models we used the glm.nb function of the MASS R package(Venables et al., 2002). Goodness of fit testing of the models was performed using both Akaike information criterion (AIC) and Bayesian information Criterion (BIC) in R. Although these two approaches score fit in slightly different ways, BIC returned results consistent with AIC and we thus report only the more commonly used AIC scores. We scored each tested distribution against up to 4 parameters. We reported only up to the number of parameters that improved model fit. Lower raw AIC values indicate better fit; however, the relative differences are not immediately intuitive and so we calculated Akaike weighted values(Jackson and Loeb, 1998; Wagenmakers and Farrell, 2004). To illustrate this approach, the AIC values in Fig. 2.1b were 637, 537, and 511. The first step in getting weighted AIC values is to determine Δ_iAIC: the difference between each AIC value and the AIC with the lowest value (so for these numbers: 126, 26, 0). The likelihood of each is then calculated by exp(-1/2 x Δ_iAIC). The weighted AIC value for a given model is its likelihood divided by the sum of all competing likelihoods. From these calculations the weighted AIC values are 4.3e-28 (P, k=1), 2.2e-6 (P, k=2), and 0.9999978 (nb), respectively. Thus, the negative binomial model is far more likely than the other two models to account for the observed data. Mixture model graphs were constructed using the ggplot2 package R(Villanueva and Chen, 2019). Spearman rank correlation coefficients were calculated using the Scipy Stats package in Python and graphs generated with Seaborn 0.9.

Simulation of negative binomial models

We wrote a Python script (Fig2.py, see Appendix or Github link below) to simulate the expected correlation between Dm and Mm under two distinct models of mutagenesis (Fig. 2.2). The script uses the θ (60.42) and μ (138) parameters estimated by glm.nb for the negative binomial model of mismatches segregated to mother (Mm) or daughter (Dm) cells (see FMM.R). (Note that glm.nb actually returns the natural log value for μ (in this case 4.927), which must be exponentiated ($e^{4.927}$) to get 138). In the first model, we assumed that the negative binomial distribution was created by variation in mutation rate along chromatid pairs, so that upon segregation, Dm and Mm from the same division were free to vary within the predicted negative binomial distribution. To simulate this process with Scipy.stats.nbinom.rvs, we converted the θ and μ shape parameters to the n and p inputs (see script for details) for nbinom.rvs and then, for each division, we selected two random values from the distribution to represent the Dm and Mm counts. In the second model, we assumed that the negative binomial was created by a gamma distribution of λ values for a series of Poisson processes acting in different cell divisions. We used Scipy.stats.gamma.rvs to simulate λ values from a gamma distribution with shape and scale parameters derived from those of the negative binomial. The shape parameter for the gamma distribution is simply equal to θ. With variance (v) equal to μ^2/θ, the scale parameter is equal to v/μ. With a random λ from the gamma distribution as an input for Scipy.stats.poisson.rvs, we selected two values from the associated Poisson distribution to serve as Dm and Mm counts for each division. To examine the relationship between Dm and Mm in these different models and the actual data, we performed linear regression with Scipy.stats.linregress and visualized the data and regression line using Seaborn 0.9 regplot.

Simulation of gamma-Poisson-binomial process

We wrote Python scripts to create a Poisson-binomial model of the contributions of semi-conservative DNA replication and mitotic segregation to the over-dispersion of mutations in individual yeast (Fig3ef.py, ExFig6.py) and human cells (Fig3ghij.py) depicted in Fig. 2.3 and Supplementary Fig. 2.6. For yeast simulations, we determined the amount of unmasked DNA on each chromosome in the repeat-masked genome and then divided these values by the total length of unmasked DNA in the haploid genome. The rate of mismatches per haploid genome (69 mismatches/haploid genome/division for *pol3-01/pol3-01 msh6Δ/msh6Δ* cells) was then multiplied in each case by these fractions to obtain the per chromosome rate of mismatch formation. These values were used as input for scipy.stats.poisson.rvs to simulate the number of errors per chromosome in a single division. We created two independent entries per chromosome to model the diploid genome. To mimic the binomial process of mitotic segregation, we then multiplied the number of simulated errors on each chromosome by a randomly chosen 1 or 0. Finally, we summed the mutation counts from all chromosomes to obtain the total number of new mutations per cell division. To create a gamma-Poisson-binomial model, we selected a value for lambda at each division from the gamma distribution described in Fig. 2.2 rather than using a constant rate for mismatch formation. As a control we performed the above simulation without the binomial process, using the mutation rate per haploid genome (34.5 mutations/haploid genome/division). We used the same approach for the human simulations except that we multiplied the fraction of each human chromosome of the total genome (GRCh38) by

a mismatch rate comparable to that observed with *pol3-01/pol3-01 msh6Δ/msh6Δ* yeast: 69 mismatches/ haploid yeast genome/division x (3.03×10^9 bp/human haploid genome / 11×10^7 bp/yeast haploid genome) = 1900 mismatches/human haploid genome/division. We compared the resulting distribution to that from a Poisson distribution with λ equal to 950 mutations/haploid genome. To simulate the diversity in mutation burdens that this process generates, we summed the simulated mutation counts for individual lines from 30 divisions.

Figures

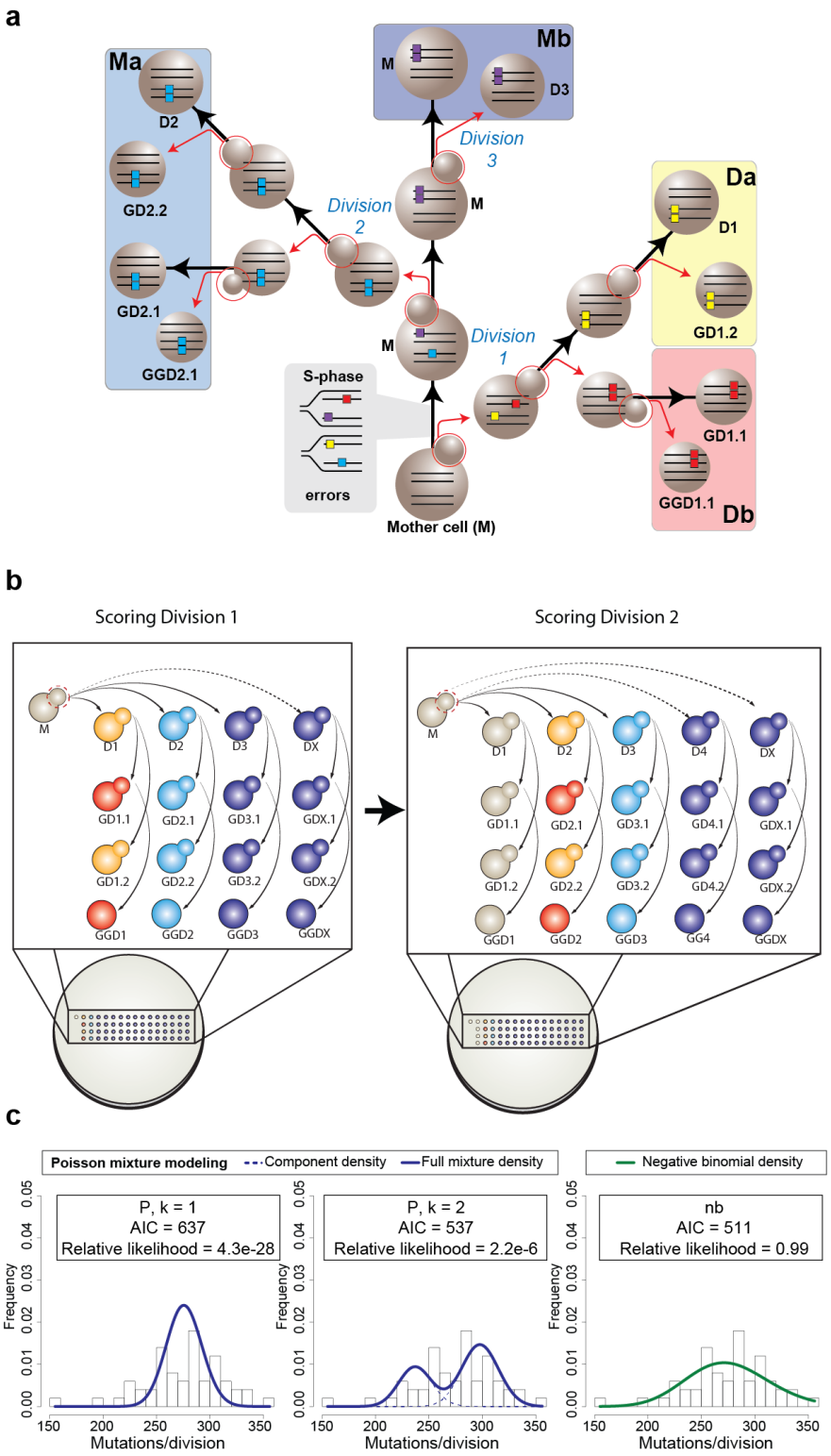


Fig. 2.1: Mutator DNA polymerase errors at single cell resolution

a, Isolation of single cell pedigrees. Using microdissection, the founding mother (M), daughter (e.g. D1), granddaughter (e.g. GD1.1, GD1.2) and great-granddaughter (e.g.

GGD1.1) cells from each maternal division (*n* = 50) are separated (red arrows) and moved to isolated regions on the plate to form colonies, which are then sequenced. Polymerase errors arising during the initial S-phase are passed on to four segregant groups, highlighted by large colored boxes (Da, Db, Ma, Mb), the sum of which represents the full error count for that division. Large spheres connected by black arrows represent the same cell through multiple divisions. Small spheres circled in red represent budding daughter cells; parallel lines in cells, double-stranded DNA; colored boxes on lines, polymerase errors. **b**, Scoring full error counts from sequential divisions. Arrows depict movement of each dissected daughter cell and their descendants to unique positions on the plate to form sublineages. Color-coding indicates cells that will form colonies used for the Da (yellow), Db (red), Ma (blue), and Mb (purple) segregant groups in a given division (See AH121 in Supplementary Data Fig. 2.2 for an example). Segregant group identities shift one sublineage to the right with each division. **c**, Fitting the distributions of full error counts from diploid *pol3-01/pol3-01 msh6Δ*/*msh6Δ* divisions to different models. k = 1, single Poisson; k = 2, two-Poisson; nb, negative binomial; AIC, Akaike information criterion.

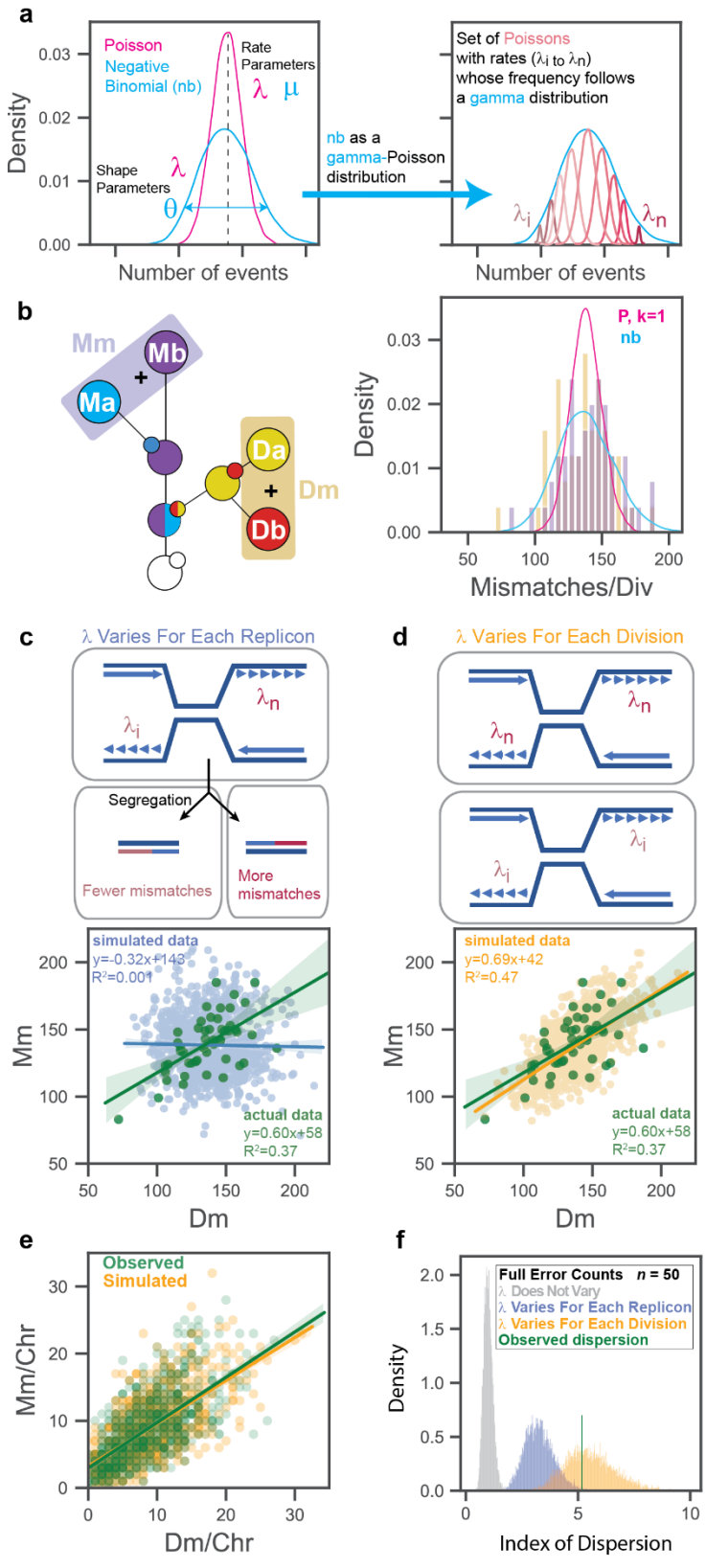

a
Poisson
Negative
Binomial (nb)
Rate
Parameters
λ μ
Shape
Parameters
λ
θ
Density
Number of events
nb as a
gamma-Poisson
distribution
Set of Poissons
with rates (λi to λn)
whose frequency follows
a gamma distribution
λi
λn
Number of events
b
Mm
Mb
Ma
Da
Db
Dm
P, k=1
nb
Density
Mismatches/Div
c
λ Varies For Each Replicon
λn
λi
Segregation
Fewer mismatches
More
mismatches
simulated data
y=-0.32x+143
R²=0.001
actual data
y=0.60x+58
R²=0.37
Mm
Dm
d
λ Varies For Each Division
λn
λi
simulated data
y=0.69x+42
R²=0.47
actual data
y=0.60x+58
R²=0.37
Mm
Dm
e
Observed
Simulated
Mm/Chr
Dm/Chr
f
Full Error Counts n = 50
λ Does Not Vary
λ Varies For Each Replicon
λ Varies For Each Division
Observed dispersion
Density
Index of Dispersion

Fig. 2.2: Evidence that mutation rate varies between divisions

a, The negative binomial as a gamma-Poisson distribution. The gamma distribution takes the same shape parameter (θ) as the negative binomial and describes the variation in the rate parameter (λ) of a continuous mixture of Poisson distributions. **b**, Schematic of single cell lineage showing summing of segregant groups to determine the number of mismatches segregated to the mother (Mm) or daughter (Dm) in a single division. Actual distributions are represented by gold (Dm) and purple (Mm) bars. Lines depict models of data: pink, single Poisson (P, k = 1); aqua, negative binomial (nb). **c**, **d**, Correlations between Mm and Dm counts from actual data (green, *n* = 50) and simulations (*n* = 1000) under two different models. In **c**, top panel depicts a cell with converging replication forks from two replicons with different mutation rates. Bottom panel shows correlation of simulated Mm and Dm values (blue) drawn from the full negative binomial and their linear regression. In **d**, top panel depicts two cells replicating DNA with different mutation rates. Bottom shows correlation of simulated Mm and Dm values (orange) and their linear regression. **e**, Correlation between the number of mismatches per chromosome segregated to Mother (Mm) or Daughter cells (Dm). green, observed counts; orange, simulated counts from model in (**d**). **f**, Simulated index of dispersion of full replication error counts from small cohorts (n = 50) assuming the models from (**c** and **d**).

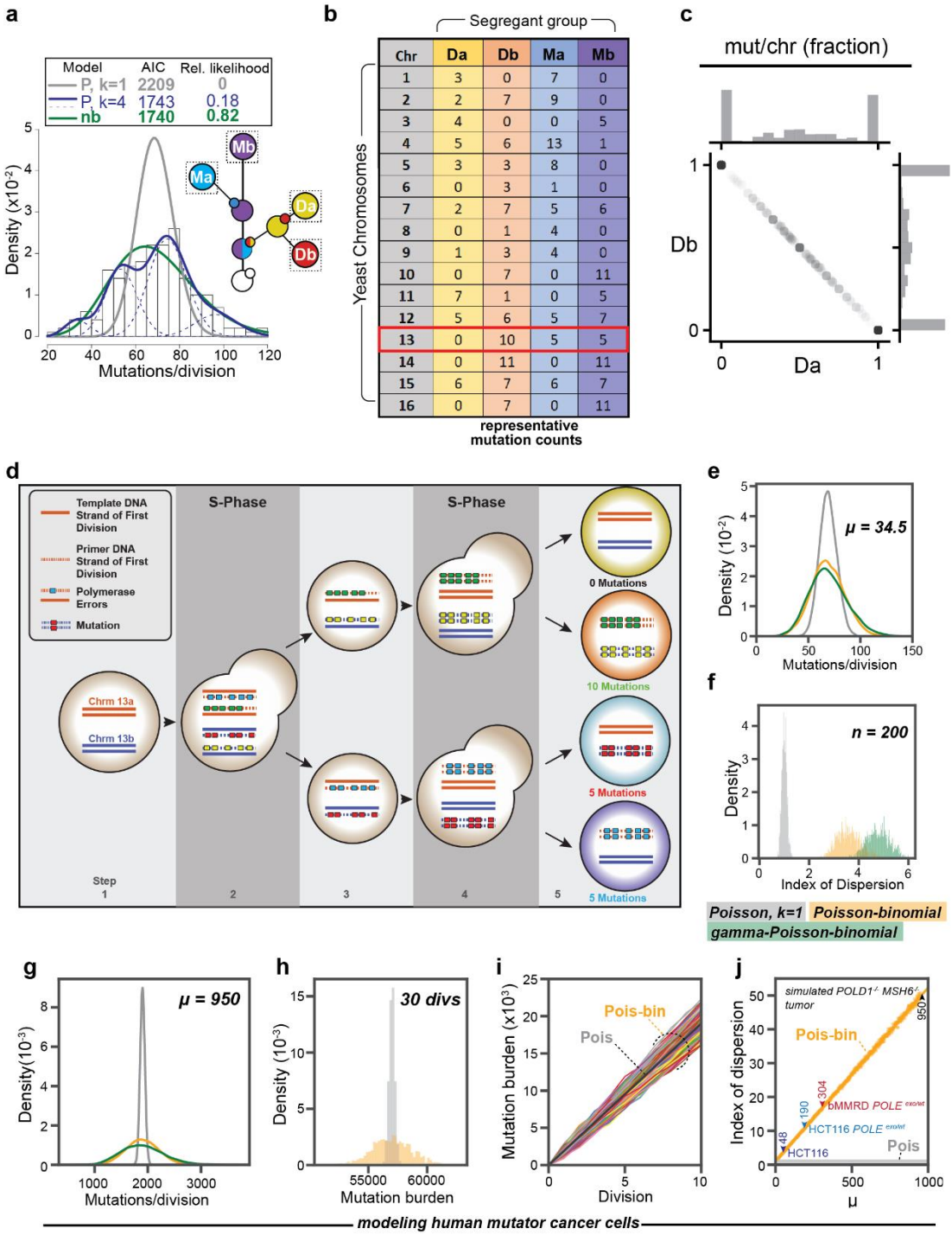


Fig. 2.3: Asymmetric segregation broadens the distribution of mutation burden in mutator cell populations.

a, Combined distribution of new mutations arising in the Da, Db, Ma, and Mb segregant groups (see inset) from *pol3-01/pol3-01 msh6Δ/msh6Δ* lineages (*n = 200*). Key of models (top): gray line, single Poisson (P, k = 1); blue lines, four-Poisson (P, k = 4), green line, negative binomial (nb). AIC, Akaike information criterion. **b**, Table of

representative mutation counts from one division of a diploid mutator cell. Columns represent different segregant groups; rows, the chromosome (chr) number; values, the total number of new mutations found on homologous chromosome pairs. Red box indicates a chromosome with both asymmetric and equal sharing of mutations. **c,** Segregation of double-stranded mutations between Da and Db. For each division, the fraction of mutations observed in Da or Db on each chromosome was determined and then plotted against each other. **d**, Asymmetric segregation of new mutations: 1) Two homologous chromosomes in mother cell (orange and blue lines) prior to scored division. 2) During the first S-phase, mutator Pol δ generates errors (colored boxes) in the nascent strands (dashed lines). 3) At segregation, mother and daughter each inherit two chromosomes with mismatches. 4) In the next S-phase, strands with mismatches produce new double-stranded mutations, while error-free strands do not. 5) Segregation results in cells with 0, 1, or 2 chromosomes with new double-stranded mutations. **e**, Simulated distributions of mutations/division at a rate of μ = 34.5 (*n* = 10000) assuming a single Poisson process (grey), a Poisson-binomial process (orange), or a gamma-Poisson-binomial process (green). **f**, Variation in the index of dispersion of simulated data from the 3 models (*n* = 200) over 1000 iterations. **g**, Simulated distribution of mutations/division in human ultra-mutator cells assuming a mutation rate (*μ* = 950, *n* = 10000) comparable to *pol3-01/pol3-01 msh6Δ*/*msh6Δ* yeast and a single Poisson process (grey), Poisson-binomial process (orange), or gamma-Poisson-binomial process (green). **h**, The cumulative mutation burden of a human ultra-mutator cell after 30 simulated divisions with (orange) and without (grey) asymmetric segregation. **i**, Simulated trajectory of mutation burden of human mutator tumor cells (Colored lines, *n* = 1000) undergoing a Poisson-binomial process compared to a Poisson process (black line). **j**, Change in the index of dispersion under a Poisson-Binomial process (orange line) compared to the static index of dispersion under a Poisson process (grey line at bottom) with an increasing mutation rate. Colored markers represent estimated mutation rates for clinically relevant mutator-driven HCT116-derived mammalian cancer cell lines(Hodel et al., 2018) and a tumor from a patient with biallelic MMR deficiency (bMMRD)(Shlien et al., 2015).

Supplementary Data

Supplementary Information

Haploid *pol2-4 msh6Δ* Mutators

Prior to switching to stronger diploid mutators, we first obtained full replication error counts for 44 *pol2-4 msh6Δ* divisions from 7 independent lineages, encompassing 308 mutations (Supplementary Table 2.1). We sequenced only those clones that would contribute to a full error count (Supplementary Fig. 2.1). Since our previous study suggested that mutations arose in *pol2-4 msh6Δ* mother cells at a rate of 0.4 or 4 mutations/genome/division(Kennedy et al., 2015), with full replication error counts, the

volatility model predicts two well-separated Poisson distributions centered around 1.6 and 16 replication errors per division. Instead, we observed a single distribution centered around 6.5 (±3.9) replication errors per division. The distribution of full replication error counts in *pol2-4 msh6Δ* cells had a D̂ of 2.2, which is consistent with a less pronounced volatility of the *pol2-4 msh6Δ* mutator phenotype. In keeping with this interpretation, fitting these data to different probability distributions revealed they matched a negative binomial better than a single or two-Poisson mixture as judged by AIC. Parsing this data into the number of mutations per individual cell division produces a distribution (N=176) that fits a single Poisson with a rate of 1.75 mutations/division. This finding does not negate the hypothesis of a mild mutator volatility based on the full replication error counts. The expected dispersions of mutations in *pol2-4 msh6Δ* haploid cells (n=176, λ=1.75) were comparable for the Poisson-binomial (D̂ = 1.13±0.12) and Poisson (D̂ = 1.0±0.1) models (Supplementary Fig. 2.6). The rate of 1.75 mutations/division lies almost directly between the predicted underlying rates from our published two-Poisson Model[1]. Thus, our previous distribution likely contained a preponderance of cell divisions with this intermediate mutation rate. The high number of divisions in that earlier dataset with no mutations could have partly been the result of a biological "zero-inflation" due to the unequal sharing of mutations described in Fig. 2.3 for *pol3-01/pol3-01 msh6Δ/msh6Δ* cells. If so, why are there fewer cells with 0 mutations in the current distribution? We suspect that the stringent requirement of eight viable clones to obtain a full replication error count may have introduced an ascertainment bias. Due to unequal sharing of mutations, members of the lineage with the highest number of mutations may fail to form a colony. The reciprocal clones with no errors from that same division would also not be scored. This potential ascertainment bias would affect our estimates of mutator volatility, since divisions with a higher mutation rate are more likely to have at least one progeny fail to form a colony.

Colonies Not Included In Analysis

Many complete sub-lineages (comprised of d, gd1, gd2, and ggd) were not sequenced because inviability later in the lineage prevented us from gaining a full replication error count. For instance, full replication error counts for AH120 divisions that yielded d3 and d4 are not possible because the d5 sublineage was completely inviable (Supplementary Fig. 2.1). Likewise, sometimes colonies within informative sub-lineages (e.g. AH119 gd8-2, AH120 gd10-2/ggd10, AH121 gd9-1) were not sequenced because they were not required for a full replication error count. In some cases (AH156 d14, AH160 d5, AH157 d8), colonies that provided identical information on a division segregant subgroup were sequenced when the preferred colony failed to form a viable clone (Supplementary Fig. 2.2). In other cases (AH158 gd4-2/ggd4, AH158 gd7-2/ggd7, AH162 gd6-2/ggd6), colonies had poor sequencing coverage and were censored from the analysis. In rare cases, sequence analysis and review of dissection notes suggested an assignment error occurred; however, our careful analysis of the patterns of shared mutations enabled post-hoc deconvolution of the events. In one example, in the 8th division of AH156, we separated 3 cells from the mother. The two larger,

similarly sized cells were both clearly daughter cells and the smaller cell was a granddaughter cell, although it was not clear which was its parent. We moved the daughter cells to the d8 (AH15629) and d9 positions (AH15633) and placed the smaller cell below in the gd8-1 position (AH15630). We then proceeded to isolate the remaining members of the lineage. Sequencing analysis of the resulting colonies revealed that AH15633 was in fact d8, AH15629 was d9, and AH15630, the daughter of AH15633, not AH15729. This error meant we had to reorder the lineage. AH15631 was gd9-1 not gd8-2. AH15632, daughter of AH15630, was ggd8 and AH15634 was gd8-2. Since we didn't realize AH15631 was gd9-1, we failed to dissect her first daughter to serve as ggd9 and consequently were unable to obtain a full replication error count for the 9th division. In another example, AH160 division 2 was censored entirely from analysis after finding from the pattern of mutations that AH16005 was a granddaughter derived from d1 of this lineage (AH16001).

Distribution of Mutations and Spectrum

Plotting the mutations scored from all divisions of *pol3-01/pol3-01 msh6Δ/msh6Δ* mutator mother cells reveals mutations were generated across much of the unmasked portions of the sequenced genome (Supplementary Fig. 2.3a). Upon investigation, the few tracts of unmasked chromosomes lacking mutations are likely artifacts of regions of low sequence coverage which consistently fell below our target thresholds for quality and depth in at least one or more members of a lineage.

As expected, C→T mutations are the most abundant single nucleotide substitutions, followed by T→C and C→A. The trinucleotide context reveals a prominent peak of C→A mutations at a TCT context, a hallmark of proofreading deficiency(Alexandrov et al., 2013; Shinbrot et al., 2014), as well as a peak at CCT (Supplementary Fig. 2.3b).

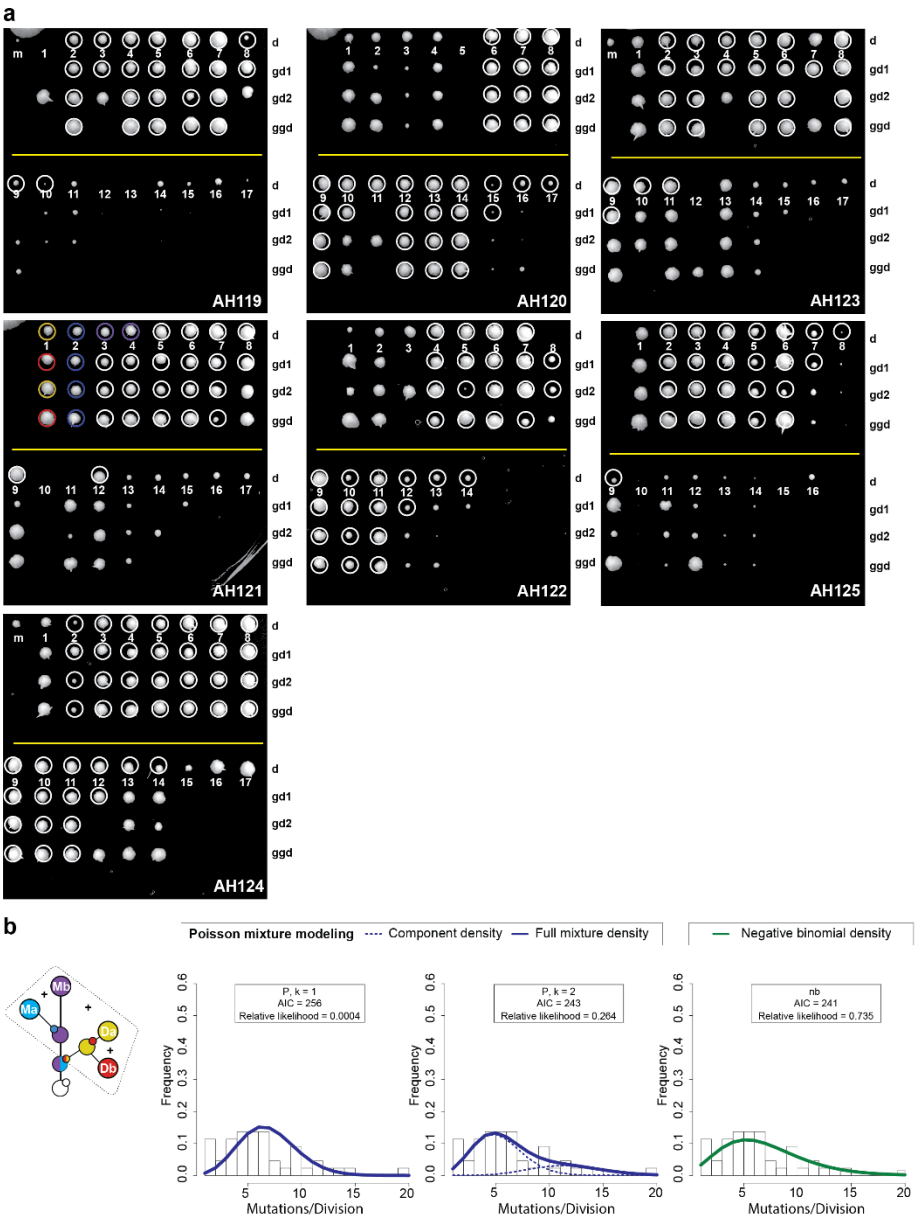
Supplementary Table 2.1

Lineage[a]	Scored Sites[b]	Mutations[c]	Divisions[d]	Mutation Rate[e]
pol2-4 msh6				
119	11,080,506	33	5	0.006
120	11,120,599	51	7	0.0066
121	11,203,731	38	6	0.0057
122	10,909,400	59	7	0.0077
123	11,099,548	24	4	0.0054
124	10,094,023	58	10	0.0057
125	10,897,295	45	5	0.0083
Total		*308*	*44*	

Mean (stdev ±)	10,915,014 (3.51 x 10^5)	44 (12)	6.3 (1.8)	0.0065 (0.001)
pol3-01/pol3-01 msh6Δ/msh6Δ				
151	10,541,044	1,597	6	2.53
153	10,636,360	1,582	6	2.48
156	9,521,963	2,976	10	3.13
157	10,594,546	1,599	6	2.52
158	9,818,623	1,347	5	2.74
160	9,679,326	1,543	5	3.19
162	8,578,576	3,157	12	3.07
Total		*13,801*	*50*	
Mean (stdev ±)	9,910,063 (6.95 x 10^5)	1,972 (699)	7 (2.5)	2.81 (0.29)

[a] Lineage refers to descendants of the same mother cell. See Supplementary Figs. 1 and 2 for images of colonies and Supplementary Data 1 and 2 for mutations.
[b] Scored sites refers to the number of genomic nucleotide positions confidently scored in all members of the lineage.
[c] The total number of independent mutations identified within each lineage.
[d] The number of divisions with full replication error counts (see Fig. 2.1).
[e] Mutation rate (x 10^{-5} mutations/bp/division): the number of mutations divided by the total number scored sites divided by the number of divisions.

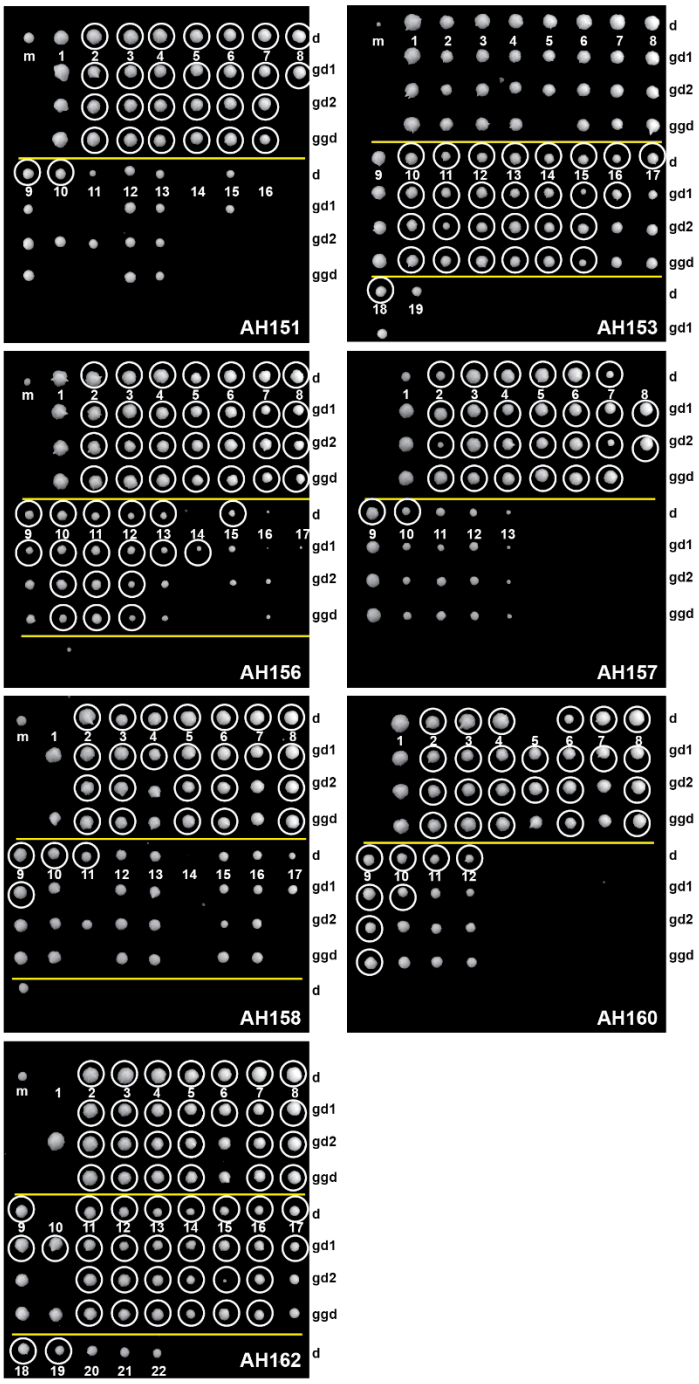
Supplementary Figures



Supplementary Fig. 2.1: *pol2-4 msh6Δ* lineages.

a, Photographs of agar plates with colonies formed from single cell lineages. The lineage number is given in the lower right-hand corner. Locations of rows of daughter (d), first granddaughter (gd1), second granddaughter (gd2), and great-granddaughter (ggd) colonies are given on the right-hand side of the images. Sublineage number is indicated below each daughter colony. Circles indicate sequenced colonies. Colored circles in Lineage AH121 illustrate segregant groupings for the first division (Fig. 2.1). Yellow line divides earlier sublineages from later sublineages. Gaps in colony growth reflect lethality. **b**, Fitting the distributions of full error counts from haploid *pol2-4 msh6Δ*

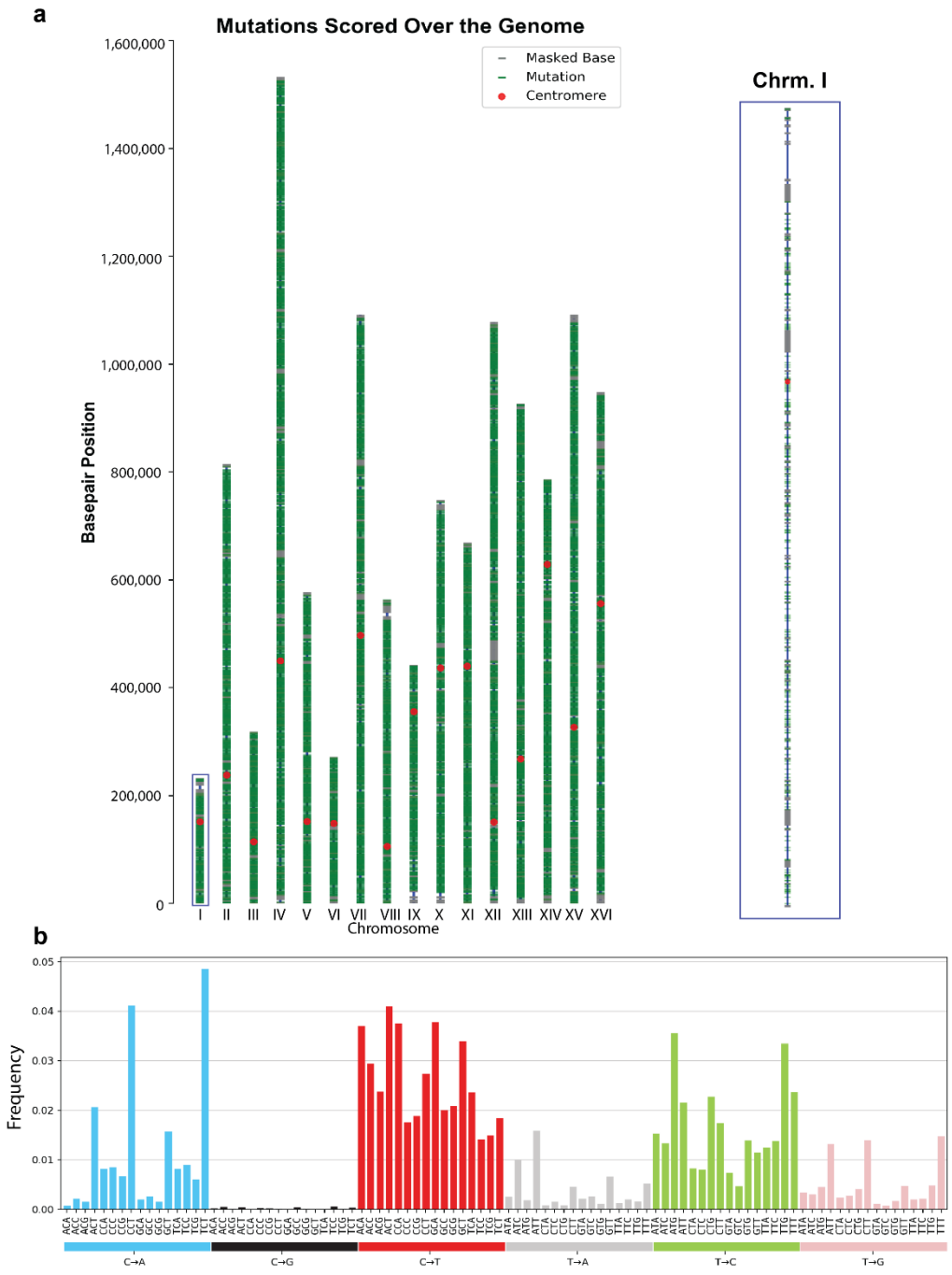
divisions (*n* = 44) to alternative models. k=1, single Poisson; k=2, two-Poisson; nb, negative binomial; AIC, Akaike information criterion.



Supplementary Fig. 2.2: *pol3-01/pol3-01 msh6Δ/msh6Δ* lineages.

Photographs of agar plates with colonies formed from single cell lineages. The lineage number is given in the lower right-hand corner. Locations of rows of daughter (d), first granddaughter (gd1), second granddaughter (gd2), and great-granddaughter (ggd) colonies are given on the right-hand side of the images. Sublineage number is indicated

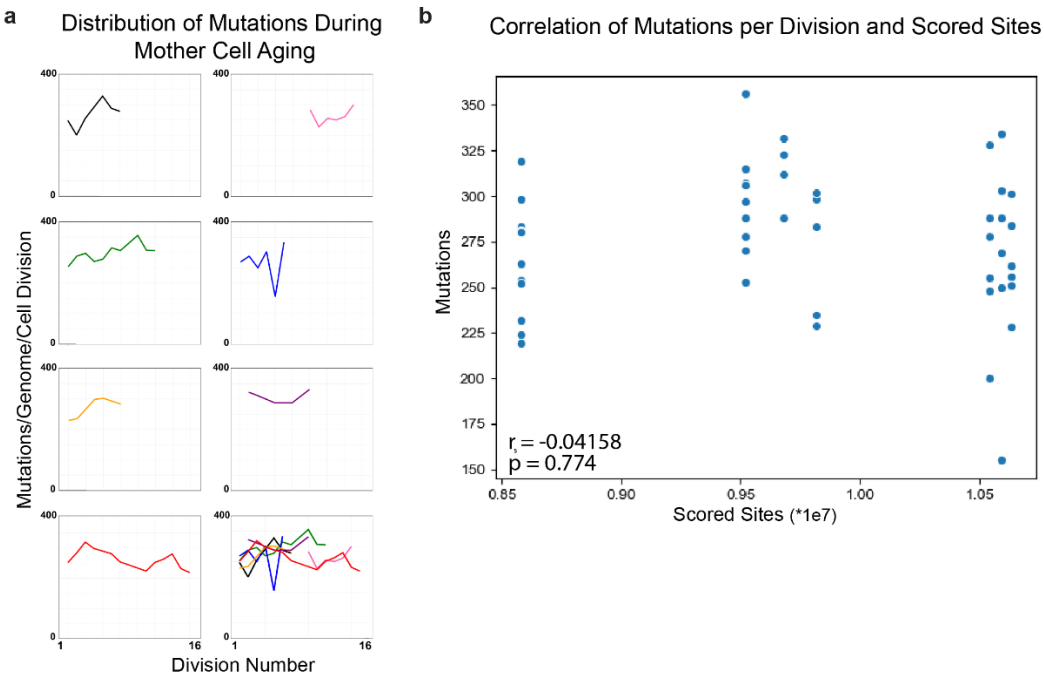
below each daughter colony. Yellow line divides earlier sublineages from later sublineages. Gaps in colony growth reflect lethality.



Supplementary Fig. 2.3: Distribution of mutations and spectra in *pol3-01/pol3-01 msh6Δ/msh6Δ* lineages.

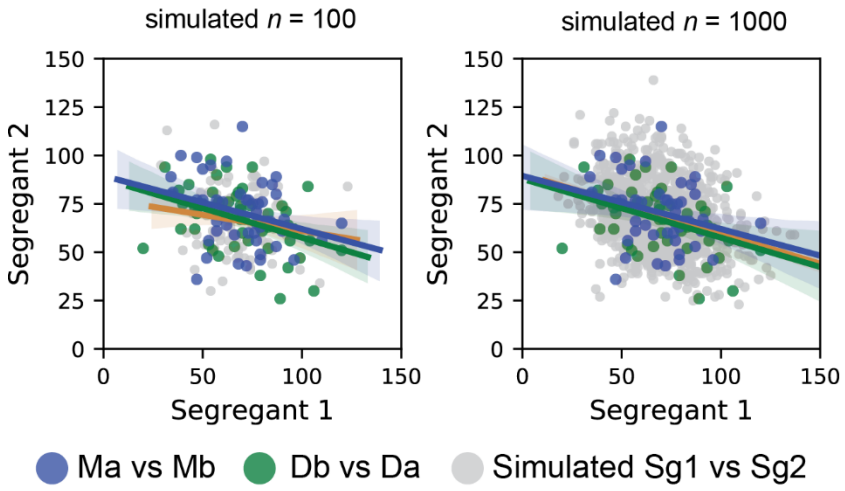
a, Genome level distribution of mutations (green) over yeast chromosomes (blue lines). 13,801 mutations pooled from 50 scored divisions of *pol3-01/pol3-01 msh6Δ/msh6Δ* diploid mother cells, representing approximately 1 mutation per 1000 bases of the yeast

genome. Close up view of representative chromosome I (right). Masked bases are represented by grey ticks. **b**, 96-trinucleoide mutation spectra context of all mutations (spectrum) by frequency that arose over 50 divisions of *pol3-01/pol3-01 msh6Δ/msh6Δ* diploid mother cells, generated using the snv-spectrum program (https://github.com/aroth85/snv-spectrum).



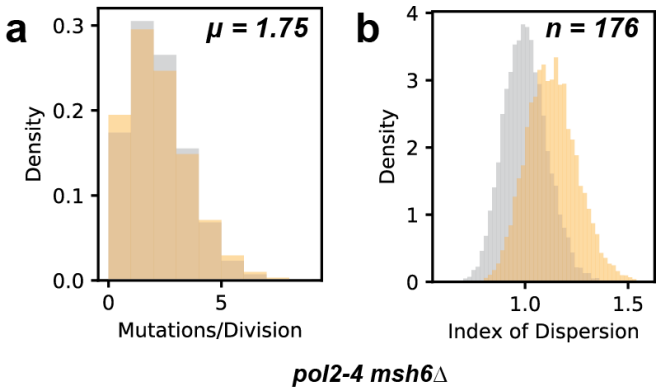
Supplementary Fig. 2.4: Excluding simple explanations for *pol3-01/pol3-01 msh6Δ/msh6Δ* mutator volatility.

a, Mutation counts and maternal age. The total mutation counts from individual divisions is plotted relative to maternal age (Division number). **b**, Mutation counts and size of scored genome. The proportion of the genome scored in all members of a lineage varies between lineages due to sequencing depth and number of lineage members, but is not correlated with mutation counts (Spearman Correlation).



Supplementary Fig. 2.5: Correlations between segregation groups.

X/Y Scatter plots of segregant group pairs (Db/Da, Ma/Mb, *n* = 50 for each) from *pol3-01/pol3-01 msh6Δ/msh6Δ* divisions are plotted alongside simulated data (*n* = 100, left; *n* = 1000, right). Segregant 1 corresponds to Db or Ma (daughters). Segregant 2 corresponds to Da or Mb (mothers). The furthest outlier point for Ma/Mb (65,120) in the upper-half of the plot comes from Division 8 which produced a Db/Da point located at (120,51; on the green line) that yielded similar mismatch totals (Dm, 171; Mm, 185). This suggests the division had a high mutation rate. Ma/Mb (47,36) and Db/Da (20, 52) segregant pairs in the lower left-hand quadrant also appear as outliers. Both pairs are derived from Division 15, leading to the conclusion that the mutation rate in that division was inherently low.



Supplementary Fig. 2.6: Simulation of *pol2-4 msh6Δ* haploid mutagenesis

a, The simulated distribution of mutations from haploid *pol2-4* m*sh6Δ* cells at a rate of μ = 1.75 mutations/division (*n* = 10,000) assuming a single Poisson process (grey) or a Poisson-binomial process (orange). **b**, Variation in the index of dispersion of simulated data from Poisson and Poisson-binomial models (*n* = 176) over 10,000 iterations.

Chapter Two References

Alexandrov, L.B., Nik-Zainal, S., Wedge, D.C., Aparicio, S.A.J.R., Behjati, S., Biankin, A.V., Bignell, G.R., Bolli, N., Borg, A., Børresen-Dale, A.-L., et al. (2013). Signatures of mutational processes in human cancer. Nature *500*, 415–421.

Barbari, S.R., and Shcherbakova, P.V. (2017). Replicative DNA polymerase defects in human cancers: Consequences, mechanisms, and implications for therapy. Dna Repair *56*, 16–25.

Beckman, R.A., and Loeb, L.A. (2017). Evolutionary dynamics and significance of multiple subclonal mutations in cancer. Dna Repair *56*, 7–15.

Brody, Y., Kimmerling, R.J., Maruvka, Y.E., Benjamin, D., Elacqua, J.J., Haradhvala, N.J., Kim, J., Mouw, K.W., Frangaj, K., Koren, A., et al. (2018). Quantification of somatic mutation flow across individual cell division events by lineage sequencing. Genome Res *28*, 1901–1918.

Burnham, K.P., and Anderson, D.R. (2004). Multimodel Inference. Sociol Method Res *33*, 261–304.

Cairns, J. (1975). Mutation selection and the natural history of cancer. Nature *255*, 197–200.

Church, D.N., Briggs, S.E.W., Palles, C., Domingo, E., Kearsey, S.J., Grimes, J.M., Gorman, M., Martin, L., Howarth, K.M., Hodgson, S.V., et al. (2013). DNA polymerase ε and δ exonuclease domain mutations in endometrial cancer. Hum Mol Genet *22*, 2820–2828.

Claussin, C., Porubský, D., Spierings, D.C., Halsema, N., Rentas, S., Guryev, V., Lansdorp, P.M., and Chang, M. (2017). Genome-wide mapping of sister chromatid exchange events in single yeast cells using Strand-seq. Elife *6*, e30560.

DePristo, M.A., Banks, E., Poplin, R., Garimella, K.V., Maguire, J.R., Hartl, C., Philippakis, A.A., Angel, G. del, Rivas, M.A., Hanna, M., et al. (2011). A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat Genet *43*, 491–498.

Dowsett, I.T., Sneeden, J.L., Olson, B.J., McKay-Fleisch, J., McAuley, E., Kennedy, S.R., and Herr, A.J. (2021). Rate volatility and asymmetric segregation diversify mutation burden in cells with mutator alleles. Commun Biology *4*, 21.

Faust, G.G., and Hall, I.M. (2014). SAMBLASTER: fast duplicate marking and structural variant read extraction. Bioinformatics *30*, 2503–2505.

Foster, P.L. (2006). Methods for Determining Spontaneous Mutation Rates.

Furusawa, M. (2014). The disparity mutagenesis model predicts rescue of living things from catastrophic errors. Frontiers Genetics *5*, 421.

Herr, A.J., Kennedy, S.R., Knowels, G.M., Schultz, E.M., and Preston, B.D. (2014). DNA Replication Error-Induced Extinction of Diploid Yeast. Genetics *196*, 677–691.

Hodel, K.P., Borja, R. de, Henninger, E.E., Campbell, B.B., Ungerleider, N., Light, N., Wu, T., LeCompte, K.G., Goksenin, A.Y., Bunnell, B.A., et al. (2018). Explosive mutation accumulation triggered by heterozygous human Pol ε proofreading-deficiency is driven by suppression of mismatch repair. Elife *7*, e32692.

Jackson, A.L., and Loeb, L.A. (1998). The mutation rate and cancer. Genetics *148*, 1483–1490.

Kane, D.P., and Shcherbakova, P.V. (2014). A common cancer-associated DNA polymerase ε mutation causes an exceptionally strong mutator phenotype, indicating fidelity defects distinct from loss of proofreading. Cancer Res *74*, 1895--1901.

Kennedy, S.R., Schultz, E.M., Chappell, T.M., Kohrn, B., Knowels, G.M., and Herr, A.J. (2015). Volatility of Mutator Phenotypes at Single Cell Resolution. Plos Genet *11*, e1005151.

Koboldt, D.C., Zhang, Q., Larson, D.E., Shen, D., McLellan, M.D., Lin, L., Miller, C.A., Mardis, E.R., Ding, L., and Wilson, R.K. (2012). VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res *22*, 568–576.

Lada, A.G., Stepchenkova, E.I., Waisertreiger, I.S.R., Noskov, V.N., Dhar, A., Eudy, J.D., Boissy, R.J., Hirano, M., Rogozin, I.B., and Pavlov, Y.I. (2013). Genome-Wide Mutation Avalanches Induced in Diploid Yeast Cells by a Base Analog or an APOBEC Deaminase. Plos Genet *9*, e1003736.

Lee, M.B., Dowsett, I.T., Carr, D.T., Wasko, B.M., Stanton, S.G., Chung, M.S., Ghodsian, N., Bode, A., Kiflezghi, M.G., Uppal, P.A., et al. (2019). Defining the impact of mutation accumulation on replicative lifespan in yeast using cancer-associated mutator phenotypes. Proc National Acad Sci 201815966.

Leisch, F. (2004). FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression inR. J Stat Softw *11*.

Levine, D.A., Getz, G., Gabriel, S.B., Cibulskis, K., Lander, E., Sivachenko, A., Sougnez, C., Lawrence, M., Kandoth, C., Dooling, D., et al. (2013). Integrated genomic characterization of endometrial carcinoma. Nature *497*, 67–73.

Li, H., and Durbin, R. (2009). Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics *25*, 1754–1760.

Li, H.-D., Cuevas, I., Zhang, M., Lu, C., Alam, M.M., Fu, Y.-X., You, M.J., Akbay, E.A., Zhang, H., and Castrillon, D.H. (2018). Polymerase-mediated ultramutagenesis in mice produces diverse cancers with high mutational load. J Clin Invest *128*, 4179–4191.

Luria, S.E., and Delbrück, M. (1943). Mutations of Bacteria from Virus Sensitivity to Virus Resistance. Genetics *28*, 491–511.

Lynch, H., Lynch, P., Lanspa, S., Snyder, C., Lynch, J., and Boland, C. (2009). Review of the Lynch syndrome: history, molecular genetics, screening, differential diagnosis, and medicolegal ramifications. Clin Genet *76*, 1–18.

Lynch, M., Sung, W., Morris, K., Coffey, N., Landry, C.R., Dopman, E.B., Dickinson, W.J., Okamoto, K., Kulkarni, S., Hartl, D.L., et al. (2008). A genome-wide view of the spectrum of spontaneous mutations in yeast. Proc National Acad Sci *105*, 9272–9277.

Mertz, T.M., Sharma, S., Chabes, A., and Shcherbakova, P.V. (2015). Colon cancer-associated mutator DNA polymerase δ variant causes expansion of dNTP pools increasing its own infidelity. Proc National Acad Sci *112*, E2467--E2476.

Muzny, D.M., Bainbridge, M.N., Chang, K., Dinh, H.H., Drummond, J.A., Fowler, G., Kovar, C.L., Lewis, L.R., Morgan, M.B., Newsham, I.F., et al. (2012). Comprehensive molecular characterization of human colon and rectal cancer. Nature *487*, 330–337.

Nik-Zainal, S., Alexandrov, L.B., Wedge, D.C., Loo, P.V., Greenman, C.D., Raine, K., Jones, D., Hinton, J., Marshall, J., Stebbings, L.A., et al. (2012). Mutational Processes Molding the Genomes of 21 Breast Cancers. Cell *149*, 979–993.

Palles, C., Cazier, J.-B., Howarth, K.M., Domingo, E., Jones, A.M., Broderick, P., Kemp, Z., Spain, S.L., Guarino, E., Salguero, I., et al. (2013). Germline mutations affecting the proofreading domains of POLE and POLD1 predispose to colorectal adenomas and carcinomas. Nat Genet *45*, 136.

Petljak, M., Alexandrov, L.B., Brammeld, J.S., Price, S., Wedge, D.C., Grossmann, S., Dawson, K.J., Ju, Y.S., Iorio, F., Tubio, J.M.C., et al. (2019). Characterizing Mutational Signatures in Human Cancer Cell Lines Reveals Episodic APOBEC Mutagenesis. Cell *176*, 1282-1294.e20.

Robinson, J.T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E.S., Getz, G., and Mesirov, J.P. (2011). Integrative genomics viewer. Nat Biotechnol *29*, 24–26.

Sherman, F. (2002). Getting started with yeast. Methods Enzymol *350*, 3–41.

Shinbrot, E., Henninger, E.E., Weinhold, N., Covington, K.R., Göksenin, A.Y., Schultz, N., Chao, H., Doddapaneni, H., Muzny, D.M., Gibbs, R.A., et al. (2014). Exonuclease mutations in DNA polymerase epsilon reveal replication strand specific mutation patterns and human origins of replication. Genome Res *24*, 1740--1750.

Shlien, A., Campbell, B.B., Borja, R. de, Alex, rov, L.B., Merico, D., Wedge, D., Loo, P.V., Tarpey, P.S., Coupl, et al. (2015). Combined hereditary and somatic mutations of

replication error repair genes result in rapid onset of ultra-hypermutated cancers. Nat Genet *47*, 257--262.

van Wietmarschen, N., and Lansdorp, P.M. (2016). Bromodeoxyuridine does not contribute to sister chromatid exchange events in normal or Bloom syndrome cells. Nucleic Acids Res *44*, 6787–6793.

Venables, W.N., Ripley, B.D., Venables, W.N., and Ripley, B.D. (2002). Modern Applied Statistics with S. 41–68.

Villanueva, R.A.M., and Chen, Z.J. (2019). ggplot2: Elegant Graphics for Data Analysis (2nd ed.). Meas Interdiscip Res Perspectives *17*, 160–167.

Wagenmakers, E.-J., and Farrell, S. (2004). AIC model selection using Akaike weights. Psychon B Rev *11*, 192–196.

Werner, B., and Sottoriva, A. (2018). Variation of mutational burden in healthy human tissues suggests non-random strand segregation and allows measuring somatic mutation rates. Plos Comput Biol *14*, e1006233.

Williams, L.N., Marjavaara, L., Knowels, G.M., Schultz, E.M., Fox, E.J., Chabes, A., and Herr, A.J. (2015). dNTP pool levels modulate mutator phenotypes of error-prone DNA polymerase ε variants. Proc National Acad Sci *112*, E2457–E2466.

Yoshida, R., Miyashita, K., Inoue, M., Shimamoto, A., Yan, Z., Egashira, A., Oki, E., Kakeji, Y., Oda, S., and Maehara, Y. (2011). Concurrent genetic alterations in DNA polymerase proofreading and mismatch repair in human colorectal cancer. Eur J Hum Genet *19*, 320–325.

Youn, A., and Simon, R. (2013). Using passenger mutations to estimate the timing of driver mutations and identify mutator alterations. Bmc Bioinformatics *14*, 363.

Zhang, C.-Z., Spektor, A., Cornils, H., Francis, J.M., Jackson, E.K., Liu, S., Meyerson, M., and Pellman, D. (2015). Chromothripsis from DNA damage in micronuclei. Nature *522*, 179–184.

Chapter Three
The Effect of dNTP Concentration on Volatility of the Mutator Phenotype in Human Cancer Alleles at Single Cell Resolution

Ian T. Dowsett[a], Maxwell A. Tracy[a], Alan J. Herr[a,*]

Author Information
[a]Department of Laboratory Medicine and Pathology, University of Washington, Seattle, WA 98195-7705, USA

Abstract

We previously found evidence that cells exhibit a 'mutator volatility' phenotype that unexpectedly varies the mutation rate significantly from one cell division to the next. This volatility persisted even when accounting for the asymmetric segregation of mutations generated through the basic biological processes of semi-conservative replication of DNA and mitotic segregation of chromosomes. Here we model suspected human cancer alleles in yeast and examine the effect of genetically suppressing dNTP concentrations on the mutator phenotype. We then examine whether the particularly potent pol2-P301R allele displays mutator volatility and whether S-phase checkpoint signaling may play a role in broadening the distribution of errors made during replication. Through the use of whole genome sequencing and lineage tracing we score errors made during the replication of single cell divisions in cells with and without suppressed checkpoint signaling. We find that suppression of dNTP pools consistently limits the number of errors made during replication. Further, the distribution of errors more closely resembles a single broad distribution rather than the set of discrete distributions observed in unsuppressed cells.

Introduction

DNA polymerase Epsilon (Pol ε), the leading strand polymerase (encoded by the *POLE* gene), plays a major role in faithful DNA replication (Pursell *et al.* 2007). The high fidelity of Pol ε relies upon its high accuracy of nucleotide selection and a 3' exonuclease proofreading domain that edits misinsertions, which create mismatched 3' primer termini. Occasional Pol ε extension of these mispairs produce substrates for mismatch repair (MMR). Together polymerase accuracy, proofreading, and MMR limit replication errors to a rate of around 1 in 10^{10} basepairs synthesized (Loeb 1991).

Mutations that inactivate proofreading or MMR produce a mutator phenotype that increase the spontaneous mutation rate throughout the genome. In a unicellular species, such as *Escherichia coli*, a mutator phenotype rapidly drives the adaptation of a population to a novel environment, but, if unresolved, severely compromises fitness in

subsequent generations (Giraud *et al.* 2001). In multicellular organisms, a mutator phenotype can generate genetic diversity in pre-cancerous tissue, providing ample opportunities for affected cells to overcome the innate and evolutionarily conserved barriers that protect the host organism against oncogenesis (Fox *et al.* 2013). In a similar manner, this heterogeneous population provides tumors with the necessary genetic diversity to overcome chemotherapeutic agents.

Mutations affecting the proofreading domain of Pol ε have been repeatedly identified in hypermutated colorectal and endometrial cancers suggesting that proofreading dysfunction in replicative polymerases may be an initiating event in many sporadic or hereditary cancers (Shinbrot *et al.* 2014). These amino acid substitutions include P286R, F367S, V411L, L424V, P436R among others. Although the mutation load in these tumors suggests that they confer a mutator phenotype, this hypothesis has not been confirmed in most cases. The budding yeast, *Saccharomyces cerevisiae*, has been used to demonstrate that the common P286R variant (P301R in yeast) confers a potent mutator phenotype, 50 times higher than simply ablating proofreading function with the well characterized *pol2-4* mutant (yeast Pol2 is homologous to mammalian POLE). This suggests that the P301R variant somehow diminishes nucleotide selectivity in addition to impairing proofreading (Kane and Shcherbakova 2014). Recent evidence suggests that the P301R mutator retains some proofreading activity, but more frequently extends mispairs than simple proofreading ablation (Xing *et al.* 2019). MMR defects are a well-established route to micro-satellite unstable colorectal carcinomas (Peltomaki *et al.* 1993). Though many of the above POLE variants were identified in micro-satellite stable tumors, in some cases mutation of MMR genes may exacerbate its mutator phenotype. This has been demonstrated to contribute to the rapid generation of tumors observed in the childhood cancer syndrome bMMRD (Shlien *et al.* 2015). In this model, germline biallelic MMR deficiency leads to the formation of tumors, which subsequently develop DNA polymerase exonuclease mutations and this interaction rapidly generates multiple, primary, micro-satellite stable tumors with an ultra-hypermutated genome. Mutator subclones, even in other tumor types, may be an important source of genetic heterogeneity by which cancer evades otherwise lifesaving chemotherapy. With these clinical implications in mind, it is critical that we understand how these human *POLE* cancer alleles might synergize with mismatch repair deficiency and identify routes to disrupt this selection-driven process.

Using yeast and a combination of single-cell lineage isolation, clonal outgrowth, and whole genome sequencing, our lab recently found that *pol2-4 msh6* cells appear to assume one of two mutator states that differ by an order of magnitude (Kennedy *et al.* 2015). In chapter two of this dissertation I have shown that this volatility persists when accounting for the asymmetric segregation of mutations in *pol3-01* mutators by summing all the mismatches segregated into four possible segregant groups of single divisions (Dowsett *et al.* 2021). Elucidating the mechanisms underlying this mutator volatility may enable the future development of techniques to manipulate the mutator states of cancer cells and thus the evolution of human tumors. In a separate study, we found that the *pol2-4* mutator phenotype varies proportionally with available dNTPs,

which are regulated as part of the S-phase checkpoint (Williams *et al.* 2015). Putting together these observations, we hypothesized that mutator volatility reflects variation in cellular dNTP pool levels, resulting from intermittent activation of the S-phase checkpoint. To address this hypothesis we developed a yeast genetics-based approach, which models these specific cancer-associated alleles in the context of a genetic background that influences dNTP pools.

Results and Discussion

As a genetics-based method to lower dNTP concentrations we deleted the yeast *DUN1* gene. Dun1 regulates the availability of dNTPs by inhibiting negative regulators of RNR activity. These proteins regulate RNR at a transcriptional level (Crt1), subcellular localization (Dif1), and the assembly of the quaternary structure of the holoenzyme (Sml1). Sml1 binds to the dimer α-α subunit of the tetrameric holoenzyme, restricting the ability of the subunit pair from binding to the β-β dimer and thus preventing enzymatic activity (Elledge and Davis 1990). Dif1 regulates the import of the β-β dimer into the nucleus, thus controlling its availability within the cytoplasm to bind with the α-α subunit (Lee *et al.* 2008). We previously showed that dNTP levels decrease two-fold in *pol2-4 dun1Δ* cells, while mutation rates decrease 4-fold. In *pol2-4 sml1 crt1* cells, dNTP levels increase three-fold and mutation rates increase 8-fold (Williams *et al.* 2015).

To address whether S-phase checkpoint activation is responsible for mutator volatility and assess the influence of Dun1 on cancer-associated mutator alleles we isolated extensive single cell lineages from individual mother cells, followed by clonal outgrowth and whole genome sequencing. Conventional mutation rate measurements rely on scoring gene-inactivating mutations in a selectable marker, such as *CAN1,* over the course of colony development, which masks the mutation rates at a per cell division resolution. A single cell resolution approach of whole genome sequencing colonies formed from all daughters, first and second granddaughters, and the first granddaughters's first daughter allows us to track the emergence of errors and their segregation from each division of the mother cell. This multigenerational technique is crucial to our ability to accurately score emergent mutations. Spontaneous mutations that occur as mispairs during replication become fixed errors during the next replication event and thus identifying those mutations requires sequencing an additional generation. To unambiguously identify true mutations despite the error rate of sequencing technology we only scored those mutations shared by both resulting daughter cells. We anticipated that applying this technique to Dun1 proficient and deficient strong mutators would provide strong evidence as to whether dNTP pools contribute to mutator volatility and may be a potential route of pharmacological intervention. Ultimately, determining how to influence the mutation rate status of mutators may enable the development of interventions capable of shifting a tumor from its native mutation rate: either downwards to a hypomutator state, to slow its evolution,

or even potentially upwards, to a level beyond its survival threshold while minimizing the effect on the mutation rate of uncompromised and healthy tissue.

We selected human *POLE* cancer alleles from the TCGA database and engineered the corresponding mutations into a heterozygous *msh2*Δ*::LEU2* mutant yeast strain derived from parent strain BY4743. *DUN1* deletion mutations were introduced into the resulting *URA3::pol2-x/POL2* heterozygous strains (where *pol2-X* is one of five selected *POL2* variants) by replacement with the *kanMX* transgene (*dun1*Δ::*kanMX*). Mutation rates were determined by fluctuation analysis of canavanine-resistant mutants (Can^r). To facilitate forward mutation rate measurements in diploids (see below), this strain background has been engineered to be heterozygous at the *CAN1* locus (*can1*Δ*::HIS3/CAN1::natMX*). Mutations in the *CAN1* gene can be scored on plates containing canavanine and nourseothricin, which selects for maintenance of the *natMX* transgene. Sporulation of *URA3::pol2-x/POL2, msh2*Δ*::LEU2/MSH2 can1*Δ*::HIS3/CAN1::natMX, dun1*Δ*::KanMX/DUN1* diploids and dissection of the resulting tetrads onto rich media yielded a mixed population of haploid segregant colonies. All colonies were plated on selection plates containing canavanine; at the same time dilutions were plated on genotyping plates to score the presence or absence of transgenes used to create each mutant allele.

Experimental data suggest that the mutator phenotype of some, but not all alleles, is significantly suppressed in the absence of Dun1. Deletion of *DUN1* suppressed the mutation rates conferred by *pol2-F382S* ($\lambda = 6.24$, $p=1.25^{-02}$), *pol2-P451R* ($\lambda = 21.9$, $p=2.84^{-06}$), *pol2-P301R* ($\lambda = 24.3$, $p=8.15^{-07}$), and *pol2-L439V* ($\lambda = 20.4$, $p=6.14^{-06}$) alleles. Deletion of *DUN1* causes the most dramatic effect on the *pol2-P301R* mutator allele, which was the strongest mutator allele, lowering the mutation rate from 3.0×10^{-5} to 2.4×10^{-6} Can^r mutants/cell division (Fig. 3.1a). *DUN1* deletion did not significantly reduce the mutation rate of mismatch repair defective haploid spores with an unmutated *POL2* (Fig. 3.1b). This provides further evidence that *DUN1* deletion and the subsequent dNTP depletion effect on the mutation rate preferentially affects the mutator polymerase.

Tetrad dissection revealed that several *pol2-x msh2Δ* double mutant strains exhibit a synthetic lethal or synthetic sick phenotype (Fig. 3.1c). This is a manifestation of error-induced extinction (EEX), resulting from the attrition of viable cells within the growing clone due to deleterious mutations. In keeping with the above antimutator effect, *dun1*Δ enables the formation of *pol2-L439V msh2Δ and pol2-P451R msh2Δ* double mutant colonies approximately 30x larger than those with a functional *DUN1*. Only a few small colonies, ranging from approximately $1x10^3$ to $3x10^4$ cells, developed from *pol2-F382S msh2Δ dun1Δ* spores, but those with a functional *DUN1* allele form microcolonies of only approximately $5x10^2$ cells (Fig. 3.1d). In contrast, *pol2-V426L*

msh2Δ dun1Δ cells show minimal effect on mutation rate or growth compared to *pol2-V426L msh2Δ DUN1*. Finally, deletion of *DUN1* failed to rescue *pol2-P301R msh2Δ* from *EEX*. The mutation rate for these cells likely remains above the lethal mutation rate threshold for haploid cells, approximately 1 x 10^{-3} Can^r mutants per cell division (Herr *et al.* 2011).

To determine the precise effect of *DUN1* deletion on these cells required switching to using diploid cells which have a higher threshold for maximum tolerable mutation rate (approximately 1×10^{-2} Can^r mutants/cell division) due to the buffering capacity imparted by two copies of each essential gene (Herr *et al.* 2014). Crossing *pol2-x msh2Δ* spores with and without *DUN1* revealed that two copies of the P301R allele was indeed lethal, but was rescued by *DUN1* deletion (Fig. 3.2a). Our results, outlined above, establish that human *POLE* cancer alleles, modeled in haploid and diploid yeast, exhibit mutator phenotypes that synergize with mismatch repair deficiency and are sensitive to dNTP pool depletion. This lays the groundwork to determine in diploid cells if these alleles confer a volatile mutator phenotype, manifested as a bimodal mutation count distribution from individual cell divisions, and whether *dun1Δ* shifts these counts into a unimodal distribution. With this in mind, we next sought to determine which combination of alleles would provide the conditions suitable to test the effects of *DUN1* deletion on mutator volatility. Ideally, we needed a mutator allele with a strong enough phenotype to allow for scoring of greater than 1 mutation per division to avoid the effects zero inflation. We first considered that a mutator hemizygous at the *MSH2* locus might allow for titration of the extreme mutator phenotype of the pol2-P301R variant. We found that mismatch repair remains not significantly altered in functional correction of mispairs in the presence of only a single copy of *MSH2* in the context of most mutator *POL2* alleles (Fig 3.2b). The *pol2-P301R* mutator allele, unlike *MSH2*, imparted a partial phenotype when heterozygous and was capable of forming colonies even in the presence of *DUN1* (Fig. 3.2c). Deletion of *DUN1* in POL2/pol2-P301R *msh2Δ/ msh2Δ* reduced the CAN^r mutation rate by nearly 2-fold from 2.8 x 10^{-4} to 1.5 x 10^{-4} CAN^r mutants/Division (n = 16 and 19, p = 2.01 x 10^{-4}). These distinct but relatively high mutation rates make this combination of alleles an ideal context to test the hypothesis that S-phase checkpoint activation is a key component of mutator volatility.

Next, in order to measure mutator volatility, we dissected tetrads on media that selected for spores with the mutant alleles of interest (*pol2-P301R, msh2Δ, dun1Δ*) and then crossed the resulting haploid cells to obtain homozygous double or triple mutant zygotes. The first daughter from the zygote was used to establish a single-cell lineage while the subsequent offspring of the zygote were transferred by microdissection to a new plate for colony PCR genotyping. The strategy for establishing a single cell lineage from the target mother cell was identical to that in chapter 2 of this dissertation, in which

daughters, first and second granddaughters and first great-granddaughters were dissected to distinct regions of the plate. Each member of the lineage was allowed to form a colony, and each was subjected to whole genome sequencing, allowing us to score the occurrence of new mutations arising at each cell division. This extended pedigree allows us to observe the fixation of replication errors that occur in the mother but are segregated to daughters. After confirming the genotypes of each lineage as either POL2/*pol2-P301R msh2Δ /msh2Δ dun1Δ/dun1Δ* or POL2/*pol2-P301R msh2Δ /msh2Δ DUN1/DUN1* by the integrative genomics viewer (IGV)(Robinson *et al.* 2011), the full error count for each division was summed and recorded (Fig. 3.3a).

As expected, based on the results of the above fluctuation analysis of canavanine-resistant mutants, the average mutation rate for *DUN1*/*DUN1* in terms of mutations/cell division was nearly double that of cells without *DUN1* (98.6 ±32.5 and 53.8 ±22.3 mutations/cell division respectively, Student's t-Test, p = 6.5e-9, Fig. 3.3b). Interestingly, though the distribution of count data from lineages with functional copies of *DUN1* exhibited a higher variance than that cells without (1056.1 vs 498.9), due to lower mean of the latter group, the Index of Dispersion($\hat{D}$) was only slightly lower ($\hat{D}$ = 10.7 and 9.3, respectively). This suggests the distributions of both datasets are overdispersed relative to a single Poisson ($\hat{D}$ = 1). To gain a better understanding of the different effects of mutator volatility on these two populations we again used finite mixture modeling to determine the underlying mixture of Poisson(s) that might fit these data and then compared these models using the Akaike Information Criteria (AIC). As expected from the high Index of Dispersion the distribution of mutation count data for both genotypes were poorly described by a single Poisson (Fig. 3.3c). This is consistent with the hypothesis of mutator volatility as described in Chapter Two.

For *DUN1/ DUN1* cells, the three Poisson model with an AIC score of 350.5 (lower values indicate greater parsimony) outcompeted others including a single Poisson (AIC = 630.6) and a negative binomial (AIC = 357.1). Transforming AIC values into Akaike weighted values as described in Chapter 2, the three Poisson (relative likelihood of 0.9999) and four Poisson (0.135) models significantly outcompete other models, followed by negative binomial (0.0369). Mutation count data for the division of cells without *DUN1* more closely resembled a negative binomial (AIC = 281.3, Akaike weighted value of 0.9999) which narrowly outcompeted a 4 Poisson model (283.7, 0.301) and a single Poisson model performing poorly (630.6, 2.6E-40). These results suggest that both *DUN1/DUN1* and *dun1Δ/dun1Δ* cells in this mutator context are still subject to mutator volatility. *DUN1/DUN1* cells, however, with this dataset seem to exhibit marginally more of a set of discrete mutator states whereas those cells without *DUN1* more closely resemble a single, broadly distributed variable state. This may indicate that S-phase checkpoint activation is important to transition mutator volatility into one of several discrete states.

The yeast *pol2-P301R* mutator allele used in these experiments is a uniquely potent variant and, in the form of the homologous human POLE-P286R variant, is the most commonly identified POLE mutator identified in human cancers. Because of this, this variant is also the most relevant to explore the effects of mutator volatility on human disease. Our previous research of mutator volatility was carried out in mutator models in which DNA polymerase proofreading had been completely abrogated by mutation of two critical residues in the exonuclease domain. The *P301R* allele, however, reportedly maintains a small amount of proofreading function, but this function is severely curtailed by the impedance of the mutant Arginine residue into the exonuclease site of the enzyme thus inhibiting ssDNA binding(Parkash *et al.* 2019; Xing *et al.* 2019). Despite this limited functionality of the proofreading domain, this variant likely maintains a much stronger mutator phenotype than the proofreading deficient *pol2-4* allele due to its propensity to extend mispairs and greater level of activity (Xing *et al.* 2019). Despite these biochemical differences between *pol2-4* and *pol2-P301R,* we find the phenomenon of mutator volatility to be maintained to varying degrees between these two mutator alleles, suggesting it to be a property of DNA polymerase mutators in general. Additionally, we found that mutator volatility may be influenced in part, but not in whole, by S-phase checkpoint signaling, suggesting that volatility is more complex than a simple dNTP concentration dependent reaction. One factor that might impart a more broadly distributed mutation count data than expected could be the heterozygous state of *POL2/pol2-P301R*. If a variable amount of the genome is replicated by one or the other versions of the two polymerases we might expect to see greater volatility of the mutator phenotype than observed with other homozygous strong mutators. If this is the case, moving to a homozygous *pol2-P301R* should compress the index of dispersion of the distribution of mutation count data.

Methods

Strains

DUN1 deletion strains were engineered in BY4743 derived AH401 derived AH5801, AH6201, AH6301, AH6404, and AH11304 strains corresponding to F382S, P451R, V426L, L439V, P301R *POL2* alleles. Construction of AH11304 was previously described (Tracy *et al.* 2020) and construction of other strains was performed in a similar manner. These strains were engineered to be heterozygous for *POL2* putative mutator alleles (*POL2/pol2-x::URA3*). Each is also engineered to be hemizygous for both *MSH2* (MSH2/*msh2Δ::LEU2)* and *CAN1* (*CAN1*::natMX */can1Δ*::*HIS3*). Hemizygous deletion of *DUN1* in these strains was performed by amplification of pUG6 Kanmx vector using primers Dun1:kanmx F

(GAACCCTTTATAGTCGAGAGTAACAAGTAAAGGGGCTTAACATACAGTAAAAAAGG CAATACATGGAGGCCCAGAATACCCT) and Dun1:kanmx R (AGAGGCAAGATAATTCTGAGTATGTTTTGGGTATTTTATTGTCAGTAATTTGTAAAC GCTCAGTATAGCGACCAGCATTCAC) using Phusion polymerase (New England Biolabs) and the following PCR steps: 98 °C for 1 minute; repeat 30x: 98 °C for 10 seconds, 55 °C for 30 seconds, 72 °C for 1:30. Yeast cells of the target strain were then transformed with the PCR product using standard transformation protocols of yeast previously described (Williams *et al.* 2013). The resultant transformants were screened by plating cells on YPD containing G418 and individual transformant colonies were picked as strain founders.

Mutation Rate Assay

Estimates of the mutation rates by fluctuation analysis were calculated as previously described (Herr *et al.* 2014; Lee *et al.* 2019). Briefly, whole colonies of either haploids with a copy of *CAN1* or diploids hemizygous for both *CAN1* and *natMX*, were suspended in 100 µl of sterile H_2O. Suspended colony solution was then serially diluted and 10 µl was plated onto both synthetic complete (SC) media and SC-MSG media containing canavanine and lacking arginine. An additional 10ul of suspended cells were plated onto SC-URA, SC-LEU, and SC-HIS for genotyping. Plates were incubated for two days at 30 °C. CAN^r mutant colonies were counted and the data from colonies with matching genotypes were combined for fluctuation analysis. Mutation rates were calculated in R using the using newton.LD.plating function in the rSalvador package, which utilizes the MSS-maximum Likelihood Method. This returned *m,* the average number of mutational events in a colony. Can^r/Cell Division was then calculated by dividing *m* by Nt. Significant differences were identified using a log-likelihood ratio test (LRT.LD.plating) in rSalvador, which calculates a test statistic (λ) and p-value.

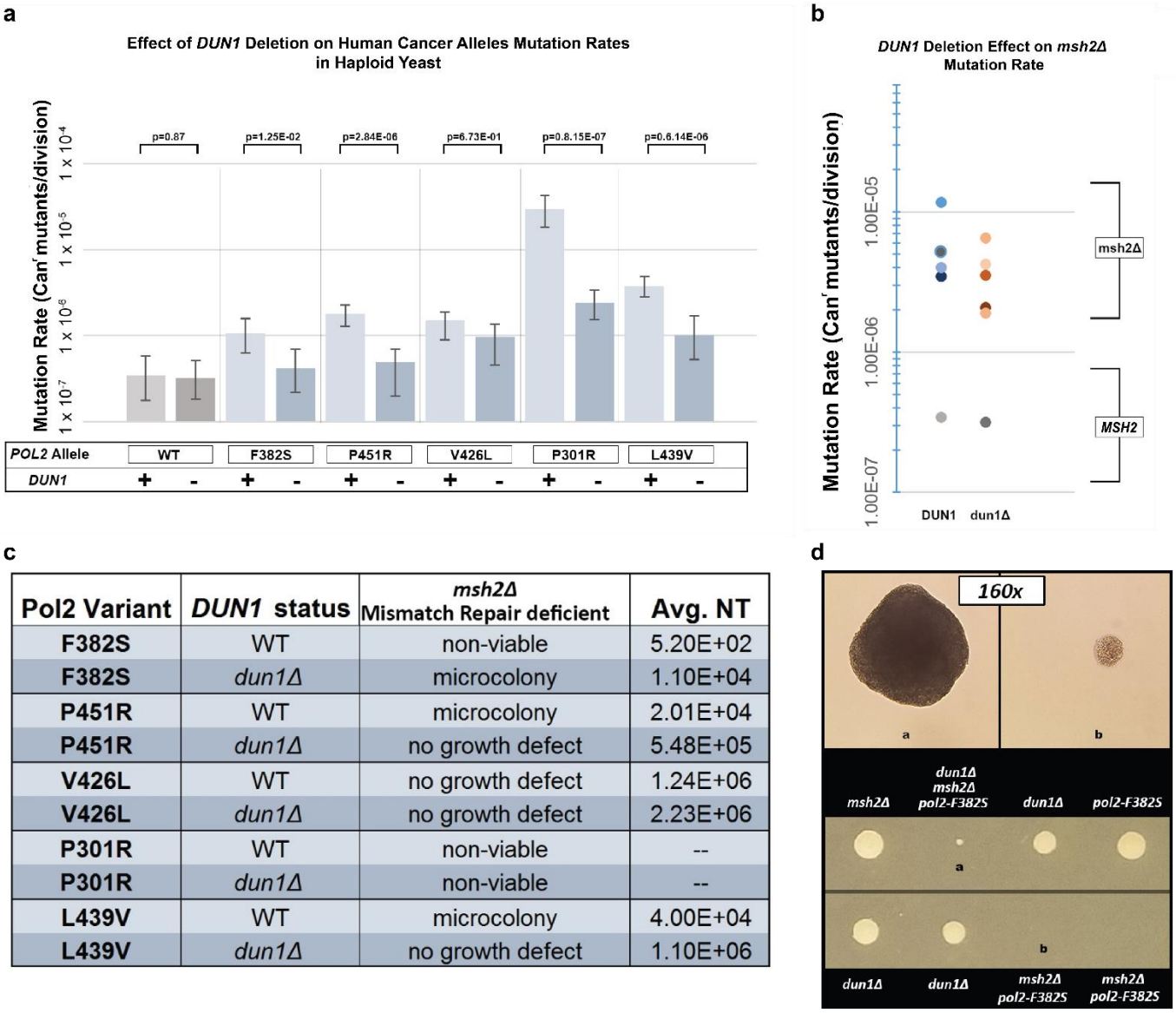
Genome Sequencing

Whole genome sequencing of constructed POL2/pol2-P301R *msh2Δ/ msh2Δ DUN1/ DUN1 and* POL2/pol2-P301R *msh2Δ/ msh2Δ dun1Δ/ dun1Δ* lineages was performed in much the same manner as the sequencing approach outlined in the Methods section of Chapter 2 of this dissertation. Our method of DNA fragmentation was the primary departure from the previously described method. In lieu of sonication, we used an enzymatic approach using the NEBNext® Ultra™ II FS DNA Library Prep Kit for Illumina (E7805L) kit to fragment DNA into appropriately sized fragments and ligate adapters. Samples were prepared according to kit instructions. Whole genome sequencing was performed primarily on the Illumina NovaSeq™ 6000 sequencing platform. Read data analysis again proceeded nearly identically to that described in chapter two with the exception that the previously described pipeline was adapted to a

new Snakemake approach (Köster and Rahmann 2012), however the tools and steps contained therein remain unaffected. Variant calling with Varscan2 we again used a minimum read-depth of 18 for both strains and a variant frequency cut-off of 0.22. Lineage analysis proceeded using our custom Python script, JLSLineageCaller, to determine the total number of shared variants within each lineage. These results were analyzed and tallied in Microsoft Excel as described in chapter two.

Mutation spectra of *pol2-P301R/ pol2-P301R msh2Δ/msh2Δ DUN1/DUN1* and *pol2-P301R/ pol2-P301R msh2Δ/msh2Δ dun1Δ/dun1Δ* diploid mother lineages were generated with toyplot 0.19.0 using the Nucleic Python module (https://github.com/clintval/nucleic).

Figures



Pol2 Variant	*DUN1* status	*msh2Δ* Mismatch Repair deficient	Avg. NT
F382S	WT	non-viable	5.20E+02
F382S	*dun1Δ*	microcolony	1.10E+04
P451R	WT	microcolony	2.01E+04
P451R	*dun1Δ*	no growth defect	5.48E+05
V426L	WT	no growth defect	1.24E+06
V426L	*dun1Δ*	no growth defect	2.23E+06
P301R	WT	non-viable	--
P301R	*dun1Δ*	non-viable	--
L439V	WT	microcolony	4.00E+04
L439V	*dun1Δ*	no growth defect	1.10E+06

Fig. 3.1 Mutation rates of haploid *POL2* candidate mutators and the effect of *DUN1* deletion

a, *POL2* mutations confer a wide distribution of elevated mutation rates which is significantly lowered in all but *pol2-V426L* in the absence of Dun1. **b,** Unlike Pol2 mutators, MMR-deficient msh2 mutators do not show a consistent significant mutation rate reduction in the absence of Dun1. **c,** Putative mutator alleles frequently synergize with *MSH2* deletion resulting in micro- or null-colony formation. *DUN1* deletion rescues or partially rescues *msh2Δ pol2* mutants from micro- or null phenotypes. Column NT represents estimated colony size in terms of number of cells at timepoint. **d,** in one such case, msh2*Δ /pol2-F382S*, *dun1Δ* enables the emergence of visible colonies.

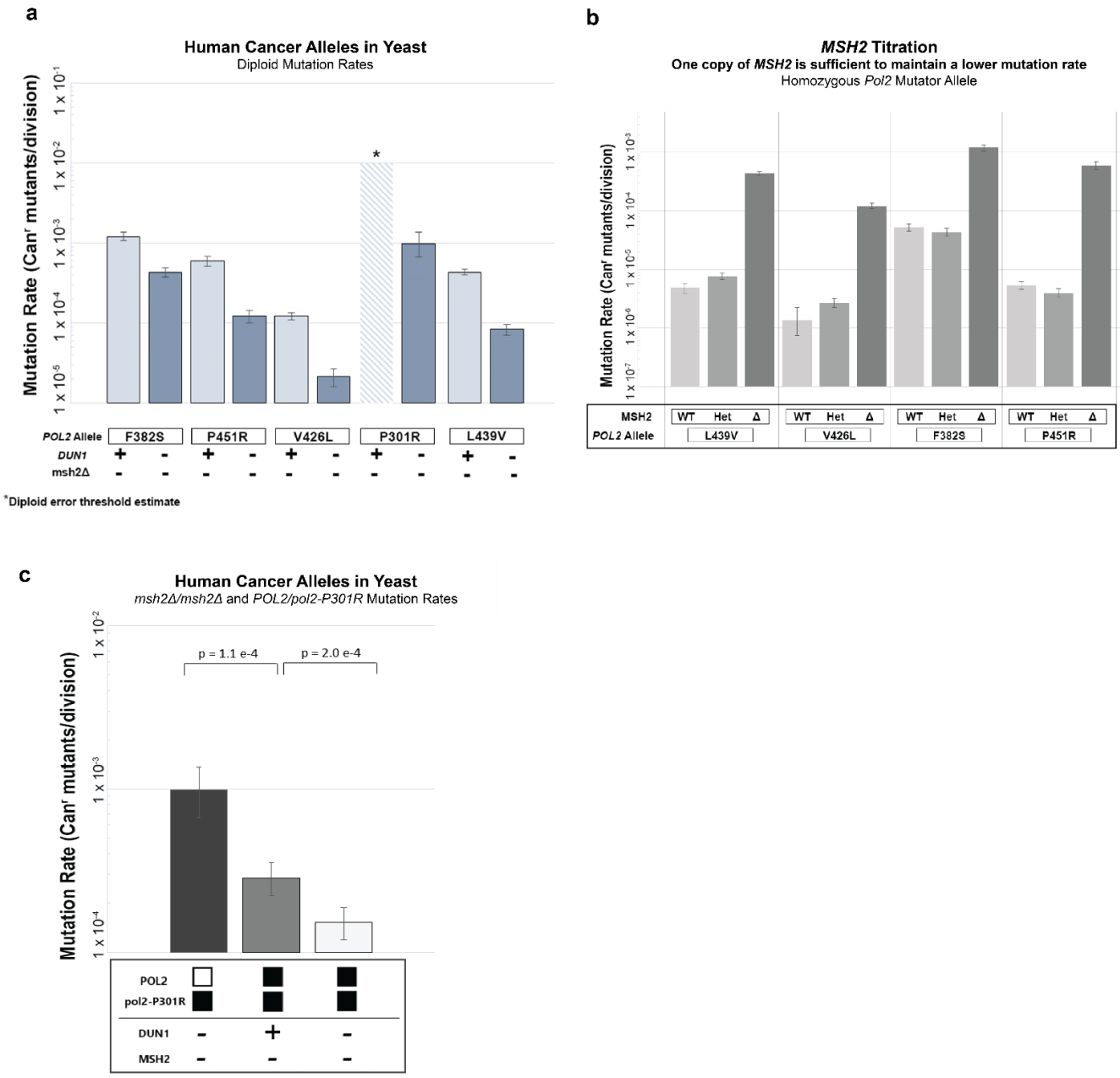


Fig. 3.2 Mutation rates of diploid *POL2* candidate mutators and the effects of *MSH2* and *DUN1* deletion

a, Diploid *POL2* mutator alleles exhibit a mutator phenotype partially rescued by *DUN1* deletion. **b**, mutator alleles capable of forming colonies in the absence of MSH2 were found to not be significantly affected by deleting a single copy of the *MSH2* gene. **c**, Heterozygous *pol2-P301R* confers a partial mutator phenotype when paired with a wildtype copy that can be further reduced by deletion of *DUN1*.

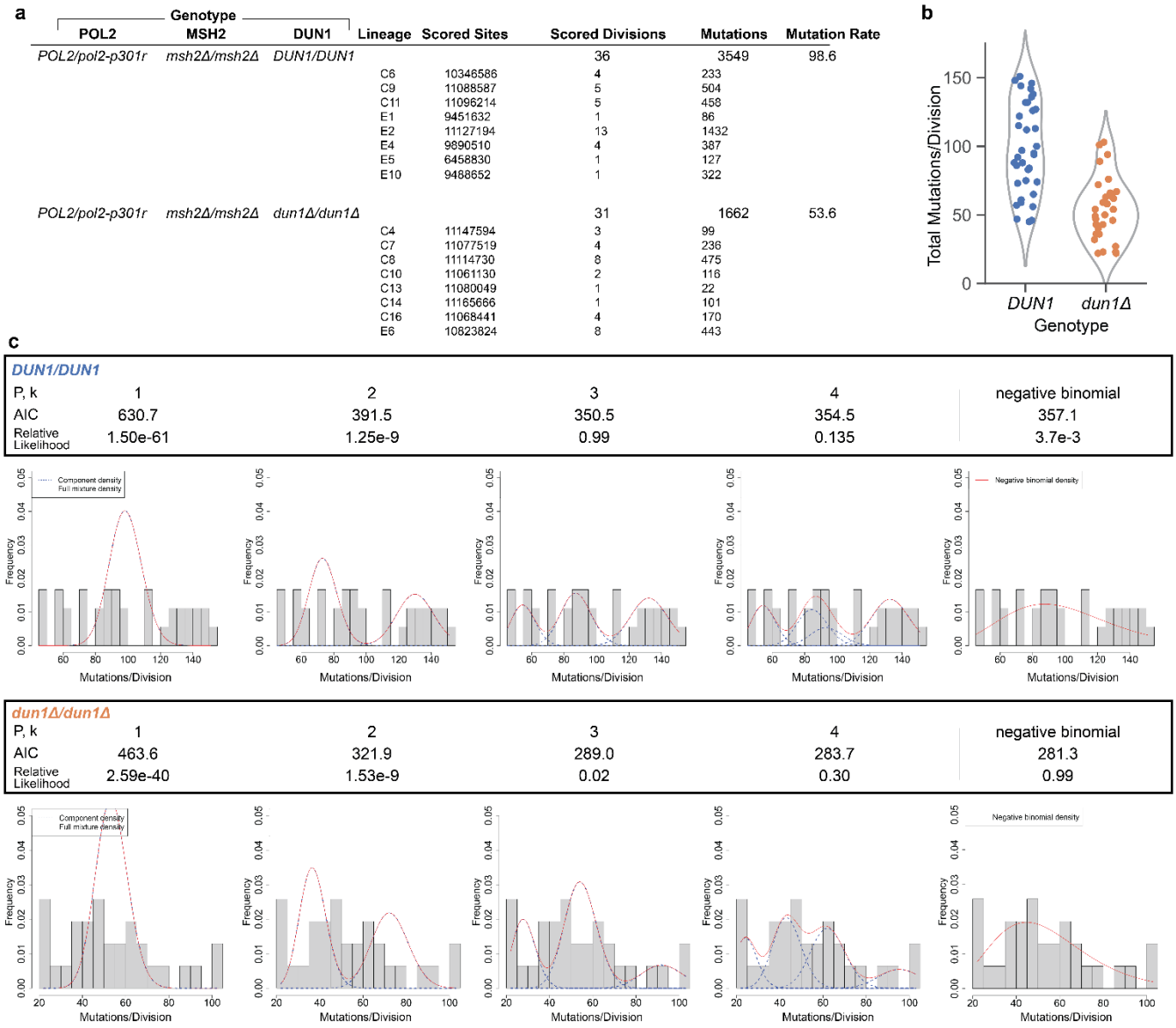
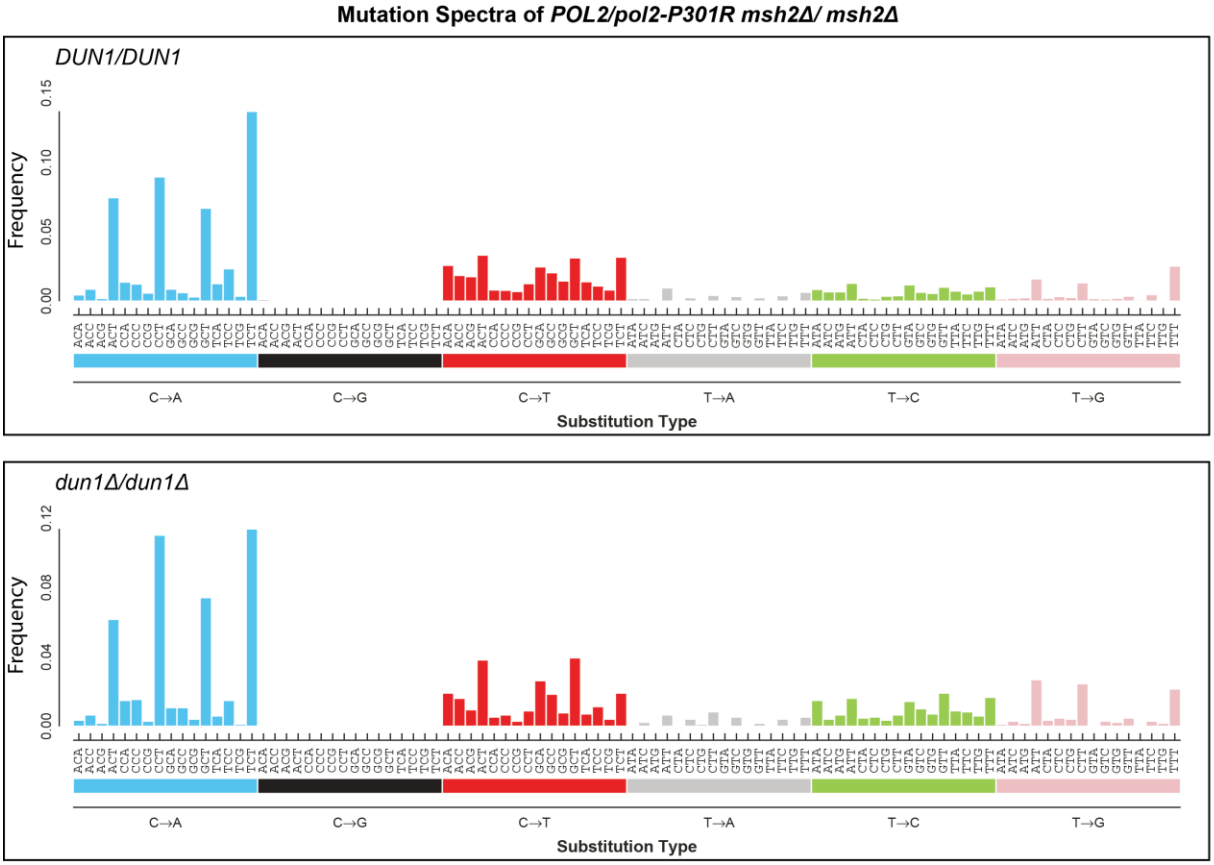


Fig. 3.3 The effect of *DUN1* deletion on mutator volatility at single cell resolution

a, Tabulation of total scored mutations per sub lineage and separated by genotype. Mutation rate expressed in Mutations/Division. **b**, Violin and scatter plot of mutations scored per division by *DUN1* genotype. **c**, Fitting the distributions of full error counts from diploid *POL2/pol2-P301R msh2Δ/msh2Δ* divisions, with and without *DUN1,* to different models. P, k indicates number of Poisson components, (e.g. k = 1, single Poisson; k = 2, two-Poisson, etc); nb, negative binomial; AIC, Akaike information criterion. Comparison of weighted AIC values expressed as relative likelihood.

Supplemental Figures

Supplemental. Fig. 3.1



Supplemental. Fig. 3.1 Mutation spectra of *POL2/pol2-p301r msh2Δ/msh2Δ* lineages with and without *DUN1*

The 96-trinucleoide mutation spectra context of all mutations scored by frequency from 36 divisions of *pol2-P301R/ pol2-P301R msh2Δ/msh2Δ DUN1/DUN1* diploid mother cells (top), and 31 divisions of *pol2-P301R/ pol2-P301R msh2Δ/msh2Δ dun1Δ/dun1Δ* cells (bottom), generated using the Nucleic Python module (https://github.com/clintval/nucleic).

Chapter Three References

Dowsett I. T., J. L. Sneeden, B. J. Olson, J. McKay-Fleisch, E. McAuley, *et al.*, 2021 Rate volatility and asymmetric segregation diversify mutation burden in cells with mutator alleles. Communications Biology 4: 21. https://doi.org/10.1038/s42003-020-01544-6

Elledge S. J., and R. W. Davis, 1990 Two genes differentially regulated in the cell cycle and by DNA-damaging agents encode alternative regulatory subunits of ribonucleotide reductase. Gene Dev 4: 740--751. https://doi.org/10.1101/gad.4.5.740

Fox E. J., M. J. Prindle, and L. A. Loeb, 2013 Do mutator mutations fuel tumorigenesis? Cancer Metast Rev 32: 353--361. https://doi.org/10.1007/s10555-013-9426-8

Giraud A., I. Matic, O. Tenaillon, A. Clara, M. Radman, *et al.*, 2001 Costs and benefits of high mutation rates: adaptive evolution of bacteria in the mouse gut. Science 291: 2606--2608. https://doi.org/10.1126/science.1056421

Herr A. J., M. Ogawa, N. A. Lawrence, L. N. Williams, J. M. Eggington, *et al.*, 2011 Mutator Suppression and Escape from Replication Error–Induced Extinction in Yeast. Plos Genet 7: e1002282. https://doi.org/10.1371/journal.pgen.1002282

Herr A. J., S. R. Kennedy, G. M. Knowels, E. M. Schultz, and B. D. Preston, 2014 DNA Replication Error-Induced Extinction of Diploid Yeast. Genetics 196: 677–691. https://doi.org/10.1534/genetics.113.160960

Kane D. P., and P. V. Shcherbakova, 2014 A common cancer-associated DNA polymerase ε mutation causes an exceptionally strong mutator phenotype, indicating fidelity defects distinct from loss of proofreading. Cancer Res 74: 1895--1901. https://doi.org/10.1158/0008-5472.can-13-2892

Kennedy S. R., E. M. Schultz, T. M. Chappell, B. Kohrn, G. M. Knowels, *et al.*, 2015 Volatility of Mutator Phenotypes at Single Cell Resolution. Plos Genet 11: e1005151. https://doi.org/10.1371/journal.pgen.1005151

Köster J., and S. Rahmann, 2012 Snakemake—a scalable bioinformatics workflow engine. Bioinformatics 28: 2520–2522. https://doi.org/10.1093/bioinformatics/bts480

Lee Y. D., J. Wang, J. Stubbe, and S. J. Elledge, 2008 Dif1 Is a DNA-Damage-Regulated Facilitator of Nuclear Import for Ribonucleotide Reductase. Mol Cell 32: 70--80. https://doi.org/10.1016/j.molcel.2008.08.018

Lee M. B., I. T. Dowsett, D. T. Carr, B. M. Wasko, S. G. Stanton, *et al.*, 2019 Defining the impact of mutation accumulation on replicative lifespan in yeast using cancer-

associated mutator phenotypes. Proc National Acad Sci 201815966. https://doi.org/10.1073/pnas.1815966116

Loeb L. A., 1991 Mutator phenotype may be required for multistage carcinogenesis. Cancer Res 51: 3075--3079.

Parkash V., Y. Kulkarni, J. ter Beek, P. V. Shcherbakova, S. C. L. Kamerlin, *et al.*, 2019 Structural consequence of the most frequently recurring cancer-associated substitution in DNA polymerase ε. Nat Commun 10: 373. https://doi.org/10.1038/s41467-018-08114-9

Peltomaki P., L. Aaltonen, P. Sistonen, L. Pylkkanen, J. Mecklin, *et al.*, 1993 Genetic mapping of a locus predisposing to human colorectal cancer. Science 260: 810--812. https://doi.org/10.1126/science.8484120

Pursell Z. F., I. Isoz, E.-B. Lundström, E. Johansson, and T. A. Kunkel, 2007 Yeast DNA Polymerase ε Participates in Leading-Strand DNA Replication. Science 317: 127–130. https://doi.org/10.1126/science.1144067

Robinson J. T., H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, *et al.*, 2011 Integrative genomics viewer. Nat Biotechnol 29: 24–26. https://doi.org/10.1038/nbt.1754

Shinbrot E., E. E. Henninger, N. Weinhold, K. R. Covington, A. Y. Göksenin, *et al.*, 2014 Exonuclease mutations in DNA polymerase epsilon reveal replication strand specific mutation patterns and human origins of replication. Genome Res 24: 1740--1750. https://doi.org/10.1101/gr.174789.114

Shlien A., B. B. Campbell, R. de Borja, Alex, L. B. rov, *et al.*, 2015 Combined hereditary and somatic mutations of replication error repair genes result in rapid onset of ultra-hypermutated cancers. Nat Genet 47: 257--262. https://doi.org/10.1038/ng.3202

Tracy M. A., M. B. Lee, B. L. Hearn, I. T. Dowsett, L. C. Thurber, *et al.*, 2020 Spontaneous Polyploids and Antimutators Compete During the Evolution of Saccharomyces cerevisiae Mutator Cells. Genetics 215: 959–974. https://doi.org/10.1534/genetics.120.303333

Williams L. N., A. J. Herr, and B. D. Preston, 2013 Emergence of DNA Polymerase ε Antimutators That Escape Error-Induced Extinction in Yeast. Genetics 193: 751–770. https://doi.org/10.1534/genetics.112.146910

Williams L. N., L. Marjavaara, G. M. Knowels, E. M. Schultz, E. J. Fox, *et al.*, 2015 dNTP pool levels modulate mutator phenotypes of error-prone DNA polymerase ε variants. Proc National Acad Sci 112: E2457–E2466. https://doi.org/10.1073/pnas.1422948112

Xing X., D. P. Kane, C. R. Bulock, E. A. Moore, S. Sharma, *et al.*, 2019 A recurrent cancer-associated substitution in DNA polymerase ε produces a hyperactive enzyme. Nat Commun 10: 374. https://doi.org/10.1038/s41467-018-08145-2

Chapter Four
Genome-wide Single Cell Resolution Lineage Analysis Reveals that Replicative Aging Does Not Substantially Increase Loss of Heterozygosity in Yeast Cells

Ian T. Dowsett and Alan J. Herr

Abstract

Loss of heterozygosity, a rare consequence of DNA damage repair through homologous recombination, and in particular break-induced replication, has the potential to catalyze a host of deleterious health effects including the development and evolution of cancer. LOH events may also be a contributing factor to aging. Previous research investigating the rates of recombination present in aged *Saccharomyces cerevisiae* cells at a limited number of loci within the genome found evidence for increased break-induced replication as a function of aging. Later studies suggested that this increase may have been strain-specific. Here, we used single-cell lineage analysis of hybridized wild yeast strains, combined with whole genome sequencing, to explore whether the number and breadth of LOH across the entire yeast genome increases as a function of age for individual yeast cells. We found evidence that such events occur with similar frequencies in both aged and young wildtype cells.

Introduction

Over a lifetime, somatic cells are subjected to an accumulation of mutations as a consequence of DNA damage, both endogenous and exogenous. The somatic mutation accumulation theory of aging posits that the continuous degradation of genomic information propels the relentless decline of tissue function and stem cell attrition (Morley 1998; Kirkwood 2005). The mechanism of how somatic mutation accumulation would drive aging is unclear. In general, mutations that accumulate will be heterozygous and many will be recessive, or synonymous, with a potentially minimal impact on overall cell function. Dominant deleterious mutations could, of course, impact organism health but whether these accumulate in sufficient numbers is unknown. Perhaps more likely, epistatic interactions between pairs of mutations, which increase at roughly the square of the mutation burden, may drive loss of function. A further possibility, and the focus of this current study, is the role that mitotic recombination may play in causing heterozygous mutations to become homozygous. This loss of heterozygosity (LOH) may arise as a consequence of a number of unresolved DNA repair processes, most notably the repair of double strand breaks (DSBs) through homology directed repair (HDR) using a homologous chromosome, rather than a sister chromatid, as a repair template. LOH events result in a net loss of potentially critical genetic information in the form of gene conversion (GC). An extreme form of this phenomenon, Break Induced Replication (BIR), can result in loss of the unique information of an entire chromosome

arm, affecting a significant portion of the genome (Kramara *et al.* 2018). LOH occurs frequently in cancer biology and represents a key method by which tumor cells deactivate important tumor suppressors or activate oncogenes, catalyzing further tumor development (Cavenee *et al.* 1983; Huang *et al.* 1992). Through much the same mechanism, LOH may drive the emergence of null phenotypes that promote aging. This process may be exacerbated by the compounding failures of DNA repair processes coupled with the constant accumulation of unrepaired DNA damage associated with age (Rossi *et al.* 2007). An important unanswered question in the aging field is whether LOH occurs in aged cells at rates high enough to be relevant to the aging phenotype.

Several methods have been used previously to investigate loss of heterozygosity, in the context of aged budding yeast cells. A classic approach used a colony sectoring assay to detect loss of heterozygous alleles of *ADE2/MET15* at a single loci resulting in either red, black, or both red and black colony sectors indicative of reciprocal or non-reciprocal LOH events (Lindstrom *et al.* 2011). This tactic, though useful for high throughput screening, is only able to screen a single locus at a time. LOH events at select loci were found to increase in aged mothers while remaining relatively low at others. An approach to analyze whole genomes for loss of heterozygosity outside of an aged context combines using hybrid yeast strains with a SNP microarray. Several studies using this technique have been able to identify hotspots for recombination in various mutant backgrounds (Andersen *et al.* 2016), (O'Connell *et al.* 2015). Neither technique, however, allows for genome-wide screening of LOH events in the context of a single aging mother cell.

To remedy these limitations we took advantage of the natural genetic diversity of wild yeast strains that have been engineered to be amenable to standard laboratory procedures(Liti *et al.* 2009; Louvel *et al.* 2014). As above, our approach relies on generating hybrid diploid cells with unique SNVs present across the entire length of homologous chromosomes. By whole genome sequencing, each SNV will be present at 50% frequency in the initial hybrid cells. LOH events can be readily identified from single or sequential SNVs increasing in frequency to 100%. All LOH events scored in our study relies on the concordance between colonies derived from pairs of cells separated by a single division. Thus, we can say with certainty that the LOH events took place prior to that division and could not have arisen during outgrowth of the colonies in culture. The theoretical resolution of this approach then may be defined as the size of the genome divided by the number of unique heterozygous SNVs. Coupling this approach with the isolation of single yeast lineages by micromanipulation and then whole genome sequencing allowed us to investigate whether aged mother cells at the very end of their life are prone to non-reciprocal recombination events or break-induced replication across the breadth of the entire yeast genome.

Results

Available Yeast Strain Diversity

To develop a genome-wide screen for LOH using wild yeast we needed a combination of strains with a large number of unique SNVs. We considered five wild isolate strains as potential progenitors for hybrid crosses (see Yeast Strains in Methods). Each of the wild strains is a representative of one of five largely distinct genetic groups of *Saccharomyces cerevisiae*; Wine/European, North American, West African, Sake, and Malaysian (Liti *et al.* 2009). The strains we used in this study have been engineered with a number of markerless auxotroph deletions along with the critical deletion of the *HO* gene to inhibit homothallic switching and enable strain specific hybrid matings (Louvel *et al.* 2014). Only SNVs unique to one of the two parent strains are useful for our approach as these SNVs must be heterozygous in the resulting hybrid offspring. Consequently, SNVs shared between the two strains but unique from the reference genome do not provide additional useable information in this context. The majority of SNVs present in the wild isolate strains are homozygous due to engineering backcrossing to homogenize the genomes providing high confidence that each SNV will be present in hybrid progeny (Louvel *et al.* 2014). Whole genome sequencing of the five potential wild strains confirmed their identities and provided the means to choose hybrids that maximized the number of unique SNVs (Table 4.1). The unique characteristics of the wild strains also influenced our choice of hybrid cross. We found several strains difficult to transform or to micromanipulate during or after tetrad dissections due to a tendency to flocculate. West African derived strain yML308 (DBVPG6044 parent) and the European/wine derived strain yML306 (DBVPG6765) also exhibited low sporulation efficiency (see Methods). Sake x Malaysian hybrids emerged as the best combination due to a high number of potential heterozygous SNVs (~59,300) coupled with efficient sporulation, enabling swift generation of a number of hybrid crosses by micromanipulation. Mapping of all potential SNVs from this cross onto the yeast genome revealed an even distribution of SNVs across the genome, providing broad coverage for identifying LOH events (Fig. 4.1).

SGS1 mutant hybrids

We sought to generate mutant Sake x Malaysian hybrid strains with increased rates of DNA recombination in order to first validate the use of hybrid strains to map age-dependent recombination across the genome. Sgs1, the only RecQ helicase found in *S. cerevisiae*, plays an important role in the maintenance of genome integrity. *sgs1* mutants have long been found to exhibit increased rates of both inter- and intra-chromosomal recombination(Watt *et al.* 1996; McVey *et al.* 2001). Construction of homozygous *sgs1Δ/ sgs1Δ* mutants by crossing *sgs1Δ* spores of engineered *SGS1/sgs1Δ* Sake and Malaysian strains failed to produce mother cells capable of forming ‘complete’ lineages, those amenable to analysis with a macroscopic initial and end-state colony, in all but a single case. Hemizygous *SGS1/sgs1Δ* hybrid mothers, though noticeably short lived, survived long enough to provide colonies capable of

sequencing for lineage analysis. Previous studies have noted the extreme lethality of *SGS1* mutants (McVey *et al.* 2001), these effects may be more pronounced in our hybrids due to interacting genetic variants present in the parental wild strains.

Several recombination events could be readily identified in Sake x Malaysian hybrid, *SGS1/sgs1Δ* lineages (Fig. 4.2, Supplementary Table 4.1). Most notably, two lineages, B6 and B2, experienced significant loss of heterozygosity on the right arm of chromosome XII, immediately downstream of the rDNA locus, a previously identified recombination hotspot in *Saccharomyces cerevisiae* (Christman *et al.* 1988).Two of these events extended from this region to the end of the chromosome arm, a genomic distance of over half a megabase. Lineages A3 and B7 demonstrate significant amounts of LOH on the distal regions of chromosomes XIII and XV. Analysis of affected SNVs in the region strongly suggests a single event for each converted tract (Supplemental Data Table 4.1). All SNVs that converted from 50 to 100 percent frequency originated from the same parental strain, while SNVs that dropped from 50 to 0% came from the other parental strain. Read depth analysis of lineage A2 with LOH detection events across the entire length of chromosome II, suggests a complete substitution of that chromosome in the form of uniparental disomy rather than aneuploidy. Interestingly, *sgs1* mutants have previously been shown to increase chromosome missegregation frequency and tend to favor non-disjunction events (Watt *et al.* 1995). Events in lineages B7 and A3 on chromosomes XV and XIII, respectively, both appear to extend to the ends of their chromosome arms as no SNVs within this region remained heterozygous. These observations imply these events may be a result of BIR. The ability to detect LOH tracts in this system indicate that this approach to identifying LOH in yeast lineages is viable and enables detailed whole genome analysis of heterozygosity.

Wildtype Wild Hybrids

In order to determine if recombination increased dramatically in normal aged cells we next moved to characterizing wild *SGS1*-proficient hybrids. To determine if the frequency of LOH truly increases as a function of age, we isolated two parallel lineages for each mother cell: an aging lineage and a ‘Forever Young’ (FY) lineage. The aging lineage was isolated much as our *SGS1/sgs1Δ* hemizygous lines by dissecting daughter cells away from mothers throughout her replicative lifespan. Daughter cells were retained until two additional replicatively active daughters were recovered, whereupon they were discarded. The final daughter capable of dividing and her granddaughter were moved to defined locations on the plate to form colonies. We isolated a parallel FY lineage for each mother cell as a means of distinguishing between the effects of aging and the effect of a number of age-independent divisions. This line was derived from these same mother cells, but rather than retaining an aging mother cell, for this line we instead retained only the daughter cells from each division (Fig. 4.3A). The final pair from this line was allowed to form a colony when the number of divisions matched the final replicative age of the mother cell. Ideally, this procedure

produces two pairs of colonies that have each undergone the same number of divisions, one pair that has been generated by an aged mother cell and one that is entirely absent an aged context from the same lineage.

Obtaining an equivalent number of divisions in the FY lineage failed in six out of eleven analyzed lineages due to the unexpected finding that newborn daughters occasionally fail to divide. Though we cannot discard the possibility that the failure of newborn daughters is due to the micromanipulation used to remove the mother cell, this seems unlikely as mother cells subjected to the same treatment on the same plate do not experience this failure. Linear regression of this occurrence reveals a nearly 1% chance of newborn cell sterility (Supplementary Fig. 4.1, m = -0.0096, R^2 = 0.94). In cases of this early replicative failure in the Forever Young lineage, the last two mothers from the Forever Young lineage that could produce progeny were retained as the end-state colonies for that lineage.

Among the set of 12 scored Forever Young lineages a total of 4 tracts of LOH events could be found (Fig. 4.3c, Supplementary Table 4.2). Tract length size can be estimated as a possible range. The minimum tract length includes only the converted SNV(s) and the distance between, barring any heterozygous unconverted SNVs between them. The maximum possible range extends to the closest flanking unconverted heterozygous SNVs. These tracts averaged between a minimum estimated tract length of 1-2272 basepairs and a maximum possible tract length ranging from 115-11950 basepairs. The 12 Aged lineages produced a much narrower range of 1-2 to a maximum 26-101 possible tract length among 4 LOH events.

Discussion

Many previous investigations into LOH as a consequence of aging have been limited to isolated regions of the genome since they relied on visible or selectable marker genes. Further, rather than investigating the outer edges of life by querying the last daughters of a lineage, other studies have been limited by investigating a heterogenous population of enriched mother cells in the latter portions of their life (Lindstrom and Gottschling 2009). While these studies provide important insights into the aging process and enable high throughput analysis, such approaches do not provide the resolution to determine what happens at the critical threshold of age-associated reproductive failure. With this study we aimed to develop a method capable of tackling both of these challenges by analyzing the entire genome of the very last daughters aged mothers produce. Unlike colony sectoring assays, in which a daughter or granddaughter of the colony progenitor may give rise to a smaller sector as a result of recombination at the target locus in a later division, our method instead relies on sequencing pairs of cells and only scoring those events detected in both sequenced colonies. We hoped that this approach would allow us to test whether mothers just prior to reproductive failure experience a sudden burst of DNA repair dysfunction characterized by large and frequent LOH tracts. Though the number of lineages analyzed is limited in this study, the low number of observed events in both the aged

and forever young lineages suggests that aging does not play a critical role in driving loss of heterozygosity in wildtype hybrid yeast cells.

In studies performed using single-locus colony sectoring analysis, the ability to detect LOH requires LOH tracts to overlap the alleles of interest. Because of this, the size of LOH tracts informs our understanding of the probability of a given LOH event across the genome to be observed. We found tract lengths to be greatly reduced in wild hybrids compared to hybrids containing only a single functional copy of *SGS1*. This finding is supported in the literature by the observation that Sgs1 regulates gene conversion tract length (Lo *et al.* 2006). The extensive tracts observed in our *sgs1Δ/sgs1Δ* lineage B7 and in *SGS1/sgs1Δ* lineage A3 are likely the product of BIR and the lack of any such observations in wildtype lineages provides further support to previous findings in the literature that BIR is more common for cells that lack SGS1 rather than more conservative and local gene conversion (Lydeard *et al.* 2010). The fact that such large tracts occur even in lineages hemizygous for *SGS1* suggests that two functional copies of *SGS1* are necessary to limit the propagation of extensive LOH events. Surprisingly, despite extremely long tract lengths and a reported high frequency of recombination events in *sgs1* mutants in the literature (Myung *et al.* 2001), we found the overall frequency of events to be only marginally higher to those observed in wildtype lineages at 0.027 LOH tracts/division for aged *SGS1* hemizygotes and 0.0121 LOH tracts/division for wildtype cells. This may be due to the majority of analyzed lineages possessing a single functional copy of *SGS1*. Sgs1 plays multiple roles in resolving DSBs and it is possible large LOH tract lengths are vulnerable to the effects of haploinsufficiency while a sufficient quantity of Sgs1 remains to limit the number of events. It may also be that observations limited to just a single locus at a time are unable to distinguish the difference between many tracts and few tracts of far greater extent. We note that the lone viable *sgs1Δ/ sgs1Δ* lineage we analyzed experienced a single LOH event over nine full divisions.

In comparison to observed values of LOH in aged yeast, reported at a rate of 200-300 per 10,000 cell division (0.02-0.03) per division at a single locus in aged cells (McMurray and Gottschling 2003), we noted a remarkably similar number of events in our small dataset across the complete genome. We scored 0.0121 LOH events per cellular division in our aged lineages and a similar rate of 0.0162 LOH events per division in forever young lineages. Comparing our results to single-locus rates are complicated by the uncertain size of reported LOH domains in the latter type of experiment. Of those we found, most include a single converted SNV, equivalent to up to several hundred basepairs of converted genome. Such small tracts occurring as infrequently as they do in our dataset would make chance observations outside of known hotspots incredibly unlikely when using single or two loci reporter approaches.

It remains likely that small increases in particularly vulnerable loci, especially those outside the rDNA locus, do slightly increase as a function of age based on other's findings at this particular region (Lindstrom *et al.* 2011), but such an increase is below our detection threshold for our small cohort of single cells. Based on our findings that rates of loss of heterozygosity do not seem to accrue at greater values in wildtype aged

mothers and that such events when they do occur encompass only a small (1-2kb) region of the genome, when they do occur in healthy cells it seems unlikely that LOH is capable of significantly driving the age-associated decline of otherwise healthy cells. Our findings also offer additional evidence that loss of only a single copy of critical genome maintenance machinery, such as the RecQ helicase SGS1 may drive, over several cellular generations, loss of vast tracts of unique genetic information. The size and infrequency of LOH tracts in a given aged wildtype lineage makes this experimental system amenable to testing the effects of other changes to the genome that might influence LOH tract length. The system also allows for analysis of multiple colinear mutations or other alterations to a single copy of a given gene in a diploid cell.

Methods

Microbial Strains and Media

All bacterial cloning experiments utilized the *Escherichia coli* strain DH10B. Bacteria were grown with NZYM (MP Biomedicals) media supplemented with chloramphenicol (25 µg/ml), carbenicillin (100 µg/ml), or kanamycin (50 µg/ml) depending on the plasmid being transformed. For standard growth, yeast were propagated on synthetic complete (SC) solid media (6.7 g/L Difco yeast nitrogen base without amino acids, 2% wt/vol dextrose, 2 g/L SC amino acid mix (Bufferad), and 2% agar) or liquid YPD media (1% wt/vol Bacto yeast extract, 2% wt/vol Bacto peptone, and 2% wt/vol dextrose). To select for yeast transformations utilizing LEU2 DNA, we used solid synthetic complete media lacking leucine (SC-LEU), which is identical to SC except the amino acid mix lacks leucine (Kaiser *et al.* 1994). To sporulate yeast, we routinely used liquid sporulation media (1% potassium acetate, 0.1% yeast extract, 0.05% dextrose).

The wild yeast diploid strains used in this study carry gene deletions found in the common laboratory strain that stabilize mating type and facilitate genetic engineering (*ho::HphMX/ho::HphMX ura3Δ0/ura3Δ0 leu2Δ0/leu2Δ0 LYS2/lys2Δ0 MET15/met15Δ0*)(Louvel *et al.* 2014). The “Sake” (NCYC3920) and “Malaysian” (NCYC3930) strains are derived from SGRP diploid parent strains Y12 and UWOPS03-461.4. The “West African” (NCYC3900), “European/Wine” (NCYC3890) and “North American” (NCYC3910) strains are derived from strains DBVPG 6044, DBVPG 6765, YPS128. The common diploid laboratory strain, BY4743, is derived from S288C (Brachmann *et al.* 1998; Louvel *et al.* 2014).

The *sgs1*Δ::LEU2 mutant wild strains were engineered using the Modular Cloning (MoClo) Yeast Toolkit (YTK) built upon Golden Gate Cloning technology (Lee *et al.* 2015). All oligonucleotides used in these experiments can be found in Table S1. All Golden Gate assembly reactions utilized 20 fm of vector backbone and 40 fm of each set of annealed oligonucleotides, donor PCR fragment, or plasmid in a 20 ul reaction in 1x T4 DNA ligase buffer catalyzed by the following enzymes (New England Biolabs): 1 µl of BsmBI (10 units) or BsaI (20 units) and 1 µl of T4 DNA ligase (400 units) or T7 DNA ligase (3000 units). All assembly reactions were recovered by transformation of *E. coli*. The dual CAS9/sgRNA expression vector targeting SGS1 was constructed in a

series of steps. First, we constructed a plasmid (pAJH003) with a CAS9 gene expression cassette, regulated by the PGK1 promoter and terminator, by performing a BsaI assembly reaction with pYTK002, pYTK0011, pYTK036, pYTK054, pYTK067, and pYTK095 (37°C for 60 min; 25 cycles of 37°C for 5 min and 25°C for 10 min; 37°C for 30 min; 65°C for 20 min). We constructed a URA3-CEN6/ARS4 backbone (pAJH004) for the dual expression vector by performing a BsaI assembly reaction with pYTK008, pYTK048, pYTK073, pYTK074, pYTK082, and pYTK084. The standard YTK entry vector, pYTK001, carries GFP in the cloning site to facilitate identification of clones with positive assemblies. To create a GFP-minus entry vector (pAJH044) suitable for cloning fragments containing GFP, we performed a BsmBI assembly reaction with pYTK_2_GFP0f and pYTK_2_GFP0r oligonucleotides, which anneal to create 4 nucleotide overhangs suitable for cloning into pYTK001. Since this oligonucleotide insert contains two internal BsmBI sites, our cycling parameters finished on a long ligation step: 42°C for 60 min; 25 cycles of 42°C for 5 min and 25°C for 10 min; 25°C for 20 min. We used pAJH044 to make a plasmid (pAJH001) with an sgRNA expression cassette, flanked by DNA fragments from the mouse Adh1 gene. In the present study we used a direct cloning strategy to introduce the guide RNA sequences into the dual expression vector. In other studies, we have used these “homology arms” to facilitate transferring sgRNA cassettes into the dual vector by gap repair in yeast – a strategy outlined first by Shaw et al (Shaw *et al.* 2019). The Adh1 left homology arm was amplified from mouse genomic DNA with sgRNALHf and sgRNALH_GGr; the right homology arm was amplified with sgRNARH_GGf and sgRNARHr. The sgRNA GFP^+ cloning cassette was amplified from pYTK050 with sgRNA234_GGf and sgRNA234_GGr. Each fragment is flanked by BsmBI sites that facilitate directed assembly into pAJH044. The sgRNA cassette from pYTK050 also contains internal BsmBI sites, flanking GFP, that must be retained in the final vector. Thus, we again used cycling parameters that ended on a long ligation step, as described for pAJH044. We moved the sgRNA expression cassette from pAJH001 into an intermediate vector (pAJH089) using a BsaI assembly reaction involving pAJH001, pYTK003, pYTK072, pYTK076, pYTK081, and pYTK083. To make the CAS9/sgRNA dual expression vector (pAJH098), we performed a BsmBI assembly reaction with pAJH089, pAJH003, and pAJH004. The pAJH003, pAJH004, and pAJH089 assembly reactions included linker sequences that reintroduced BsmBI sites flanking the parts to be included in the final assembly. Since the sgRNA expression cassette also includes internal BsmBI sites flanking GFP, we used the cycling parameters described for pAJH044. We directly cloned the SGS1 sgRNA sequences into pAJH098 (pAJH098_sgs1) by performing a BsmBI assembly with annealed SgRNA_SGS1f and SgRNA_SGS1r and cycling parameters of 42°C for 60 min; 25 cycles of 42°C for 5 min and 25°C for 5 min; 55°C for 10 min; 65°C for 20 min. We created an *sgs1Δ::LEU2* repair template (pID.LEU2.SGS1), with a precise deletion of the Sgs1 coding sequence, by generating PCR fragments of the DNA upstream (with SGS1-5HOMf and SGS1-5HOMr) and downstream (SGS1-3HOMf and SGS13HOMr) of SGS1 and combining them with a

LEU2 PCR fragment (amplified with pRSfp4MoClo and pRSrp4MoClo from pRS415) in a BsmBI assembly reaction with pYTK001.

Sake and Malaysian yeast strains were transformed with 60 ng of linearized pAJH098_sgs1 and 10 ul of DNA, PCR amplified from pID.LEU2.SGS1 with PYTK001-2514f and CAMtermR, and plated onto solid SC-LEU media(Gietz and Schiestl 2007). Resultant colonies were then screened by PCR for correct insertion of the *sgs1Δ::LEU2* allele using a primers internal to *LEU2* (pRS415-1499F) and upstream of the 5' *SGS1* homology sequence used in the repair template (ID1-SGS1-F). two yeast isolates from each transformation were selected for further studies. The transformed *SGS1/sgs1Δ::LEU2* strains used in this study were the Sake derived ID0021 and ID0022 and Malaysian derived ID0034 and ID0036.

Sporulation of Diploid Cells

To induce sporulation, overnight cultures of cells were diluted 1:50 in fresh liquid YPD and grown at 30°C for 3-4 hours until a concentration of approximately $1x10^7$ cells per ml was reached (time to desired concentration varies per wild strain). Cells were pelleted by centrifugation and washed twice with sterile water. Cells were resuspended in 2mL of sporulation media and then grown at room temperature in a culture rotator for 4 - 8 days until sufficient sporulation had taken place.

Generation of Hybrid *SGS1/sgs1Δ* and *sgs1Δ/sgs1Δ* Cells

Hybrid *SGS1/sgs1Δ* and *sgs1Δ/sgs1Δ* mother cells were generated from freshly isolated zygote hybrids. To isolate hybrid zygotes, two sporulated Malaysian (ID0034 or ID0036) and Sake (ID0021 or ID0022) *SGS1/sgs1Δ* cultures were spotted onto SC-LEU. Ten tetrads of each strain were dissected into single spores in the same area on the plate and then incubated at 30°C until the subset of *sgs1Δ::LEU2* spores formed microcolonies of 8-16 cells. A single haploid cell from each microcolony was then placed near an opposing single cell from the microcolonies of the other strain. A fraction of these pairings resulted in a zygote that was then moved to an isolated and marked region of the plate. The first daughter of a zygote was then used as the founding mother cell for a given lineage.

Isolation of *SGS1* mutant Lineages

To obtain the *sgs1Δ::LEU2/sgs1Δ::LEU2* or *SGS1/sgs1Δ::LEU2* lineages, the founding mother cell was maintained at a fixed position on each plate. Plates were incubated at 30°C and then checked at 45-minute intervals for fully budded daughter cells. Upon separation from their respective mother cell, each daughter was moved to a defined location at least 5 mm from the previous daughter cell. To confirm that any LOH events were present at this division and did not arise later, the first progeny of each daughter cell was isolated and moved 5 mm below her progenitor. In this manner a grid of daughter and grand-daughter derived colonies was isolated, producing a complete lineage of the mother cell. The first and last daughter-granddaughter pair that formed healthy colonies, representing the bounds of the mother's replicative potential, were

subjected to whole genome sequencing to determine the number of LOH events that occurred during the mother's lifespan.

Isolation of Wildtype Lineages

Wildtype hybrid mother cells were derived in a similar manner as above. Sake and Malaysian strains spotted on a single plate were crossed as before to generate a zygote. This zygote was isolated and allowed to form a microcolony from which fifteen cells were selected. The first progeny of each of these cells was used as a founding mother cell for the aging lineages. The mother cells were placed at defined, well-isolated locations across the plate. Their first daughters were retained and moved to an isolated region of the plate to form colonies. Likewise, the first granddaughters of these cells were similarly retained for colony formation. The first progeny cells of these granddaughters were used to establish the 'forever young' control lineages described below. As subsequent daughters were isolated from the original mother cells, these cells were moved a short distance away from the mother, in an effort to avoid reducing nutrient density in the region and retained until the mother had produced a further two daughter cells capable of dividing. At this point the previous two daughters were moved to a 'discard' region of the plate. The total number of progeny from each mother cell was tracked and the final two replicatively active daughters of each mother cell were moved to an isolated region of the plate to form a colony for sequencing. The forever young lineages were concurrently maintained on the same plate in a similar manner, but instead of daughter cells, mother cells were removed from each daughter cell and retained until a further two mothers had been produced. The number of divisions from this line was similarly tracked. When the number of divisions reached the same number as the maximum replicative lifespan of the founding mother cell line, minus the two founding colony divisions, the last two daughters from the forever young lineages were allowed to form their own pair of colonies. In total, each set of aging and forever young lineages produced 6 colonies: the initial daughter and grand-daughter of the mother cell, the last daughter and grand-daughter of the mother cell, and the second to last and last daughter of the forever young lineage.

Whole genome sequencing

Whole genome sequencing of hybrid cell colonies was performed in parallel and in an identical manner as that described in Chapter 3 of this dissertation. Whole colonies were picked and used to inoculate liquid YPD overnight cultures. DNA isolation was performed using the Zymo DNA isolation kit. DNA was then quantified, diluted to 100 ng per 10 ul ddH_2O and enzymatically sheared using the NEB Ultra II Enzymatic Fragmentation kit and processed according to kit instructions. Indexed DNA fragments were then sequenced on the Illumina NovaSeq 6000 platform. Read information was processed as previously described (Kennedy *et al.* 2015). Briefly, read information was compared to the S288C, *S. cerevisae* masked genome (R64-1-1) using the Burrows-Wheeler Aligner (v0.6.2)(Li and Durbin 2009). Discordant and split-read groups were

removed using Samblaster (0.1.24)(Faust and Hall 2014) and Picard tools (2.21.9) AddorReplaceReadGroups function was used to generate header information. Bam files were indexed in Samtools (1.8)(Li and Durbin 2009). BAM files were then processed in sequence with the Genome Analysis Toolkit (GATK3)(DePristo *et al.* 2011) using the functions RealignerTargetCreator, IndelRealigner, LeftAlignIndels, BaseRecalibrator, and PrintReads. Samtools was used to generate a pileup file and then SNVs were scored from reads at minimum variant frequency 0.22 and minimum 18-fold depth using Varscan version 2.3.7(Koboldt *et al.* 2012).

SNV Frequency Analysis

We used an inhouse developed Python 3.8.2 tool 'LOHsearch' to scan SNV frequency returned from Varscan created noSNP.var files. The LOHsearch tool constructs a set of dictionaries for each pair of initial state colonies, final aged colonies and final 'Forever Young' colonies. Boundaries defining heterozygous and homozygous SNVs range from 20-80% frequency and 90-100% frequency, respectively, are established based on the clear separation of each of these two curves (Supplementary Fig. 2). Each set of dictionaries is scanned to determine if both sets of any given SNV change from the defined heterozygous range to homozygous for either of the end-state pairs. Any such occurrence is recorded along with contextual information including the read depth and frequency for all lineage members at that genomic position to a text document. Each of these reports was then verified for accuracy using the Integrative Genome Viewer (Robinson *et al.* 2011) and Microsoft Excel. Plots were generated using an in-house developed Python 3 script, loh_genome_graph.py, using the Matplotlib v3.1.2 Python tool.

Figures

Table 4.1

	West African	**North American**	**Sake**	**Malaysian**
Wine/European	**67987** (0.56) 178	**60159** (0.50) 201	**66313** (0.55) 182	**70687** (0.59) 171
West African		**48776** (0.40) 247	**55168** (0.46) 219	**63556** (0.53) 190

North American			**39958** (0.33) 302	**52808** (0.44) 229
Sake	Unique SNVs (%n) Bp/Polymorphism			**59300** (0.49) 204

Table 4.1 Unique SNVs available in hybrid crosses
Table describing the number of unique SNVs available in four different strains for lineage LOH analysis (top number), the percent of the masked genome unique to that strain (parentheses) and the average number of basepairs between each SNV in a hypothetical hybrid strain (bottom row of each).

Figure 4.1

Fig. 4.1 SNV map of the Malaysian x Sake cross
A representation of the location of all unique SNVs, present at a frequency of approximately 50%, in Sake X Malaysian hybrid strains. Chromosomes are depicted by blue horizontal lines, with red circles to denote centromere locations. Green lines represent each SNV unique to one of the parent strains and not detected in the other. Bases masked in the reference genome are represented by low opacity grey lines to visualize masked regions.

Figure 4.2

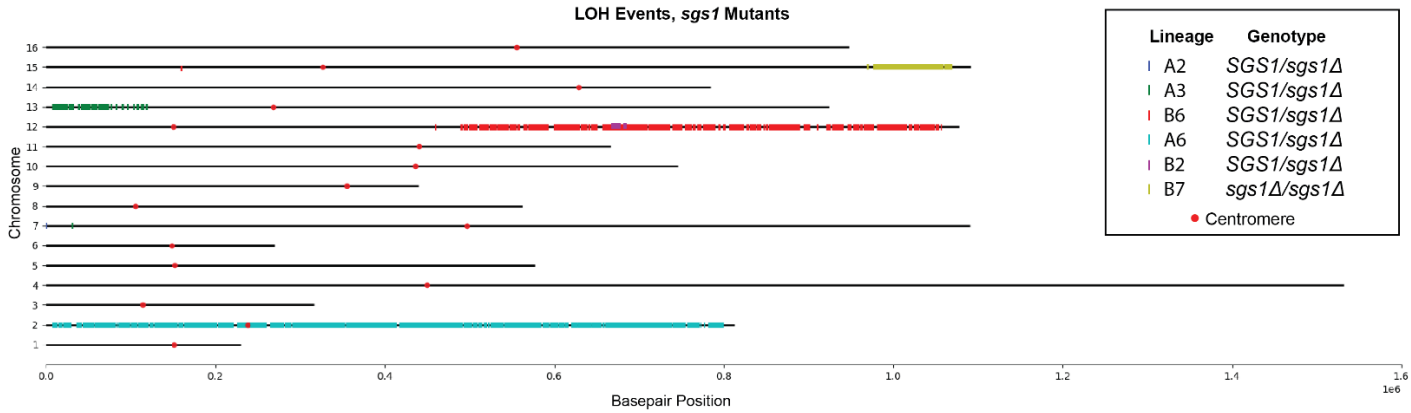


Fig. 4.2 Loss of heterozygosity in *sgs1Δ/sgs1Δ* and *SGS1/sgs1Δ* lineages
Loss of Heterozygosity events scored against the masked yeast genome (colored bars) in *sgs1* mutant hybrid lineages. Black lines represent each of the sixteen *S. cerevisiae* chromosomes and centromeres indicated by red circles.

Figure 4.3

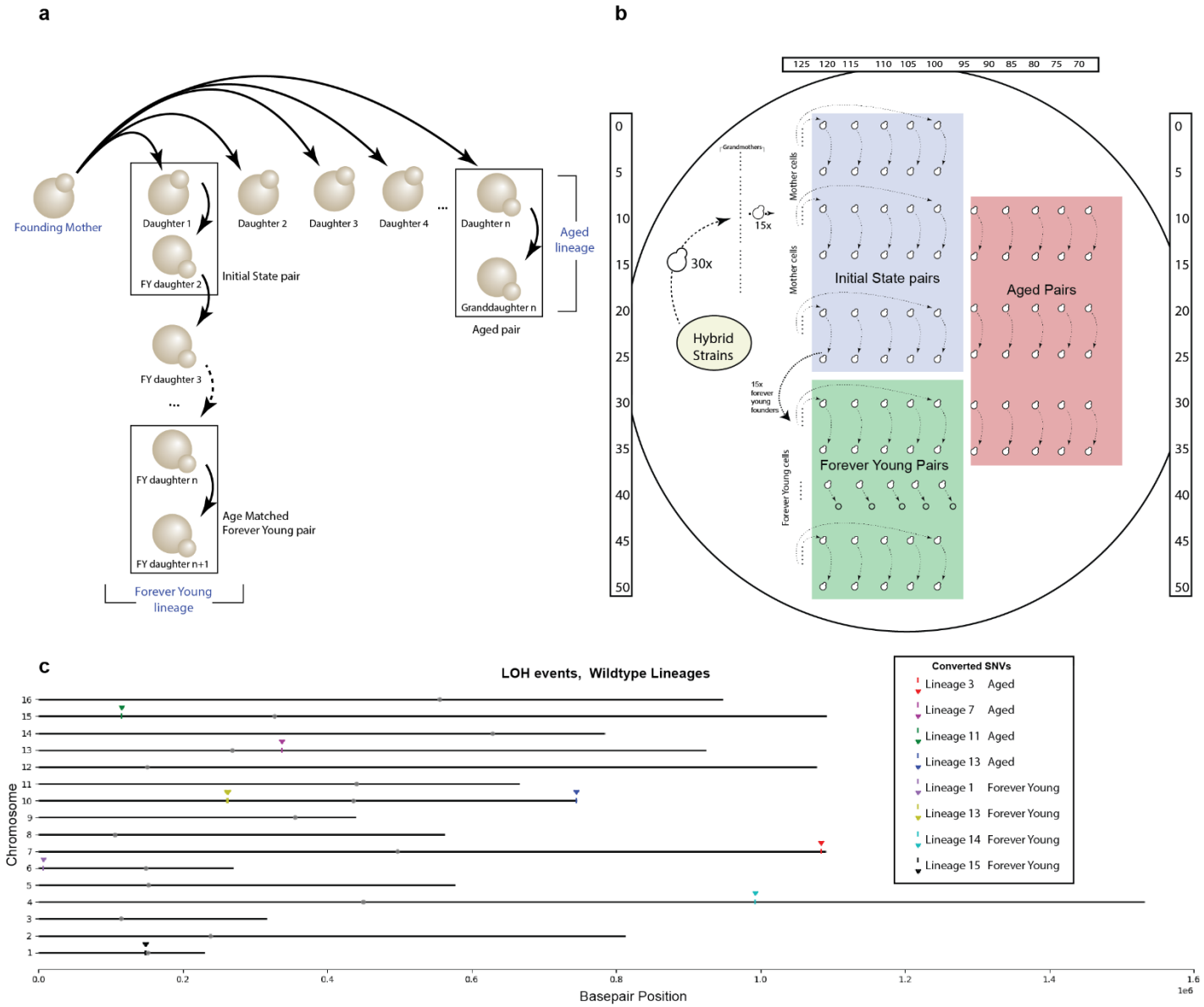


Fig. 4.3 Wildtype Wild Strain Hybrid Lineages

a Schematic of the isolation of parallel aged and control 'Forever Young' (FY) lineages for each wildtype hybrid mother cell (founding mother). Aged Lineages were isolated from the direct offspring of a single mother cell until the end of her replicative lifespan (Daughter n). Where possible, division-matched FY lineages were isolated, beginning with the first daughter of the same mother cell, by retaining the new daughter cell at each division until reaching an equivalent number of divisions (FY daughter n). Colonies derived from the final pair of cells from the Aging and FY lineages, along with the initial daughter and grand-daughter cells, are then subjected to whole genome sequencing to identify LOH events.**b** Schematic of an ideal plate containing 15 lineages as described in (a)**.** Top and side numbers indicate location on the agar plate in millimeters.

Grandmothers are selected from an initial plating of hybrid cells. Founding mothers are removed from this population to defined sections on the plate. Initial daughters and first granddaughters are similarly removed to form initial state pairs (blue). First daughters of the granddaughters form the founders of forever young lineages. Last replicatively capable daughters and their first daughters are retained to form aged pair colonies (red). Division matched forever young daughters are similarly retained to form forever young colony pairs. **c** Loss of Heterozygosity events in wildtype hybrid yeast lineages scored against the masked yeast genome (colored bars and triangles). Black lines represent each of the sixteen *S. cerevisiae* chromosomes. Centromeres represented by gray circles.

Supplemental Figures

Supplementary Table 4.1

Lineage	Strain	Lifespan	Sequenced Division	*SGS1 genotype*	LOH tracts
A1	ID0021xID0034	23	23	***SGS1/sgs1***	0
A2	ID0021xID0034	33	31	***SGS1/sgs1***	0
A3	ID0021xID0034	33	33	***SGS1/sgs1***	2
A4	ID0021xID0034	19	18	***SGS1/sgs1***	0
A5	ID0021xID0034	23	23	***SGS1/sgs1***	0
A6	ID0021xID0034	13	9	***SGS1/sgs1***	1
A8	ID0021xID0034	16	15	***SGS1/sgs1***	0
B1	ID0022xID0036	2	2	***SGS1/sgs1***	0
B2	ID0022xID0036	4	3	***SGS1/sgs1***	1
B6	ID0022xID0036	35	33	***SGS1/sgs1***	2
B8	ID0022xID0036	18	17	***SGS1/sgs1***	0
B9	ID0022xID0036	17	13	***SGS1/sgs1***	0
Total			***220***	***SGS1/sgs1***	***6***
B7	ID0022xID0036	13	9	***sgs1/sgs1***	1

Supplementary Table 4.1 *sgs1* mutant lifespan and loss of heterozygosity
Loss of heterozygosity in *Sake* X *Malaysian* sgs1 mutant strains. Total number of divisions scored in aged mother cell indicated by lifespan column. Sequenced division column indicates the last pair of colonies that could be sequenced due to colony formation failure of one of the two last daughter colony pairs.

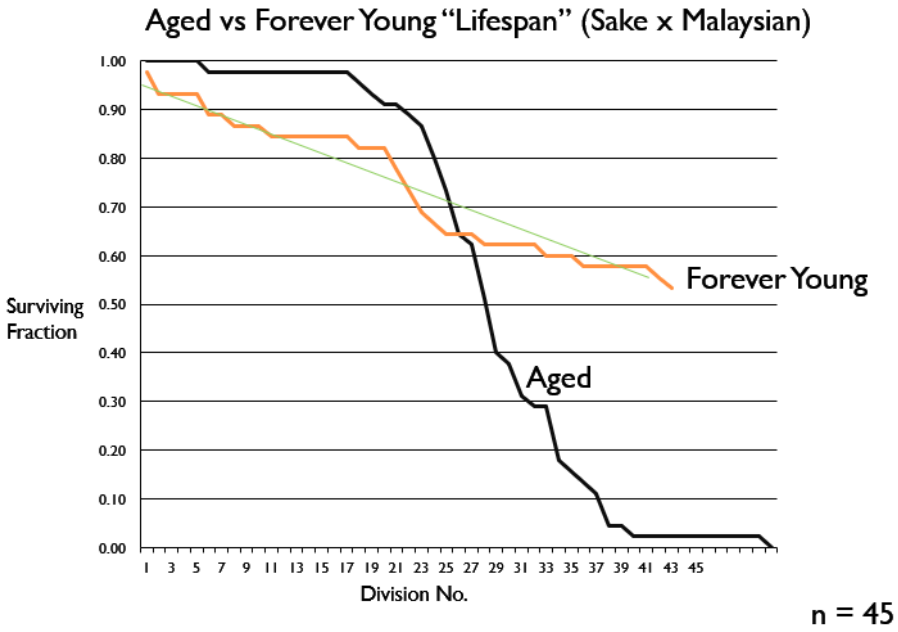
Supplementary Table 4.2

Lineage	Aged LS	FY LS*	FY division sequenced	Aged LOH tracts	FY LOH tracts
1	25		25		1

2	38	21	21		
3	28	11	11	1	
4	28	36	28		
5	24	24	24		
6	28	21	21		
7	25	8	8	1	
8	40	22	22		
9	26	42	26		
10	37		37		
11	30	22	22	1	
12	18	43	18		
13	20	33	20	1	1
14	27		27		1
15	28		28		1
Total	***331***		***247***	***4***	***4***

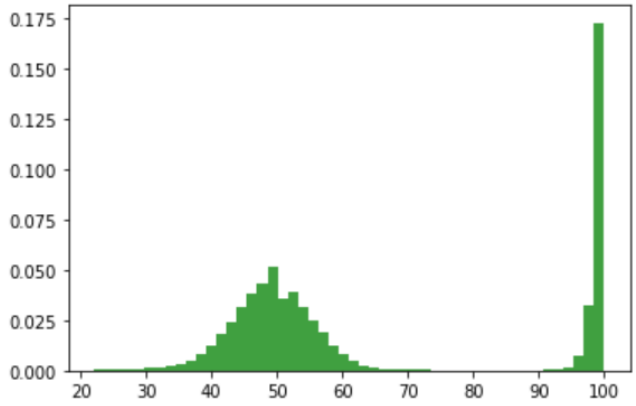
* Forever young "Lifespan" indicated in cases of replicative failure. Where blank no mortality was observed in >45 divisions.

Supplementary Figure 4.1



Supplementary Figure 4.1: Linear regression of Forever Young lineage 'lifespan'
Lifespan of aging mother cells (black line, *n* = 45) and parallel forever young lineage (orange) as represented by the surviving fraction of the population by the number of cellular divisions. Linear regression of forever young lineage lifespan (green line).

Supplementary Figure 4.2



Supplementary Figure 4.2: Distribution of detected SNV frequency
Two distinct peaks describing heterozygous and homozygous distributions of SNVs

Acknowledgements

We would like to thank the Kaeberlein lab for access to wild strains used throughout this study. This study was supported by the National Institute for General Medical Sciences (NIH/NIGMS R01GM118854). ITD was supported by the Genetic Approaches to Aging Training Grant (NIH/NIA T32AG000057) and the Biological Mechanisms for Healthy Aging Training Grant (NIH/NIA T32 AG066574 in). The content herein is solely the responsibility of the authors and does not necessarily represent the official views of the NIH, NIA, or NIGMS.

Chapter Four References

Andersen S. L., A. Zhang, M. Dominska, M. Moriel-Carretero, E. Herrera-Moyano, *et al.*, 2016 High-Resolution Mapping of Homologous Recombination Events in rad3 Hyper-Recombination Mutants in Yeast. Plos Genet 12: e1005938. https://doi.org/10.1371/journal.pgen.1005938

Brachmann C. B., A. Davies, G. J. Cost, E. Caputo, J. Li, *et al.*, 1998 Designer deletion strains derived from Saccharomyces cerevisiae S288C: A useful set of strains and plasmids for PCR-mediated gene disruption and other applications. Yeast 14: 115–132. https://doi.org/10.1002/(sici)1097-0061(19980130)14:2<115::aid-yea204>3.0.co;2-2

Cavenee W. K., T. P. Dryja, R. A. Phillips, W. F. Benedict, R. Godbout, *et al.*, 1983 Expression of recessive alleles by chromosomal mechanisms in retinoblastoma. Nature 305: 779–784. https://doi.org/10.1038/305779a0

Christman M. F., F. S. Dietrich, and G. R. Fink, 1988 Mitotic recombination in the rDNA of S. cerevisiae is suppressed by the combined action of DNA topoisomerases I and II. Cell 55: 413–425. https://doi.org/10.1016/0092-8674(88)90027-x

DePristo M. A., E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, *et al.*, 2011 A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat Genet 43: 491–498. https://doi.org/10.1038/ng.806

Faust G. G., and I. M. Hall, 2014 SAMBLASTER: fast duplicate marking and structural variant read extraction. Bioinformatics 30: 2503–2505. https://doi.org/10.1093/bioinformatics/btu314

Gietz R. D., and R. H. Schiestl, 2007 High-efficiency yeast transformation using the LiAc/SS carrier DNA/PEG method. Nat Protoc 2: 31–34. https://doi.org/10.1038/nprot.2007.13

Huang Y., R. F. Boynton, P. L. Blount, R. J. Silverstein, J. Yin, *et al.*, 1992 Loss of heterozygosity involves multiple tumor suppressor genes in human esophageal cancers. Cancer Res 52: 6525–30.

Kaiser C., S. Michaelis, and A. Mitchell, 1994 *Methods in Yeast Genetics: A Cold Spring Harbor Laboratory Course Manual*.

Kennedy S. R., E. M. Schultz, T. M. Chappell, B. Kohrn, G. M. Knowels, *et al.*, 2015 Volatility of Mutator Phenotypes at Single Cell Resolution. Plos Genet 11: e1005151. https://doi.org/10.1371/journal.pgen.1005151

Kirkwood T. B. L., 2005 Understanding the Odd Science of Aging. Cell 120: 437–447. https://doi.org/10.1016/j.cell.2005.01.027

Koboldt D. C., Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, *et al.*, 2012 VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res 22: 568–576. https://doi.org/10.1101/gr.129684.111

Kramara J., B. Osia, and A. Malkova, 2018 Break-Induced Replication: The Where, The Why, and The How. Trends Genet 34: 518–531. https://doi.org/10.1016/j.tig.2018.04.002

Lee M. E., W. C. DeLoache, B. Cervantes, and J. E. Dueber, 2015 A Highly Characterized Yeast Toolkit for Modular, Multipart Assembly. Acs Synth Biol 4: 975–986. https://doi.org/10.1021/sb500366v

Li H., and R. Durbin, 2009 Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics 25: 1754–1760. https://doi.org/10.1093/bioinformatics/btp324

Lindstrom D. L., and D. E. Gottschling, 2009 The Mother Enrichment Program: A Genetic System for Facile Replicative Life Span Analysis in Saccharomyces cerevisiae. Genetics 183: 413–422. https://doi.org/10.1534/genetics.109.106229

Lindstrom D. L., C. K. Leverich, K. A. Henderson, and D. E. Gottschling, 2011 Replicative Age Induces Mitotic Recombination in the Ribosomal RNA Gene Cluster of Saccharomyces cerevisiae. Plos Genet 7: e1002015. https://doi.org/10.1371/journal.pgen.1002015

Liti G., D. M. Carter, A. M. Moses, J. Warringer, L. Parts, *et al.*, 2009 Population genomics of domestic and wild yeasts. Nature 458: 337--341. https://doi.org/10.1038/nature07743

Lo Y.-C., K. S. Paffett, O. Amit, J. A. Clikeman, R. Sterk, *et al.*, 2006 Sgs1 Regulates Gene Conversion Tract Lengths and Crossovers Independently of Its Helicase Activity†. Mol Cell Biol 26: 4086–4094. https://doi.org/10.1128/mcb.00136-06

Louvel H., A. Gillet-Markowska, G. Liti, and G. Fischer, 2014 A set of genetically diverged Saccharomyces cerevisiae strains with markerless deletions of multiple auxotrophic genes. Yeast 31: 91–101. https://doi.org/10.1002/yea.2991

Lydeard J. R., Z. Lipkin-Moore, S. Jain, V. V. Eapen, and J. E. Haber, 2010 Sgs1 and Exo1 Redundantly Inhibit Break-Induced Replication and De Novo Telomere Addition at Broken Chromosome Ends. Plos Genet 6: e1000973. https://doi.org/10.1371/journal.pgen.1000973

McMurray M. A., and D. E. Gottschling, 2003 An Age-Induced Switch to a Hyper-Recombinational State. Science 301: 1908–1911. https://doi.org/10.1126/science.1087706

McVey M., M. Kaeberlein, H. A. Tissenbaum, and L. Guarente, 2001 The short life span of Saccharomyces cerevisiae sgs1 and srs2 mutants is a composite of normal aging processes and mitotic arrest due to defective recombination. Genetics 157: 1531–42.

Morley A., 1998 Somatic Mutation and Aging. Ann Ny Acad Sci 854: 20–22. https://doi.org/10.1111/j.1749-6632.1998.tb09888.x

Myung K., A. Datta, C. Chen, and R. D. Kolodner, 2001 SGS1, the Saccharomyces cerevisiae homologue of BLM and WRN, suppresses genome instability and homeologous recombination. Nat Genet 27: 113–116. https://doi.org/10.1038/83673

O'Connell K., S. Jinks-Robertson, and T. D. Petes, 2015 Elevated Genome-Wide Instability in Yeast Mutants Lacking RNase H Activity. Genetics 201: 963–975. https://doi.org/10.1534/genetics.115.182725

Robinson J. T., H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, *et al.*, 2011 Integrative genomics viewer. Nat Biotechnol 29: 24–26. https://doi.org/10.1038/nbt.1754

Rossi D. J., D. Bryder, J. Seita, A. Nussenzweig, J. Hoeijmakers, *et al.*, 2007 Deficiencies in DNA damage repair limit the function of haematopoietic stem cells with age. Nature 447: 725–729. https://doi.org/10.1038/nature05862

Shaw W. M., H. Yamauchi, J. Mead, G.-O. F. Gowers, D. J. Bell, *et al.*, 2019 Engineering a Model Cell for Rational Tuning of GPCR Signaling. Cell 177: 782-796.e27. https://doi.org/10.1016/j.cell.2019.02.023

Watt P. M., E. J. Louis, R. H. Borts, and I. D. Hickson, 1995 Sgs1: A eukaryotic homolog of E. coil RecQ that interacts with topoisomerase II in vivo and is required for faithful chromosome segregation. Cell 81: 253–260. https://doi.org/10.1016/0092-8674(95)90335-6

Watt P. M., I. D. Hickson, R. H. Borts, and E. J. Louis, 1996 SGS1, a homologue of the Bloom's and Werner's syndrome genes, is required for maintenance of genome stability in Saccharomyces cerevisiae. Genetics 144: 935–45.

Chapter Five
Genomic Instability at Single Cell Resolution

Ian T. Dowsett

Author Information
Department of Laboratory Medicine and Pathology, University of Washington, Seattle, WA 98195-7705, USA

In the previous chapters I have shown how a single cell sequencing approach is an essential method for making observations and testing hypotheses of processes that can be masked using standard population-based sequencing approaches. I outline two different methods that operate using such an approach to address several previously unaddressed questions relating to genomic instability.

Based on previous research discussed in chapter two(Kennedy *et al.* 2015), I first tested two competing hypotheses in a strong mutator yeast strain. According to one hypothesis, that of mutator volatility, divisions were subject to one of a variable set of polymerase error rates per division. An alternative possibility was that cells unequally shared mismatches between mother and daughter cells. By capturing and scoring the full set of mismatches made per division in a DNA polymerase δ proofreading and mismatch repair deficient (*pol3-01/pol3-01 msh6Δ/msh6Δ*) mutator model I was able to test these two hypotheses. Using mixture modeling and Akaike Information Criterion (AIC) the distribution of mismatches per division was a better fit to a broad negative binomial, rather than a single or two Poisson model, suggesting that a composite of multiple Poissons govern the underlying mutation rate of an error-prone polymerase. This finding strongly supported a mutator volatility model. I next explored the reason for this broader distribution. One way to achieve a broader distribution of replication errors than expected is for each chromatid to be subject to an independent replication error rate. Alternatively, each division might be subject to a single Poisson-derived error rate drawn from a broader gamma distributed set of Poissons. Simulating each of these scenarios I found that the latter was a better match for our dataset suggesting that each division is subject to single governing mutation rate. I also noted an intriguing pattern in our data in which errors seemed to be unevenly distributed at the chromosome level between mitotically dividing cells. I traced this observation to the basic biological processes of semiconservative DNA replication and mitotic segregation of chromosomes, but exploring the effects of this process on simulated yeast cells produced the unexpected finding that these basic mechanisms increased the range of distributed errors. This effect was greatly amplified in simulated human cells due to the substantially larger chromosome size. Taken together these results indicate that human cancer populations with a high mutation rate are likely subject to a much more variable degree of subclonal heterogeneity than would be expected using a single Poisson

model of mutagenesis, the norm in the field for over 70 years(Luria and Delbrück 1943; Jackson and Loeb 1998).

In chapter three, I asked whether the potent cancer-associated *POLE* allele, *POLE-P286R* modeled in yeast (*pol2-P301R*) similarly exhibited mutator volatility at single cell resolution and whether that volatility was subject to genetic manipulation of the S-phase checkpoint. This particular mutator allele is suspected to readily extend mispairs and likely contributes to a larger fraction of the genome than its wildtype counterpart(Kane and Shcherbakova 2014; Xing *et al.* 2019). Based on previous published research from the lab(Williams *et al.* 2015), I suspected that deletion of the *RNR* regulating gene *DUN1* would diminish the potency of the mutator phenotype by decreasing dNTP levels. It was unclear, however, if the regulated fluctuation of dNTP concentrations during S-phase played a primary role in driving the phenomenon of mutator volatility. To test this, I deleted *DUN1* in *POL2/pol2-P301R msh2Δ/msh2Δ* mutator cells and compared the distribution of mutations/cell division to cells with *DUN1.* In both cases I found strong evidence for mutator volatility, suggesting that the S-phase checkpoint and, in turn, dNTP concentration is not primarily responsible for the phenomenon of mutator volatility in this genetic background. This would imply that dNTP targeting drug interventions, while likely capable of reducing the overall mutation rate of a growing tumor, would be unlikely to prevent the broadening of genetic heterogeneity outlined in chapter two.

In chapter four, I describe a method of screening the entire genome of a single yeast cell for changes in the rate of loss of heterozygosity (LOH) with replicative age. Such events may synergize with heterozygous somatic mutations or pre-existing variants to give rise to null alleles to the detriment of an organism's health. Previous research exploring LOH in relation to replicative aging in yeast used a colony sectoring assay that was able to detect an increase associated with age at select loci (Lindstrom *et al.* 2011). Such an approach is limited to reporting LOH at a single locus at a time and could not fully interrogate the final daughters in a mother cell's lifespan. I sought to create an experimental approach that would address both of these limitations with coverage encompassing the entire yeast genome. This method uses hybrid wild cells with tens of thousands of unique SNVs and then, by sequencing daughter colonies of single mother cells, I identified regions in which those SNVs become homozygous over the course of a lifespan. I first verified that this method was a viable approach by deleting the gene *SGS1* to elevate recombination in these cells and noted a number of large tracts of LOH across the genome. I then investigated if LOH increased dramatically at the end of life in wildtype hybrid cells. In comparison to a division-matched control, I failed to find such a marked increase in an aged lineage, though I did observe a number of small LOH tracts in both aged and control lineages. The small number of observed events in each lineage suggests this method might be readily adapted for use in testing other hypotheses, including addressing whether sets of mutations in genes of interests are colinear or if both copies of a gene are required to be mutated in order to observe a given phenotype.

The projects and findings outlined above and in the preceding chapters are examples of how a single cell resolution approach can provide valuable, and at times surprising, insights into processes that give rise to genomic instability. Findings, such as mutator volatility and the asymmetric segregation of mismatches among chromosomes would certainly be masked by the effects of averaging a large number of divisions into a single mutation rate. The processes that give rise to these forms of genomic instability operate at the level of single cells and as such it is essential to continue to pursue methods of investigating these mechanisms at this level of resolution.

Chapter Five References

Jackson A. L., and L. A. Loeb, 1998 The mutation rate and cancer. Genetics 148: 1483–90.

Kane D. P., and P. V. Shcherbakova, 2014 A common cancer-associated DNA polymerase ε mutation causes an exceptionally strong mutator phenotype, indicating fidelity defects distinct from loss of proofreading. Cancer Res 74: 1895--1901. https://doi.org/10.1158/0008-5472.can-13-2892

Kennedy S. R., E. M. Schultz, T. M. Chappell, B. Kohrn, G. M. Knowels, *et al.*, 2015 Volatility of Mutator Phenotypes at Single Cell Resolution. Plos Genet 11: e1005151. https://doi.org/10.1371/journal.pgen.1005151

Lindstrom D. L., C. K. Leverich, K. A. Henderson, and D. E. Gottschling, 2011 Replicative Age Induces Mitotic Recombination in the Ribosomal RNA Gene Cluster of Saccharomyces cerevisiae. Plos Genet 7: e1002015. https://doi.org/10.1371/journal.pgen.1002015

Luria S. E., and M. Delbrück, 1943 Mutations of Bacteria from Virus Sensitivity to Virus Resistance. Genetics 28: 491–511.

Williams L. N., L. Marjavaara, G. M. Knowels, E. M. Schultz, E. J. Fox, *et al.*, 2015 dNTP pool levels modulate mutator phenotypes of error-prone DNA polymerase ε variants. Proc National Acad Sci 112: E2457–E2466. https://doi.org/10.1073/pnas.1422948112

Xing X., D. P. Kane, C. R. Bulock, E. A. Moore, S. Sharma, *et al.*, 2019 A recurrent cancer-associated substitution in DNA polymerase ε produces a hyperactive enzyme. Nat Commun 10: 374. https://doi.org/10.1038/s41467-018-08145-2

Appendix

Chapter Two Scripts

FMM.R

```
# This script uses finite mixed modeling to develop likelihood ratios based on the
provided input file and compare those values using AIC and BIC
library(countreg)
library(data.table)
library(dplyr)
library(flexmix)
library(ggplot2)
# This assumes there is a directory called "Data" in the working directory
dat <- data.table::fread("Data/MutSummary.csv")
p <- ggplot(dat, aes(x=Score,
                     y=..density..,
                     group=as.factor(Lineage),
                     fill=as.factor(Lineage),
                     colour=as.factor(Lineage))) +
   geom_histogram(position="identity")
printTable <- function(dat, filename, digits=3, hline_pos=0) {
   sink(filename)
   cat("\\begin{center}", '\n')
   dat %>%
      xtable::xtable(digits=digits) %>%
      print(
           floating=FALSE,
           include.rownames=FALSE,
           latex.environments="center",
           hline.after=c(0, hline_pos),
           sanitize.text.function=function(x){x})
   cat("\\end{center}", '\n')
   sink()}
getComponentParameters <- function(model,
                                   distribution) {
   if(distribution == "poisson") {
      params <- parameters(model)} else if(distribution == "negative binomial") {
      mu <- model$coef
      theta <- summary(model)$theta
      params <- list(mu=mu,
                     theta=theta)}
   return(params)}
getComponentDensity <- function(x,
                                params,
                                component_prop,
                                distribution) {
```

```
    dens <- NA
    if(distribution == "poisson") {
        dens <- component_prop*dpois(x, exp(params))
    } else if(distribution == "negative binomial") {
        dens <- component_prop*dnbinom(x, mu=exp(params$mu),
                                       size=params$theta)}
    return(dens)}

getComponentProportions <- function(model,
                                    distribution) {
    if(distribution == "poisson") {
    proportions <- attr(model, "prior")
    } else if (distribution == "negative binomial") {
        proportions <- 1}
    return(proportions)}
getFullDensity <- function(x,
                           model,
                           distribution) {
    props <- model %>% getComponentProportions(distribution=distribution)
    params <- model %>% getComponentParameters(distribution=distribution)
    if(distribution == "negative binomial") {
        params <- list(params)}
    dens <- mapply(function(y, z) {
                       getComponentDensity(x, y, z,
                                           distribution=distribution)},
                   params,
                   props
                  )
    if(length(x) > 1) {
        dens <- dens %>% apply(., 1, sum)
    } else {
        dens <- dens %>% sum
}return(dens)}
getNegativeBinomialModel <- function(values,
                                     ...
                                    ) {
    model <- glm.nb(values ~ 1)
    return(model)}
getPoissonModel <- function(values,
                            k) {
    model <- flexmix(values ~ 1,
                     data=dat,
                     k=k,
                     model=FLXMRglm(family="poisson"))
    return(model)}
getModelAIC <- function(model,
```

```
                    distribution) {
    if(distribution == "poisson") {
        aic <- attr(summary(model), "AIC")
    } else if (distribution == "negative binomial") {
        aic <- model$aic}
    return(aic)}
getModelBIC <- function(model) {
    return(model %>% BIC)}
getPoissonMixtureDispersion <- function(model) {
    p <- model %>% getComponentProportions(distribution="poisson")
    lambda <- model %>%
        getComponentParameters(distribution="poisson") %>%
        exp
    mu_M <- sum(p*lambda)
    mu2_M <- sum(p*lambda*(lambda + 1))
    dispersion <- mu2_M/mu_M - mu_M
    return(dispersion)}

getNegativeBinomialDispersion <- function(model) {
    params <- model %>% getComponentParameters(distribution="negative binomial")
    lambda <- params$mu %>% exp
    theta <- params$theta
    dispersion <- 1 + lambda/theta
    return(dispersion)}
getDensityMaximum <- function(observations,
                              model,
                              distribution) {
    max <- 0
    for(x in range(observations)) {
        val <- getFullDensity(x,
                              model,
                              distribution)
        if(val > max) {
            max <- val}  }
    return(max)}
plotPoissonModel <- function(observations,
                             model,
                             filename,
                             distribution ) {
    k <- attr(model, "k")
    params <- model %>% parameters
    props <- model %>% getComponentProportions(distribution=distribution)
    pdf(filename,
        width=8,
        height=8)
    par(mar=c(5,6,4,1)+.1)
```

```
    hist(observations,
         prob=TRUE,
         breaks=20,
         cex.lab=2,
         cex.axis=2,
         cex.main=2,
         ylim=c(0, 0.05),
         xlab="Mutation score",
         ylab="Frequency",
         main="Histogram and Poisson mixture p.m.f., K = " %>% paste0(k))
    for(i in 1:k) {
    curve(getComponentDensity(x,
                              params=ifelse(distribution == "poisson",
                                     params[i],
                                     params[, i]),
                              component_prop=props[i],
                              distribution="poisson"),
         from=min(observations),
         to=max(observations),
         n=max(observations) - min(observations) + 1,
         add=TRUE,
         col="blue",
         lty=2,
         lwd=2)}
    curve(getFullDensity(x,
                         model,
                         distribution="poisson"),
         from=min(observations),
         to=max(observations),
         n=max(observations) - min(observations) + 1,
         add=TRUE,
         col="red")
    legend("topleft",
           legend=c("Component density",
             "Full mixture density"),
           col=c("blue", "red"),
           lty=c(2, 1),
           cex=1.5)
    dev.off()}
plotNegativeBinomialModel <- function(observations,
                                      model,
                                      filename) {
    k <- 1
    params <- model %>%
       getComponentParameters(distribution="negative binomial")
    pdf(filename,
```

```
        width=8,
        height=8)
    par(mar=c(5,6,4,1)+.1)
    hist(observations,
         prob=TRUE,
         breaks=20,
         cex.lab=2,
         cex.axis=2,
         cex.main=2,
         ylim=c(0, 0.05),
         xlab="Mutation score",
         ylab="Frequency",
         main="Histogram and negative binomial p.m.f.")
    curve(getFullDensity(x,
                         model,
                         distribution="negative binomial"),
          from=min(observations),
          to=max(observations),
          n=max(observations) - min(observations) + 1,
          add=TRUE,
          col="red")
    legend("topleft",
           legend="Negative binomial density",
           col=c("red"),
           cex=1.5,
           lty=1)
    dev.off()}

runAnalysis <- function(observations,
                        K,
                                                    analysis_name) {
    pois_models <- list()
    pois_AICs <- {}
    pois_BICs <- {}
    pois_dispersions <- {}
    for(k in 1:K) {
        model <- getPoissonModel(observations, k)
        if(attr(model, "k") == k) {
            pois_models[[k]] <- model
            pois_AICs[k] <- getModelAIC(model,
                                        distribution="poisson")
            pois_BICs[k] <- model %>% getModelBIC
            pois_dispersions[k] <- model %>% getPoissonMixtureDispersion}}

    nb_model <- getNegativeBinomialModel(observations)
    nb_AIC <- getModelAIC(nb_model,
```

```
                    distribution="negative binomial")
    nb_BIC <- nb_model %>% getModelBIC
    nb_dispersion <- nb_model %>% getNegativeBinomialDispersion

        figure_dir <- file.path("Figures",
                                                    analysis_name)
    unlink(figure_dir, rec=T)
        dir.create(figure_dir, showWarnings=FALSE)
    for(model in pois_models) {
       plotPoissonModel(observations,
                    model,
                    filename=file.path(figure_dir,
                                paste0("pois_mixture_",
                                attr(model, "k"), ".pdf")),
                    distribution="poisson")}

    plotNegativeBinomialModel(observations,
                        nb_model,
                        filename=file.path(figure_dir,
                                                    paste0("nb_mixture.pdf") ))
    summary_df <- data.table(Model=paste("Poission, k = ", 1:length(pois_models)),
                    AIC=pois_AICs,
                    BIC=pois_BICs,
                    "$\\estim D$"=pois_dispersions)
    summary_df <- rbind(summary_df,
                data.table(Model="Negative Binomial",
                        AIC=nb_AIC,
                        BIC=nb_BIC,
                        "$\\estim D$"=nb_dispersion))

        table_dir <- file.path("Tables", analysis_name)
        dir.create(table_dir, showWarnings=FALSE)
        printTable(summary_df, file.path(table_dir,
                                                        "summary.tex"))
    return(summary_df)}
getDaughterSums <- function(dat) {
    d1_scores <- dat[dat$SegregationID == 1, ]$Score
    d2_scores <- dat[dat$SegregationID == 2, ]$Score
    d_scores <- d1_scores + d2_scores
    return(d_scores)}
getMotherSums <- function(dat) {
    m1_scores <- dat[dat$SegregationID == 3, ]$Score
    m2_scores <- dat[dat$SegregationID == 4, ]$Score
    m_scores <- m1_scores + m2_scores
    return(m_scores)}
getFullSums <- function(dat) {
```

```
    d_scores <- dat %>% getDaughterSums
    m_scores <- dat %>% getMotherSums
    full_scores <- d_scores + m_scores
    return(full_scores)}
K <- 4
# Run analysis for Mother 2 (M2)
runAnalysis(dat[dat$SegregationID == 4, ]$Score,
            K=K,
                        analysis_name="M2")
# Run daughter-only analysis
daughter_sums <- dat %>% getDaughterSums
runAnalysis(daughter_sums[daughter_sums > 80],
            K=K,
                        analysis_name="Daughter")
runAnalysis(daughter_sums,
            K=K,
                        analysis_name="Daughter_full")
# Run mother-only analysis
mother_sums <- dat %>% getMotherSums
runAnalysis(mother_sums[mother_sums > 60],
            K=K,
                        analysis_name="Mother")
runAnalysis(mother_sums,
            K=K,
                        analysis_name="Mother_full")
# Run analysis for all data
full_sums <- dat %>% getFullSums
runAnalysis(full_sums[full_sums > 170],
                        K=K,
                        analysis_name="all")
runAnalysis(full_sums,
                        K=K,
                        analysis_name="all_full")
# Paired t-test
m_sums <- dat %>% getMotherSums
d_sums <- dat %>% getDaughterSums
pdf("Figures/m_d_box.pdf")
boxplot(m_sums,
        d_sums,
        names=c("Mothers", "Daughters"),
        xlab="Cell type",
        ylab="Score",
        main="Mutation scores for mothers vs daughters")
dev.off()
t_test <- t.test(m_sums, d_sums,
                 paired=TRUE)
```

Sequencing Pipeline script: eex_yeast_pipeline.sh

```
#!/bin/bash
set -e
#make subdirectories to keep things clean.  We usually run this script from within the directory that contains the fasta files for the strains listed after "strain_name".
mkdir tmp
mkdir discordant
mkdir split
REFPATH=~/Bioinformatics/genomes
SLICENUM=1
#export $REFPATH
strain_name='AH164_2_2 AH164_2_3 AH164_10_1 AH164_10_2 AH164_10_4 AH164_11_1 AH164_11_2 AH164_11_3 AH164_12_1 AH164_12_3 AH164_12_4'
#Alignment of strain genomes against entire yeast genome

count=0
for strain in $strain_name; do
bwa mem -M $REFPATH/yeast/S288C-masked-genome.fasta ${strain}_seq1_fixed.fq.gz ${strain}_seq2_fixed.fq.gz |samblaster -M -d discordant/${strain}.disc.sam -s split/${strain}.split.sam | grep -v SA:Z |grep -v XA: | samtools view -Sb -q 10 - |samtools sort -o tmp/${strain}_pe.unique_sorted.bam - &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
   java -Xmx1g -jar ~/Bioinformatics/programs/picard.jar AddOrReplaceReadGroups INPUT=tmp/${strain}_pe.unique_sorted.bam OUTPUT=tmp/${strain}_pe.fixedhdr.bam RGID=1 RGLB=1 RGPL=illumina   RGPU=TTAATA RGSM=${strain} VALIDATION_STRINGENCY=LENIENT &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
   samtools index tmp/${strain}_pe.fixedhdr.bam &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
   java -Xmx1g -jar ~/Bioinformatics/programs/GATK3/GenomeAnalysisTK.jar -T RealignerTargetCreator -I tmp/${strain}_pe.fixedhdr.bam -R ${REFPATH}/yeast/S288C-masked-genome.fasta -o ${strain}.intervals &
        let count+=1
```

```
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
   java -Xmx1g -jar ~/Bioinformatics/programs/GATK3/GenomeAnalysisTK.jar -T
IndelRealigner -R ${REFPATH}/yeast/S288C-masked-genome.fasta -I
tmp/${strain}_pe.fixedhdr.bam -targetIntervals ${strain}.intervals -o tmp/${strain}_pe-
realigned.bam &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
   java -Xmx1g -jar ~/Bioinformatics/programs/GATK3/GenomeAnalysisTK.jar -T
LeftAlignIndels -R ${REFPATH}/yeast/S288C-masked-genome.fasta -I
tmp/${strain}_pe-realigned.bam -o tmp/${strain}_pe-leftaligned.bam &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
   java -Xmx1g -jar ~/Bioinformatics/programs/GATK3/GenomeAnalysisTK.jar -T
BaseRecalibrator -I tmp/${strain}_pe-leftaligned.bam -R ${REFPATH}/yeast/S288C-
masked-genome.fasta -knownSites ${REFPATH}/yeast/BY4741-
diploid_snp_sorted_final.vcf -o tmp/${strain}_pe-recalibrated.grp &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
for strain in $strain_name; do
   java -Xmx1g -jar ~/Bioinformatics/programs/GATK3/GenomeAnalysisTK.jar -T
PrintReads -R ${REFPATH}/yeast/S288C-masked-genome.fasta -I tmp/${strain}_pe-
leftaligned.bam -BQSR tmp/${strain}_pe-recalibrated.grp -o ${strain}_pe-
recalibrated.bam &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
          samtools mpileup -Bf $REFPATH/yeast/S288C-masked-genome.fasta
${strain}_pe-recalibrated.bam > ${strain}_pe.mpileup &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
```

```
    java -jar ~/Bioinformatics/programs/VarScan.v2.3.9.jar mpileup2snp
${strain}_pe.mpileup --min-coverage 18 --min-var-freq .1 --strand-filter 1 |python
~/Bioinformatics/programs/variant_deSNPer2013.py -s
~/Bioinformatics/genomes/yeast/AH0401.snp > ${strain}.noSNP.var &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done
count=0
for strain in $strain_name; do
java -jar ~/Bioinformatics/programs/VarScan.v2.3.9.jar mpileup2indel
${strain}_pe.mpileup --min-coverage 18 --min-var-freq .22 --strand-filter 1 |python
~/Bioinformatics/programs/variant_deSNPer2013.py -s
~/Bioinformatics/genomes/yeast/AH0401indel.snp > ${strain}.no-indelSNP.var &
        let count+=1
        [[ $((count%${SLICENUM})) -eq 0 ]] && wait
done

count=0
for strain in $strain_name; do
samtools view -Sb discordant/${strain}.disc.sam|samtools sort -o ${strain}.disc.sort.bam
-
samtools view -Sb split/${strain}.split.sam|samtools sort -o ${strain}.split.sort.bam -
samtools index ${strain}.disc.sort.bam
samtools index ${strain}.split.sort.bam
rm *.intervals
rm -r tmp/
rm -r discordant
rm -r split
done
```

JLSlineage_caller.py

```
#!/usr/bin/env python
from collections import defaultdict
from argparse import ArgumentParser
import pandas as pd
def main():
    parser = ArgumentParser()
    parser.add_argument("--lineage_info", action='store', dest="lineage_info", help='csv
file consisting of the strain file name followed by the index in the lineage',
required=True)
    parser.add_argument("--min_depth", action='store', dest="min_depth", type=int,
help="Minimum depth to evaluate site", default=18)
    o = parser.parse_args()
    strain_dict = defaultdict(lambda: 0)
    scored_sites = defaultdict(lambda: 0)
    var_dict = defaultdict(lambda: [])
```

```
    lineage_info_file = open(o.lineage_info, 'r')

    for line in lineage_info_file:
        # creates a list of a line from the line of the CSV
        splitline = line.strip('\r\n').split(',')
        # creates a dictionary entry where key is the rep_event and the value is the above
list
        strain_dict.update({int(splitline[0]): splitline[1:]})
    lineage_info_file.close()
    site_count_file = open("site_counts.txt", 'w')
    sub_lineage_list = []
    for rep_event in strain_dict.keys():
        print rep_event
        # For the first two strains it goes through the pileup files and counts and scores the
available sites that have sufficient coverage
        for sub_lineage in strain_dict[rep_event][:2]:
            sub_lineage_list.append(sub_lineage)
            nuc_ctr = 0
            pileup_file = open(sub_lineage + '_pe.mpileup', 'r')
            for line in pileup_file:
                splitline = line.strip('\n').split('\t')
                if int(splitline[3]) > o.min_depth:
                    nuc_ctr += 1
                    scored_sites[splitline[0] + ":" + str(splitline[1])] += 1
            pileup_file.close()
            site_count_file.write(sub_lineage + '\t' + str(nuc_ctr) + '\n')
    scored_site_file = open("scored_sites.txt", 'w')
    scored_site_ctr = 0
    #creates a shared set for counting the shared sites.
    shared_set=set()
    for site in scored_sites.keys():
        #From the scored_sites dictionary that kept track of sites with at least 18-fold
coverage in one strain, we now assess how many times we counted it and compare it to
the
        #the number of strains in our sublineage list.  For example, if we have 12 strains in
our isolate_list and we observed it 12 times than we can add the site to the shared site
set.
        if scored_sites[site] == len(sub_lineage_list):
            shared_set.add(site)
            scored_site_ctr += 1
            scored_site_file.write(site + '\n')
    site_count_file.write("# Scored Sites:\t" + str(scored_site_ctr))
    scored_site_file.close()
    site_count_file.close()
    total_var_set = set()
    total_var_dict = defaultdict(lambda:set())
```

```
    for rep_event in strain_dict.keys():
        temp_list = []
        for sub_lineage in strain_dict[rep_event]:

            var_file = open(sub_lineage + ".noSNP.var", 'r')
            var_set = set()
            for line in var_file:
                splitline = line.strip('\n').split('\t')
                # all vars have the required coverage.  This asks if they are in the shared_set.
If they are then we score it.
                if splitline[0] + ":" + splitline[1] in shared_set and splitline[0] != "Chrom":
                    var_set.add(splitline[0] + ":" + splitline[1])
                    total_var_set.add(splitline[0] + ":" + splitline[1])
                    if splitline[1] not in total_var_dict[splitline[0]]:
                        total_var_dict[splitline[0]].add(int(splitline[1]))
            temp_list.append(var_set)
        strain_dict[rep_event] = temp_list
    pos_file = open("position_file.txt", 'w')
    for chrom in sorted(total_var_dict.keys()):
        pos_file.write(chrom + '\n')
        x = sorted(list(total_var_dict[chrom]))
        for pos in x:
            pos_file.write(str(pos) + '\n')
    pos_file.close()
    rep_event_keys = sorted(strain_dict.keys())
    df_master = pd.DataFrame()
    total_var_list = sorted(list(total_var_set))
    for x in rep_event_keys:
        dgd2_name = 'd' + str(x) + '_gd' + str(x) + '-2'
        gd1ggd1_name = 'gd' + str(x) + '-1_ggd' + str(x) + '-1'
        dgd1_name = 'd' + str(x) + '_gd' + str(x) + '-1'
        ddnplus1_name = 'd' + str(x) + '_d' + str(x + 1)
        df = pd.DataFrame({dgd2_name :
pd.Series(sorted(list(strain_dict[x][0].intersection(strain_dict[x][2]))),
                                index=sorted(list(strain_dict[x][0].intersection(strain_dict[x][2])))),
                                gd1ggd1_name :
pd.Series(sorted(list(strain_dict[x][1].intersection(strain_dict[x][3]))),
                               index=sorted(list(strain_dict[x][1].intersection(strain_dict[x][3])))),
                                dgd1_name:
pd.Series(sorted(list(strain_dict[x][0].intersection(strain_dict[x][1]))),
                                 index=sorted(list(strain_dict[x][0].intersection(strain_dict[x][1])))),
                                 ddnplus1_name :
pd.Series(sorted(list(strain_dict[x][0].intersection(strain_dict[x][4]))),

index=sorted(list(strain_dict[x][0].intersection(strain_dict[x][4])))))}).reindex(total_var_list)
        #df.to_csv(str(x) + '_test.csv')
```

```
        df_master = pd.concat([df_master,df], axis=1)
        #print df_master
    df_master.to_csv("test.csv")
if __name__ == "__main__":
    main()
```

variant_deSNPer2013a.py

```
#Written in python 2 in 2013 by Scott Kennedy.
import sys
from collections import defaultdict
from optparse import OptionParser
parser = OptionParser()
parser.add_option("-s", action="store", type="string", dest="SNPfile", default=None)
(o, args) = parser.parse_args()
SNPfile = open(o.SNPfile, 'r')
SNPList = []
for SNPline in SNPfile :
        try:
                SNPList.append((SNPline.split('\t')[0], SNPline.split('\t')[1]))
        except IndexError:
                print "indexerror", SNPline
                continue
for varLine in sys.stdin :
    if (varLine.strip('\n').split('\t')[0],varLine.strip('\n').split('\t')[1]) not in SNPList and 'N' not
in (varLine.strip('\n').split('\t')[2]) and 'N' not in (varLine.strip('\n').split('\t')[3]):
                print(varLine.strip('\n'))
SNPfile.close()
```

Fig2.py

```
# for Fig.2bcdef; Dowsett et al. Revised
import glob
import math
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import defaultdict
from scipy import stats
from scipy.stats import nbinom
from scipy.stats import poisson
from scipy.stats import gamma
import random
from statistics import mean
from statistics import stdev
```

#Introduction
"""The following Python 3 script simulates the segregation of mismatches between mother (Mm) and daughter (Dm) cells using two different models. Negative binomials can be interpreted as the sum of a gamma-distributed set of Poisson distributions.
In the first model, designated "nb", we imagine that each chromosome (including homologues) independently mutates with a rate drawn from a gamma-distribution. Thus, Dm and Mm can both be drawn from anywhere within the distribution generated by
the negative binomial model. In the second model, designated "gp", mutagenesis of the entire genome is governed by the same Poisson process, but the rate varies from one division to the next within a Gamma distribution of lambda values defined by the negative
binomial model. Thus, in a given division, Dm and Mm are constrained to be within the same Poisson distribution.

To better understand this volatility, the code simulates the expected distributions of full replication error counts from 50 divisions assuming different models of mutagenesis.
To capture the dispersion of each simulation, we calculate its index of dispersion, which is defined as the variance divided by the mean (σ^2/μ). By running multiple simulations,
the range of expected distributions from small sample sizes can be assessed and then compared
to the index of dispersion of the actual data."""

# Output Files(s) folder:
from pathlib import Path
File_save_location = Path.cwd()

#Defined functions
"""The first two functions were adapted from the following stackoverflow discussion thread:

https://stackoverflow.com/questions/40846992/alternative-parametrization-of-the-negative-binomial-in-scipy

For additional background, see the following two web pages:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.nbinom.html

https://en.wikipedia.org/wiki/Negative_binomial_distribution

scipy.nbinom takes n and p as shape parameters where "n" is the number of successes and "p" is
the probability of a single success. Whereas nb is usually defined as a sequence of Bernoulli
trials, repeated until a predefined nonrandom "r" number of FAILURES occurs, scipy defines nb
as a sequence of Bernoulli trials, repeated until a predefined, non-random "n" number of SUCCESSES
occurs. Of course, this is an issue of semantics in how you define success and failure: r is equivalent to n. Theta from glm.nb is equivalent to r, and therefore to n. (See: https://stats.stackexchange.com/questions/10419/what-is-theta-in-a-negative-binomial-regression-fitted-with-r
)
How scipy.stats.nbinom defines p, the probability of success, represents a potential point of
confusion. The standard definition of "success" for nb means "failure" in scipy.binom. Thus, p in scipy is actually 1-p.
"""
```
def convert_nbparams(muc, theta):
    r = theta
    var = muc + 1 / r * muc ** 2
    p = (var - muc) / var
    return r, 1-p

def nb_rvs(muc, theta, loc, size):
    return nbinom.rvs(*convert_nbparams(muc, theta),loc, size)

def linregparams (df):
    x = df["Dm"]
    y = df["Mm"]
    slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
    print("slope: %f    intercept: %f    p_value: %f    std_err: %f" % (slope, intercept, p_value, std_err))
    print("R-squared: %f" % r_value**2)

"""chr_mu_dict is a dictionary where the keys represent chromosomes in a diploid cell and the values, their associated mutation rates. This assumes each chr contributes proportionately to the genome-wide mutation rate."""
def mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b):
    ch_mu = []
    for i in chr_lenlist:
        ch_mu.append(mu*(i/tot_bp))
    tuplesA=list(zip(chr_list_a,ch_mu))
```

```
    tuplesB=list(zip(chr_list_b,ch_mu))
    chr_mu_dict = {}
    for i in tuplesA:
        chr_mu_dict[i[0]]=i[1]
    for i in tuplesB:
        chr_mu_dict[i[0]]=i[1]
    return chr_mu_dict

# creates dict where keys are chrs, and values are arrays (length c) of fixed muts from a
Poisson.
def chr_mut_p(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=v, size=c)) for (k,v) in chr_mu_dict.items()}

# as above, except rate is doubled to represent actual rate of mispair formation.
def chr_mut_bp(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=(v*2), size=c)) for (k,v) in chr_mu_dict.items()}

# multiplies value by randomly selected 0 or 1 to mimic Mendelian segregation.
def binomialize(x):
    return x * (random.randint(0,1))

# returns index of dispersion for a distribution
def index_dis(df):
    return ((df["sum"].std())**2)/df["sum"].mean()

# simulates c iterations of fixed mutations counts arising by a Poisson process.
def mk_df_p(chr_mu_dict,c):
    data_poisson_p = chr_mut_p(chr_mu_dict,c)
    df_p = pd.DataFrame.from_dict(data_poisson_p)
    df_p["sum"]=df_p.sum(axis=1)
    return df_p

# simulates c iterations of fixed mutation counts arising and segregating to two cells.
def mk_df_bp(chr_mu_dict,c):
    data_poisson2_p = chr_mut_bp(chr_mu_dict,c)
    dfas_p = pd.DataFrame.from_dict(data_poisson2_p)
    df_binomialize(dfas_p) #produces two dfs of mutation counts from reciprocol
segregants
    df2_cell1["sum"]=df2_cell1.sum(axis=1)
    add_chr_totals(df2_cell1)
    df2_cell2["sum"]=df2_cell2.sum(axis=1)
    add_chr_totals(df2_cell2)
```

```
def df_binomialize(df_input):
    df_input_bintem = (df_input * 0) + 1 #makes a dataframe of 1's w/dimensions of
df_input
    df_input_binom = df_input_bintem.applymap(binomialize) #Randomly changes 1's to
0 for cell1
#     print(df_input_binom)
    df_input_binom_inv=abs(df_input_binom - 1) #Inverse of dfas_p_binom for cell2
#     print(df_input_binom_inv)
    df2_cell1 = df_input.mul(df_input_binom) #Multiplies mutations in cell1 by 1 or 0 to
mimic segregation.
    df2_cell2 = df_input.mul(df_input_binom_inv) #Mimics mutations in cell2 the inverse
to mimic segregation.
#     df2_cell1["sum"] = df2_cell1.sum(axis=1)
    add_chr_totals(df2_cell1)
    df2_cell1 = pd.melt(df2_cell1,value_vars = chr_list,var_name='Chr',
value_name='Segregant 1')
#     df2_cell2["sum"] = df2_cell2.sum(axis=1)
    add_chr_totals(df2_cell2)
    df2_cell2 = pd.melt(df2_cell2,value_vars = chr_list,var_name='Chr',
value_name='Segregant 2')
    return df2_cell1,df2_cell2
def add_chr_totals(df_cell): #adds muts on a and b sets of chromosomes to get
observed mut/chr.
    df_cell["I"] = df_cell["Ia"] + df_cell["Ib"]
    df_cell["II"] = df_cell["IIa"] + df_cell["IIb"]
    df_cell["III"] = df_cell["IIIa"] + df_cell["IIIb"]
    df_cell["IV"] = df_cell["IVa"] + df_cell["IVb"]
    df_cell["V"] = df_cell["Va"] + df_cell["Vb"]
    df_cell["VI"] = df_cell["VIa"] + df_cell["VIb"]
    df_cell["VII"] = df_cell["VIIa"] + df_cell["VIIb"]
    df_cell["VIII"] = df_cell["VIIIa"] + df_cell["VIIIb"]
    df_cell["IX"] = df_cell["IXa"] + df_cell["IXb"]
    df_cell["X"] = df_cell["Xa"] + df_cell["Xb"]
    df_cell["XI"] = df_cell["XIa"] + df_cell["XIb"]
    df_cell["XII"] = df_cell["XIIa"] + df_cell["XIIb"]
    df_cell["XIII"] = df_cell["XIIIa"] + df_cell["XIIIb"]
    df_cell["XIV"] = df_cell["XIVa"] + df_cell["XIVb"]
    df_cell["XV"] = df_cell["XVa"] + df_cell["XVb"]
    df_cell["XVI"] = df_cell["XVIa"] + df_cell["XVIb"]
# Input parameters for nb from glm.nb in R.
mu = 4.927
theta = 60.42
# Number of desired iterations in the simulation.
```

```
n = 1000
# Conversions
muc = math.exp(mu) #mu is in log form and must be exponentiated.
gv = (muc)**2/theta #gv is the variance of the Gamma distribution of lambda
gscale = gv/muc #gives the scale paramater of the Gamma distribution of lambda
hap_mu = muc/4 #rate of fixed mutations per haploid genome.
print("The haploid rate of fixed mutations is", hap_mu)
data_nb = nb_rvs(muc,theta,loc=0,size=10000)
data_gamma = (gamma.rvs(a=theta,scale=gscale,size=n)) #Generates list of gamma
distributed rates.
data_poisson = poisson.rvs(mu=(mean(data_gamma)), size =10000)
"""Simulates Dm and Mm counts assuming mismatches arise each division from the
same negative
binomial process."""
counternb = 0
nbtuples = []
while counternb < n:
    nbDmMm = nb_rvs(muc,theta,loc=0,size=2)
    nbtuples.append(nbDmMm)
    counternb += 1
dfnb = pd.DataFrame.from_records(nbtuples,columns=['Dm','Mm'])
dfnb["sum"] = dfnb.sum(axis=1)
print("Linear regression of nb model:")
linregparams(dfnb)
# print(dfnb)
"""Simulates Dm and Mm counts assuming mismatches arise each division from a
Poisson process
whose rate is drawn each time from a gamma distribution."""
data_gamma = (gamma.rvs(a=theta,scale=gscale,size=n)) #Generates list of gamma
distributed rates.
gptuples = []
for g in data_gamma:
    gpDmMm = poisson.rvs(mu=g, size=2)
    gptuples.append(gpDmMm)
dfgp = pd.DataFrame.from_records(gptuples,columns=['Dm','Mm'])
dfgp["sum"] = dfgp.sum(axis=1)
print("Linear regression of gp model:")
linregparams(dfgp)
"""Creates data frame of actual data for comparison."""
df_data = pd.read_csv("G:\My Drive\Volatility File Share\Data\MutInSegGroups.csv")
# df_data["sum"] = df_data.sum(axis=1)
print("Linear regression of data:")
linregparams(df_data)
```

```
#simulates variation in index of dispersion of full replication error counts of Poisson vs
nb vs gp
counter = 0
sample = 50
Dplist=[]
Dnblist=[]
Dgplist=[]
while counter < n:
    """The following simulates Dm and Mm counts assuming the number of mismatches
are drawn
    in every division from the same Poisson distribution."""
    po1 = poisson.rvs(mu=muc,size=sample)
    po2 = poisson.rvs(mu=muc,size=sample)
    df_p = pd.DataFrame({'Dm':po1,'Mm':po2})
    df_p["sum"] = df_p.sum(axis=1)
    """The following simulates Dm and Mm counts assuming the number of mismatches
are drawn
    independently from within the negative binomial distribution."""
    nb1 = nb_rvs(muc,theta,loc=0,size=sample)
    nb2 = nb_rvs(muc,theta,loc=0,size=sample)
    df_nb = pd.DataFrame({'Dm':nb1,'Mm':nb2})
    df_nb["sum"] = df_nb.sum(axis=1)
    """The following simulates Dm and Mm counts assuming mismatches arise each
division from a
    Poisson process whose rate is drawn each time from a gamma distribution."""
    data_gamma = (gamma.rvs(a=theta,scale=gscale,size=sample)) #Generates list of
gamma distributed rates.
    gptuples = []
    for g in data_gamma:
        gpDmMm = poisson.rvs(mu=g, size=2)
        gptuples.append(gpDmMm)
    df_gp = pd.DataFrame.from_records(gptuples,columns=['Dm','Mm'])
    df_gp["sum"] = df_gp.sum(axis=1)
    Dplist.append(index_dis(df_p))
    Dnblist.append(index_dis(df_nb))
    Dgplist.append(index_dis(df_gp))
    counter += 1
print("The mean index of dispersion for Poisson, n=",sample,",", mean(Dplist),"+/-
",stdev(Dplist))
print("The mean index of dispersion for nb, n=", sample,",",mean(Dnblist),"+/-
",stdev(Dnblist))
```

```
print("The mean index of dispersion for gp, n=", sample,",",mean(Dgplist),"+/-
",stdev(Dgplist))
#Counts the number of unmasked nucleotides on each chr in the masked haploid yeast
genome
chr_list_a = [] #(list_a and list_b for two sets of chrs in 2n cell)
chr_list_b = []
chr_list = [] #Gives a list of chromosome names with out "a" or "b" added.
chr_lenlist = []
for chrfile in glob.glob(r"G:\My Drive\Volatility File Share\YeastGenome\*_rm.*"):
    chr_rm = chrfile.split(".")[4] #gets the chr number from the fasta file name.
    chr_list_a.append(chr_rm + "a")
    chr_list_b.append(chr_rm + "b")
    chr_list.append(chr_rm)
    with open(chrfile, 'r') as f:
        firstline = f.readline()
        count = int(firstline.split(":")[5]) #gets chr length from header of fasta file
        for line in f:  #counts "N"s in the fasta file and subtracts them from chr length.
            for i in line:
                if i == "N":
                    count = count - 1
    chr_lenlist.append(count)
tot_bp = sum(chr_lenlist)
#Models the occurrance and segregation of fixed mutations in pairs of segregating cells.
"""The following returns a gamma distribution of rates of mismatch formation
and then divides each value by 4, since we're looking at the number of fixed mutations
per haploid genome. This distribution will be our input for n divisions,
each using a different lambda. It then takes the dataframe of mismatch counts and
generates
the reciprocol mutation burdens expected in the resulting pairs of segregant cells."""
size=50
data_gamma = (gamma.rvs(a=theta, scale=gscale, size=size))/4
dfas_nbD = pd.DataFrame(columns=chr_list_a+chr_list_b) #D stands for Daughter cell
dfas_nbM = pd.DataFrame(columns=chr_list_a+chr_list_b) #M stands for Mother cell
for g in data_gamma: #with each iteration we call mutations at the same rate for D and
M
    chr_mu_dictD = mk_chr_mu_dict(g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    chr_mu_dictM = mk_chr_mu_dict(g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    data_poisson2D = chr_mut_bp(chr_mu_dictD, 1)
    data_poisson2M = chr_mut_bp(chr_mu_dictM, 1)
    dfa2D = pd.DataFrame.from_dict(data_poisson2D)
    dfa2M = pd.DataFrame.from_dict(data_poisson2M)
    dfas_nbD = dfas_nbD.append(dfa2D, ignore_index=True)
    dfas_nbM = dfas_nbM.append(dfa2M, ignore_index=True)
```

```
df2_bnb_cell1D,df2_bnb_cell2D = df_binomialize(dfas_nbD)#mimics segregation;
produces "tidy" tables.
df2_bnb_cell1M,df2_bnb_cell2M = df_binomialize(dfas_nbM)
df2totalbnbsums=pd.DataFrame()
df2totalbnbsums["Chr"]=df2_bnb_cell1D["Chr"]
df2totalbnbsums["Da"]=df2_bnb_cell1D["Segregant 1"]
df2totalbnbsums["Db"]=df2_bnb_cell2D["Segregant 2"]
df2totalbnbsums["Dm"]=df2totalbnbsums["Da"]+df2totalbnbsums["Db"]
df2totalbnbsums["Ma"]=df2_bnb_cell1M["Segregant 1"]
df2totalbnbsums["Mb"]=df2_bnb_cell2M["Segregant 2"]
df2totalbnbsums["Mm"]=df2totalbnbsums["Ma"]+df2totalbnbsums["Mb"]
df2totalbnbsums["Dm"] = df2totalbnbsums["Dm"].astype(float)
df2totalbnbsums["Mm"] = df2totalbnbsums["Mm"].astype(float)
# print(df2totalbnbsums)
"""Creates data frame of actual data (Da,Db,Dm,Ma,Mb,Mm) for comparison."""
df_datachr = pd.read_csv("G:\My Drive\Volatility File
Share\Data\MutperSegperChrm.csv")
sns.set(style="white", palette="bright", color_codes=True, rc={"lines.linewidth": 1.0})
plt.rcParams['patch.linewidth'] = 1
"""Plots the distributions of the mismatches segregated to daughter or mother cells in
the simulated and actual data.
Used for Fig.2b"""
fig1, ax = plt.subplots(figsize=(2,2))
sns.distplot(data_poisson,
             hist=False,
              bins=range(30,230,5),
#               kde=False,
#               norm_hist=True,
              color="deeppink",
             kde_kws={"lw":0.4})

sns.distplot(data_nb,
              bins=range(30,230,5),
             hist=False,
#               kde=False,
#               norm_hist=True,
              color="c",
            kde_kws={"lw":0.7})
# sns.distplot(df["Mm"],
#               bins=range(30,200,5),
#               kde=False,
#               norm_hist=True,
#               color="silver")
```

```
sns.distplot(df_data["Dm"],
             bins=range(30,230,5),
             kde=False,
             norm_hist=True,
             color="darkgoldenrod")
sns.distplot(df_data["Mm"],
             bins=range(30,230,5),
             kde=False,
             norm_hist=True,
             color="blueviolet")
ax.set(xlim=(30, 210), xlabel='Mismatches/Div', ylabel='Density')
ax.tick_params(bottom=True,left=True,labelsize=8)
fig1.savefig(f"{File_save_location}/MismatchDis612a.pdf",transparent =
True,bbox_inches='tight')
"""Plots the linear regressions of the nb model and the actual data. Used for Fig.2c"""
fig2, ax = plt.subplots(figsize=(2,2))
sns.regplot(x="Dm", y="Mm", data=dfnb, marker='o', color="lightsteelblue",
scatter_kws={'s':7},line_kws={'color':"steelblue"})
sns.regplot(x="Dm", y="Mm", data=df_data, color="green",marker='o',
scatter_kws={'s':11})
ax.set(ylim=(50, 225))
ax.set(xlim=(50, 225))
ax.set(xlabel='Dm', ylabel='Mm')
ax.tick_params(bottom=True,left=True, labelsize=8)
fig2.savefig(f"{File_save_location}/lineRegDmMmnb1V2a.pdf",transparent =
True,bbox_inches='tight')
"""Plots the linear regressions of the gp model and the actual data. Used for Fig.2d"""
fig3, ax = plt.subplots(figsize=(2,2))
sns.regplot(x="Dm", y="Mm", data=dfgp, marker='o', color="wheat",
scatter_kws={'s':7},line_kws={'color':"orange"})
sns.regplot(x="Dm", y="Mm", data=df_data, color="green",marker='o',
scatter_kws={'s':11})
ax.set(ylim=(50, 225))
ax.set(xlim=(50, 225))
ax.set(xlabel='Dm', ylabel='Mm')
ax.tick_params(bottom=True,left=True, labelsize=8)
fig3.savefig(f"{File_save_location}/lineRegDmMmgpV2a.pdf",transparent =
True,bbox_inches='tight')

"""Plots the linear regressions of the bnb model and the actual data."""
fig4, ax = plt.subplots(figsize=(2,2))
sns.regplot(x="Dm", y="Mm", data=df2totalbnbsums, marker='o', color="orange",
scatter_kws={'s':11,'alpha':0.3})
```

```
sns.regplot(x="Dm", y="Mm", data=df_datachr, color="green",marker='o',
scatter_kws={'s':11,'alpha':0.2})
ax.set(ylim=(0, 35))
ax.set(xlim=(0, 35))
ax.set(xlabel='Dm/Chr', ylabel='Mm/Chr')
ax.tick_params(bottom=True,left=True, labelsize=8)
fig4.savefig(f"{File_save_location}/lineRegDmMmChrsAa.pdf",transparent = True,
bbox_inches='tight')
sns.set(style="white", palette="bright", color_codes=True, rc={"lines.linewidth": 1.0})
plt.rcParams['patch.linewidth'] = 0
"""Plots Poisson vs nb vs gp distributions based on the full error counts for Fig.2f."""
fig5, ax = plt.subplots(figsize=(2, 2))
sns.distplot(Dplist,
             bins=[i for i in np.arange(0.25,10,0.02)],
               kde=False, norm_hist=True,
               color='gray')
ax = sns.distplot(Dnblist,
               bins=[i for i in np.arange(0.25,10,0.02)],
               kde=False, norm_hist=True,
               color='blue')
ax = sns.distplot(Dgplist,
               bins=[i for i in np.arange(0.25,10,0.02)],
               kde=False, norm_hist=True,
               color='orange')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlabel='Index of Dispersion', ylabel='Density')
fig5.savefig(f"{File_save_location}/SimFullCountsNbMM2a.pdf",transparent =
True,bbox_inches='tight')
```

Fig3ef.py

```
# For Fig.3e,f Dowsett et al. Uploaded onto Github 11.13.20
import glob
import pandas as pd
import numpy as np
from collections import defaultdict
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import poisson
from scipy.stats import nbinom
from scipy.stats import gamma
import random
```

```
import math
from statistics import mean
from statistics import stdev

# Introduction
"""The following Python 3 code simulates the extent to which mutator volatility
and asymmetric segregation of mutations lead to overdispersion  It first simulates
the contribution of asymmetric segregation alone, assuming a single Poisson process
governs the mutator phenotype in all cell divisions. We call this a Poisson-binomial
process in our annotation. The script then uses the parameters for
a negative binomial model of mismatches segregated to daughter (Dm) and mother
(Mm)
to define a gamma distribution of lambda values governing the underlying mutation
rates of individual divisions. With these values, the script then
simulates the distribution of error counts in individual divisions. This approach
is based on the fact that negative binomials can be modeled as a gamma-Poisson
mixture. Thus, when accounting for segregation, we refer to this as a gamma-Poisson-
binomial
process."""

# Output Files(s) folder:
from pathlib import Path
File_save_location  = Path.cwd()

# Defined functions
"""chr_mu_dict is a dictionary where the keys represent chromosomes in a diploid
cell and the values, their associated mutation rates. This assumes each chr
contributes proportionately to the genome-wide mutation rate."""
def mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b):
    ch_mu = []
    for i in chr_lenlist:
        ch_mu.append(mu*(i/tot_bp))
    tuplesA=list(zip(chr_list_a,ch_mu))
    tuplesB=list(zip(chr_list_b,ch_mu))
    chr_mu_dict = {}
    for i in tuplesA:
        chr_mu_dict[i[0]]=i[1]
    for i in tuplesB:
        chr_mu_dict[i[0]]=i[1]
    return chr_mu_dict
# creates dict where keys are chrs, and values are arrays (length c) of fixed muts from a
Poisson.
def chr_mut_p(chr_mu_dict,c):
```

```
    return {k:(poisson.rvs(mu=v, size=c)) for (k,v) in chr_mu_dict.items()}
# as above, except rate is doubled to represent actual rate of mispair formation.
def chr_mut_pb(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=(v*2), size=c)) for (k,v) in chr_mu_dict.items()}
# multiplies value by randomly selected 0 or 1 to mimic Mendelian segregation.
def binomialize(x):
    return x * (random.randint(0,1))
# returns index of dispersion for a distribution
def index_dis(df):
    return ((df["sum"].std())**2)/df["sum"].mean()
# simulates c iterations of fixed mutations counts arising by a Poisson process.
def mk_df_p(chr_mu_dict,c):
    data_poisson_p = chr_mut_p(chr_mu_dict,c)
    df_p = pd.DataFrame.from_dict(data_poisson_p)
    df_p["sum"]=df_p.sum(axis=1)
    return df_p
# simulates c iterations of fixed mutations counts arising by a Poisson-binomial process.
def mk_df_pb(chr_mu_dict,c):
    data_poisson2_p = chr_mut_pb(chr_mu_dict,c)
    dfas_p = pd.DataFrame.from_dict(data_poisson2_p)
    df2_p = dfas_p.applymap(binomialize) #Mimics segregation of each chromosome.
    df2_p["sum"]=df2_p.sum(axis=1)
    return df2_p
# Input parameters from glm.nb in R for the combined distribution of Dm and Mm
counts.
munb = 4.927 # rate parameter for negative binomial (munb)
theta = 60.42 # shape parameter
# Conversions
muc = math.exp(munb) #mu corrected (muc) is munb exponentiated.
gv = (muc)**2/theta #variance of the Gamma distribution of lambda
gscale = gv/muc #scale paramater of the Gamma distribution of lambda.
hap_mu = muc/4 #rate of fixed mutations per haploid genome.
print("The haploid rate of fixed mutations is", hap_mu)
n = 1000 # number of iterations to perform.
#Counts the number of unmasked nucleotides on each chr in the masked haploid yeast
genome
chr_list_a = [] #(list_a and list_b for two sets of chrs in 2n cell)
chr_list_b = []
chr_lenlist = []
for chrfile in glob.glob(r"G:\My Drive\Volatility File Share\YeastGenome\*_rm.*"):
    chr_rm = chrfile.split(".")[4] #gets the chr number from the fasta file name.
    chr_list_a.append(chr_rm + "a")
    chr_list_b.append(chr_rm + "b")
```

```
    with open(chrfile, 'r') as f:
        firstline = f.readline()
        count = int(firstline.split(":")[5]) #gets chr length from header of fasta file
        for line in f:  #counts "N"s in the fasta file and subtracts them from chr length.
            for i in line:
                if i == "N":
                    count = count - 1
    chr_lenlist.append(count)
tot_bp = sum(chr_lenlist)
#Generates Poisson and Poisson-binomial distributions.
mu=hap_mu
Dlist=[]
D2list=[]
D3list=[]
chr_mu_dict = mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
df_p = mk_df_p(chr_mu_dict,n)
df2_p = mk_df_pb(chr_mu_dict,n)
print("Index of Dispersion for Poisson, n=",n,",",index_dis(df_p))
print("Index of Dispersion for Poisson-binomial, n=",n,",",index_dis(df2_p))

#Generates gamma-Poisson and gamma-Poisson-binomial distributions.
"""The following returns a gamma distribution of rates of mismatch formation
and then divides each value by 4, since we're looking at the number of fixed mutations
per haploid genome. This distribution will be our input for n divisions,
each using a different lambda. Note that chr_mut_pb takes these values and doubles
them
to account for the actual rate of mispair formation. The simulated counts for each
chromosome
then become "binomialized" to account for segregation."""
data_gamma = (gamma.rvs(a=theta, scale=gscale, size=n))/4
df_nb = pd.DataFrame(columns=chr_list_a+chr_list_b)
dfas_nb = pd.DataFrame(columns=chr_list_a+chr_list_b)
for g in data_gamma:
    chr_mu_dict = mk_chr_mu_dict(g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    data_poisson = chr_mut_p(chr_mu_dict, 1)
    data_poisson2 = chr_mut_pb(chr_mu_dict, 1)
    dfa = pd.DataFrame.from_dict(data_poisson)
    df_nb = df_nb.append(dfa, ignore_index=True)
    dfa2 = pd.DataFrame.from_dict(data_poisson2)
    dfas_nb = dfas_nb.append(dfa2, ignore_index=True)
df2_nb = dfas_nb.applymap(binomialize)
df_nb["sum"] = df_nb.sum(axis=1)
df2_nb["sum"] = df2_nb.sum(axis=1)
```

```
print("Index of Dispersion for gamma-Poisson, n=",n,",",index_dis(df_nb))
print("Index of Dispersion for gamma-Poisson-binomial, n=",n,",",index_dis(df2_nb))
#simulates variation in index of dispersion given a small sample size
counter = 0
sample = 200
while counter < n:
    df_ps = mk_df_p(chr_mu_dict,sample)
    df2_ps = mk_df_pb(chr_mu_dict,sample)
    Dlist.append(index_dis(df_ps))
    D2list.append(index_dis(df2_ps))
    data_gamma = (gamma.rvs(a=theta, scale=gscale, size=sample))/4
    dfas_nbs = pd.DataFrame(columns=chr_list_a+chr_list_b)
    for g in data_gamma:
        chr_mu_dict = mk_chr_mu_dict(g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
        data_poisson2 = chr_mut_pb(chr_mu_dict, 1)
        dfa2 = pd.DataFrame.from_dict(data_poisson2)
        dfas_nbs = dfas_nbs.append(dfa2, ignore_index=True)
        df2_nbs = dfas_nbs.applymap(binomialize)
        df2_nbs["sum"] = df2_nbs.sum(axis=1)
    D3list.append(index_dis(df2_nbs))
    counter += 1
print("The mean index of dispersion for Poisson, n=",sample,",", mean(Dlist),"+/-
",stdev(Dlist))
print("The mean index of dispersion for Poisson-binomial, n=",
sample,",",mean(D2list),"+/-",stdev(D2list))
print("The mean index of dispersion for gamma-Poisson-binomial, n=",
sample,",",mean(D3list),"+/-",stdev(D3list))
sns.set(style="white", palette="bright", color_codes=True, rc={"lines.linewidth": 1.0})
plt.rcParams['patch.linewidth'] = 0
# Plots Poisson, Poisson-binomial, and gamma-Poisson-binomial distributions.
fig1, ax = plt.subplots(figsize=(2.5, 2.5))
sns.distplot(df_p['sum'],
             bins=list(range(20,120,2)),
         hist = False,
#              kde=False,
             norm_hist=True,
             color='gray')
sns.distplot(df2_p['sum'],
             bins=list(range(20,120,2)),
         hist = False,
#              kde=False,
             norm_hist=True,
             color='orange')
```

```
# sns.distplot(df_nb['sum'],
#                  bins=list(range(20,120,2)),
#              hist = False,
# #                   kde=False,
#                  norm_hist=True,
#                  color='green')
sns.distplot(df2_nb['sum'],
                 bins=list(range(20,120,2)),
            hist = False,
#                  kde=False,
                 norm_hist=True,
                 color='green')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlim=(0, 150))
ax.set(xlabel='Mutations/Division', ylabel='Density')
fig1.savefig(f"{File_save_location}/CombinedModelalt_ye2n529v2.pdf",transparent =
True,bbox_inches='tight')

#Plots the distributions of the indices of dispersion from Poisson, Poisson-binomial, and
gamma-Poisson-binomial simulations.
fig2, ax = plt.subplots(figsize=(2.5, 2.5))

sns.distplot(Dlist,
               bins=[i for i in np.arange(0.25,6,0.02)],
                 kde=False, norm_hist=True,
                 color='gray')
ax = sns.distplot(D2list,
                 bins=[i for i in np.arange(0.25,6,0.02)],
                 kde=False, norm_hist=True,
                 color='orange')
ax = sns.distplot(D3list,
                 bins=[i for i in np.arange(0.25,6,0.02)],
                 kde=False, norm_hist=True,
                 color='green')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlabel='Index of Dispersion', ylabel='Density')
fig2.savefig(f"{File_save_location}/IDmut_ye2n529.pdf",transparent =
True,bbox_inches='tight')
```

Fig3ghij.py

```
# For Fig.3(g-j), Dowsett et al. Revised version uploaded to Github on 11.12.20.
import glob
```

```
import pandas as pd
import numpy as np
from collections import defaultdict
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import poisson
from scipy.stats import nbinom
from scipy.stats import gamma
import random
import math
from statistics import mean
from statistics import stdev
# Introduction
"""The following Python 3 code simulates the extent to which asymmetric segregation of
mutations lead to overdispersion in human ultramutator cancers assuming that a single
Poisson
process governs the mutator phenotype in all cell divisions. To illustrate this concept,
we extrapolated the haploid genome mutation rate observed in yeast to humans."""

# Output Files(s) folder:
from pathlib import Path
File_save_location  = Path.cwd()
# Defined functions
"""chr_mu_dict is a dictionary where the keys represent chromosomes in a diploid
cell and the values, their associated mutation rates. This assumes each chr
contributes proportionately to the genome-wide mutation rate."""
def mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b):
    ch_mu = []
    for i in chr_lenlist:
        ch_mu.append(mu*(i/tot_bp))
    tuplesA=list(zip(chr_list_a,ch_mu))
    tuplesB=list(zip(chr_list_b,ch_mu))
    chr_mu_dict = {}
    for i in tuplesA:
        chr_mu_dict[i[0]]=i[1]
    for i in tuplesB:
        chr_mu_dict[i[0]]=i[1]
    return chr_mu_dict
# creates dict where keys are chrs, and values are arrays (length c) of fixed muts from a
Poisson.
def chr_mut_p(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=v, size=c)) for (k,v) in chr_mu_dict.items()}
```

```
# as above, except rate is doubled to represent actual rate of mispair formation.
def chr_mut_pb(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=(v*2), size=c)) for (k,v) in chr_mu_dict.items()}
# multiplies value by randomly selected 0 or 1 to mimic mitotic segregation.
def binomialize(x):
    return x * (random.randint(0,1))
# returns index of dispersion for a distribution
def index_dis(df):
    return ((df["sum"].std())**2)/df["sum"].mean()
# simulates c iterations of fixed mutations counts arising by a Poisson process.
def mk_df_p(chr_mu_dict,c):
    data_poisson_p = chr_mut_p(chr_mu_dict,c)
    df_p = pd.DataFrame.from_dict(data_poisson_p)
    df_p["sum"]=df_p.sum(axis=1)
    return df_p
# simulates c iterations of fixed mutations counts arising by a Poisson-binomial process.
def mk_df_pb(chr_mu_dict,c):
    data_poisson2_p = chr_mut_pb(chr_mu_dict,c)
    dfas_p = pd.DataFrame.from_dict(data_poisson2_p)
    df2_p = dfas_p.applymap(binomialize) #Mimics segregation of each chromosome.
    df2_p["sum"]=df2_p.sum(axis=1)
    return df2_p
# Extrapolating yeast mutation rates to human ultramutator cells.
def ye2hu_mr(yeast_1n_mu):
    hu_mu = (tot_bp/11e7)*yeast_1n_mu
    return hu_mu

def linregparams (df):
    x = df["rate"]
    y = df["ID"]
    slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
    print("slope: %f    intercept: %f    p_value: %f    std_err: %f" % (slope, intercept,
p_value, std_err))
    print("R-squared: %f" % r_value**2)
# Input parameters from glm.nb in R for the combined distribution of Dm and Mm
counts.
munb = 4.927 # rate parameter for negative binomial (munb)
theta = 60.42 # shape parameter
# Conversions
muc = math.exp(munb) #mu corrected (muc) is munb exponentiated.
gv = (muc)**2/theta #variance of the Gamma distribution of lambda
gscale = gv/muc #scale paramater of the Gamma distribution of lambda.
hap_mu = muc/4 #rate of fixed mutations per haploid genome.
```

```
n = 10000 # desired number of iterations to perform.
yeast_1n_mu = 34.5 #mutations/divisions/genome
#Gathers the number of nucleotides on each chr in human genome (a and b for 2n cell)
chr_list_a = []
chr_list_b = []
chr_lenlist = []
for chrfile in glob.glob(r"G:\My Drive\Volatility File Share\HumanGenome\*.fa"):
    chr_rm = chrfile.split(".")[4] #gets the chr number from the fasta file name.
    chr_list_a.append(chr_rm + "a")
    chr_list_b.append(chr_rm + "b")
    with open(chrfile, 'r') as f:
        firstline = f.readline()
        count = int(firstline.split(":")[5]) #gets chr length from header of fasta file
    chr_lenlist.append(count)
tot_bp = sum(chr_lenlist)
#Generates initial Poisson and Poisson-binomial distributions for comparisons.
mu=ye2hu_mr(34.5)
counter = 0
Dlist=[]
D2list=[]
chr_mu_dict = mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
df_p = mk_df_p(chr_mu_dict,n)
df_pb = mk_df_pb(chr_mu_dict,n)
print("Index of Dispersion for Poisson, n=",n,",",index_dis(df_p))
print("Index of Dispersion for Poisson-binomial, n=",n,",",index_dis(df_pb))
#Generates gamma-Poisson-binomial distribution.
"""The following returns a gamma distribution of rates of mismatch formation
and then divides each value by 4, since we're looking at the number of fixed mutations
per haploid genome. This distribution will be our input for n divisions,
each using a different lambda. Note that chr_mut_pb takes these values and doubles
them
to account for the actual rate of mispair formation.  This then becomes halved when we
binomialize it to account for segregation."""
data_gamma = (gamma.rvs(a=theta, scale=gscale, size=n))/4
dfas_nb = pd.DataFrame(columns=chr_list_a+chr_list_b)
for g in data_gamma:
    hu_g = ye2hu_mr(g)
    chr_mu_dict = mk_chr_mu_dict(hu_g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    data_poisson2 = chr_mut_pb(chr_mu_dict, 1)
    dfa2 = pd.DataFrame.from_dict(data_poisson2)
    dfas_nb = dfas_nb.append(dfa2, ignore_index=True)
df2_nb = dfas_nb.applymap(binomialize)
df2_nb["sum"] = df2_nb.sum(axis=1)
```

```
print("Index of Dispersion for gamma-Poisson-binomial, n=",n,",",index_dis(df2_nb))
#Simulates how dispersion due to the Poisson-binomial process varies as a function of mutation rate.
hu_mu_rng=list(range(1,951))
Drlist=[]
D2rlist=[]
for i in hu_mu_rng:
    mu=i
    chr_mu_dict = mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    df_p1 = mk_df_p(chr_mu_dict,n)
    df_pb1 = mk_df_pb(chr_mu_dict,n)
    Drlist.append(index_dis(df_p1))
    D2rlist.append(index_dis(df_pb1))
Df_IDrng_p=pd.DataFrame()
Df_IDrng_pb=pd.DataFrame()
Df_IDrng_p["rate"]=hu_mu_rng
Df_IDrng_p["ID"]=Drlist
Df_IDrng_pb["rate"]=hu_mu_rng
Df_IDrng_pb["ID"]=D2rlist
Df_IDrng_pb.to_csv(f'G:\My Drive\Volatility File Share\Figures\IDrng_p.csv",index=False)
Df_IDrng_pb.to_csv(f'G:\My Drive\Volatility File Share\Figures\IDrng_pb.csv",index=False)
print("Linear regression of relationship between MR and ID with pois model:")
linregparams(Df_IDrng_p)
print("Linear regression of relationship between MR and ID with poisbin model:")
linregparams(Df_IDrng_pb)
# Simulates the change in mutation burden in individual tumor cell lineages over 30 divisions.
n = 1000
counter = 0
plist=[] #
pblist=[]
dfRT=pd.DataFrame() #To keep track of cumulative mutation burden during a Poisson process.
dfRT["Div"]=list(range(1,31,1))
df2RT=pd.DataFrame() #To keep track of cumulative mutation burden during a Poisson-binomial process.
df2RT["Div"]=list(range(1,31,1))
while counter < n:
    df_p_line = mk_df_p(chr_mu_dict,30)
    dfRT[counter+1]=df_p_line["sum"].cumsum()
    mutbur_p=df_p_line['sum'].sum()
```

```
    plist.append(mutbur_p)
    df_pb_line = mk_df_pb(chr_mu_dict,30)
    df2RT[counter+1]=df_pb_line["sum"].cumsum()
    mutbur_pb=df_pb_line['sum'].sum()
    pblist.append(mutbur_pb)
    counter += 1
print("The average mutation burden for Poisson after 30 div is",mean(plist),"+/-
",stdev(plist))
print("The average mutation burden for Poisson-binomial after 30 div
is",mean(pblist),"+/-",stdev(pblist))
print("Index of Dispersion for Poisson after 30 div, n=",n,",",(stdev(plist))**2/mean(plist))
print("Index of Dispersion for Poisson-binomial after 30 div,
n=",n,",",(stdev(pblist))**2/mean(pblist))
# Introduces a row of zeros into each dataframe to represent the starting point for each
lineage.
dfRT.loc[len(dfRT)]=0
df2RT.loc[len(dfRT)]=0
dfRTm=pd.melt(dfRT,id_vars=['Div'],value_vars=list(dfRT.columns)[1:],var_name='line',
value_name='Total Mut')
df2RTm=pd.melt(df2RT,id_vars=['Div'],value_vars=list(df2RT.columns)[1:],var_name='li
ne', value_name='Total Mut')
sns.set(style="white", palette="bright", color_codes=True, rc={"lines.linewidth": 1.0})
plt.rcParams['patch.linewidth'] = 0
# Plots Poisson, Poisson-binomial, and gamma-Poisson-binomial distributions after 1
division.
fig1, ax = plt.subplots(figsize=(2, 2.5))
sns.distplot(df_p["sum"],
             bins=list(range(900,3000,10)),
            hist=False,
#                kde=False,
             norm_hist=True,
             color='gray')
sns.distplot(df_pb["sum"],
             bins=list(range(900,3000,10)),
          hist=False,
#                kde=False,
             norm_hist=True,
             color='orange')
sns.distplot(df2_nb["sum"],
             bins=list(range(900,3000,10)),
          hist=False,
#                kde=False,
             norm_hist=True,
```

```
                color='green')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlabel='Mutations/Division', ylabel='Density')
fig1.savefig(f"{File_save_location}/ASmut_hu530a.pdf",transparent =
True,bbox_inches='tight')
# Plots Poisson vs Poisson-binomial distributions after 30 divisions.
fig2, ax = plt.subplots(figsize=(2, 2.5))
sns.distplot(plist,
                bins=list(range(51000,63000,200)),
                kde=False,
                norm_hist=True,
                color='gray')
sns.distplot(pblist,
                bins=list(range(51000,63000,200)),
                kde=False,
                norm_hist=True,
                color='orange')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlabel='Mutation Burden', ylabel='Density')
fig2.savefig(f"{File_save_location}/MB_hu30a.pdf",transparent =
True,bbox_inches='tight')
# Plots Poisson vs Poisson-binomial mutation accumulation in individual lines over 10
divisions.
fig3, ax = plt.subplots(figsize=(2,2.5))
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlim=(0, 10),ylim=(0,25000))
sns.lineplot(x="Div", y="Total Mut", hue='line', palette=sns.color_palette("Set1", n),
legend=False,lw=0.5,data=df2RTm)
sns.lineplot(x="Div", y="Total Mut",legend=False, lw=1,color="k",data=dfRTm)
fig3.savefig(f"{File_save_location}/MB_huMutAc1.pdf",transparent =
True,bbox_inches='tight')
# Plots changes in the index of dispersion as a function of mutation rate.
fig4, ax = plt.subplots(figsize=(2, 2.5))
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlim=(0, 1000),ylim=(0,55))
sns.regplot(x="rate", y="ID", data=Df_IDrng_p, color="gray",marker='o',
scatter_kws={'s':1})
sns.regplot(x="rate", y="ID", data=Df_IDrng_pb, color="orange",marker='o',
scatter_kws={'s':1})
fig4.savefig(f"{File_save_location}/MR_IDcorr1.pdf",transparent =
True,bbox_inches='tight')
```

ExFig3a.py

```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from Bio import SeqIO
from matplotlib import collections  as mc
import pylab as pl
figsize = (16, 6)
fig, ax = plt.subplots(figsize=figsize)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.yticks(range (1,17))
plt.xlim(0, 1600000)
plt.ylim(0, 17)
ax.set_ylabel('Chromosome', fontname="Arial", fontsize=12)
ax.set_xlabel('Basepair Position', fontname="Arial", fontsize=12)
ax.set_title('Mutations Scored Over the Genome', fontname="Arial", fontsize=16)
# Blank Chromosomes Construction
lengths = [230218,813184, 316620,        1531933,576874,
          270161,1090940,562643,439888,745751,666816,1078177,924431,784333,1091
291,948066]
output = []
for i in range(0, 16):
    output.append([  (0, i+1), (lengths[i], i+1)    ])
c = "blue"
lc = mc.LineCollection([i for i in output], colors=c
, linewidths=2)
ax.add_collection(lc)
# Chromosome key lookup (For mutations and Masked genome construction)
dchrom = {'NC_001133' : 1, 'NC_001134' : 2, 'NC_001135' : 3, 'NC_001136' : 4,
'NC_001137' : 5 , 'NC_001138' : 6,
'NC_001139' : 7, 'NC_001140' : 8, 'NC_001141' : 9, 'NC_001142' : 10, 'NC_001143' :
11, 'NC_001144' : 12,
'NC_001145' : 13, 'NC_001146' : 14, 'NC_001147' : 15, 'NC_001148' : 16, 'NC_001224'
: 'M'}
# Generate a means of looking up a position along a chromosome to see if it's masked
('N'), requires the referenced file: S288C-masked-genome.fasta
# This section builds the yeast masked genome positions ('N') as a list (N_list) in the
form (Chromosome, position) of all masked positions
genome_dict, record_dict = {}, {}
for record in SeqIO.parse("S288C-masked-genome.fasta", "fasta"):
```

```
    record_dict[record.id] = record.seq
position = 0
N_list = []
for key,value in record_dict.items():
    position = 0
    for i in value:
        if i == 'N' and key != 'Mito':
            N_list.append((dchrom[key.split('|')[1]], position))
        position += 1
color = 'grey'
mark = '|'
for i in range (1,17):
    ax.scatter([x[1] for x in N_list if x[0] == i], ([i for x in N_list if x[0] == i]), s=None,
c=color, marker=mark, label="Masked Base" if i ==1 else "", alpha=0.2,zorder=2)
# Mutation dataframe construction and graphing
def ExtractPosition(mutations):
    Position    =  int( mutations.split(':')[1])
    Chromosome  =   mutations.split('|')[1]
    return (Position)
# Generate a list of mutations:
mutation_file = 'FullMutationList.txt'
with open(mutation_file, 'r') as rr:
    mutation_lines  = [i.split('\t') for i in rr]
Chromosome      = [dchrom[i[3].split('|')[1]] for i in mutation_lines]
mutations = pd.DataFrame(mutation_lines, columns = ['Lineage', 'SubDivion', 'Colonies',
'Position', 'Segregant', 'DivisionID', 'Mutation', 'GenomicPosition'])
mutations['Position'] = mutations['Position'].apply(ExtractPosition)
mutations.insert( 4, 'chrom', Chromosome)
color = 'green'
mark = '|'
for i in range (1,17):
    ax.scatter([x.Position for x in mutations.itertuples() if x.chrom == i], ([i for x in
mutations.itertuples() if x.chrom == i]), s=None, c=color, marker=mark, label="Mutation"
if i ==1 else "", alpha=0.8,zorder=3)




# Optional Centromere addition:
cent_file = 'centromeres.txt'
with open(cent_file, 'r') as cent:
    cent_lines = [i.split('\t') for i in cent]
```

```
cents = [(int(i[2].split('..')[0])+int(i[2].split('..')[1]))/2 for i in cent_lines]
for i in range (1,17):
    ax.scatter(cents[i-1],i, s=20, c='red', marker='o', label="Centromere" if i ==1 else "",
alpha=0.8, zorder=4)
ax.legend()
plt.show()
```

ExFig4a.py

```
#This script creates x,y plots of each lineage and number of observed mutations for
each division
data = {'division':[2,3,4,6,7,8,10,11,12,13,14,15,2,3,4,5,6,7,8,10,11,12,2,3,
4,5,6,7,2,3,5,6,8,3,4,6,8,10,2,3,4,5,7,8,11,12,13,14,15,16],
#
'mutations':[248,200,255,328,288,279,285,231,256,251,262,301,255,287,298,270,277,3
16,308,356,307,306,270,289,250,303,
#
155,334,231,236,298,302,283,117,323,312,288,295,335,252,282,319,296,284,254,224,
253,264,280,232,219]}
'mutations' : [248     ,200,   255     ,328,   288,    278,
284 ,228,       256     ,251,   262,    301,
253     ,288,   297     ,270,   278,    315     ,306,   356,    307,    306,
269     ,288,   250     ,303,   155,    334,
229     ,235,   298     ,302,   283,
323,    312     ,288,   288,    332,
252     ,283,   319     ,298,   281,    254,    224,    252,    263,    280,    232,    219]}
l151x = [2, 3, 4, 6, 7, 8]
l151y = [248,200,255,328,288,278]
l153x = [10, 11, 12, 13, 14, 15]
l153y = [284,228,256,251,262,301]
l156x = [2, 3, 4, 5, 6, 7, 8, 10, 11, 12]
l156y = [253,288,297,270,278,315,306,356,307,306]
l157x = [2, 3, 4, 5, 6, 7]
l157y = [269,288,250,303,155,334]
l158x = [2, 3, 5, 6, 8]
l158y = [229,235,298,302,283]
l160x = [3, 4, 6, 8, 10]
l160y = [323,312,288,288,332]
l162x = [2, 3, 4, 5, 7, 8, 11, 12, 13, 14, 15, 16]
l162y = [252,283,319,298,281,254,224,252,263,280,232,219]
import pandas as pd
df = pd.DataFrame(data, columns = ['division','mutations'])
from scipy import stats
import plotly
```

```
import plotly.graph_objs as go
import plotly.figure_factory as ff
spearman_coef, ps_value = stats.spearmanr(df["division"], df["mutations"])
print("Spearman Correlation Coefficient: ", spearman_coef, "and a P-value of:",
ps_value)
# Plotly mapping of relationship, all lineages seperately
# Title = "Correlation Between Age of Mother and Number of Mutations <br> Pearson
Correlation Coefficient: " + str(round(pearson_coef,5)) + "\n, p = " + str(round(p_value,
3))
Title = "All lineages"
lin151 = go.Scatter(
    x = l151x,    y = l151y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=20,        symbol='circle',        color= 'black',        opacity=1 ))
lin153 = go.Scatter(
    x = l153x,    y = l153y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=8,        symbol='circle',        color= 'hotpink',        opacity=1 ))
lin156 = go.Scatter(
    x = l156x,    y = l156y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=8,        symbol='circle',        color= 'green',        opacity=1 ))
lin157 = go.Scatter(
    x = l157x,    y = l157y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=8,        symbol='circle',        color= 'blue',        opacity=1 ))
lin158 = go.Scatter(
    x = l158x,    y = l158y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=8,        symbol='circle',        color= 'orange',        opacity=1 ))
lin160 = go.Scatter(
    x = l160x,    y = l160y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=8,        symbol='circle',        color= 'purple',        opacity=1 ))
lin162 = go.Scatter(
    x = l162x,    y = l162y,    mode = 'lines',    line = dict(width = 6), marker=dict(
size=8,        symbol='circle',        color= 'red',        opacity=1 ))
data1 = [lin151,lin153,lin156,lin157,lin158,lin160,lin162]
layout = go.Layout(
    title=Title,
    font=dict(family='Arial, sans-serif', size=10, color='black'),
    yaxis=dict(title='Mutations', autorange=False, range=[0, 400],
titlefont=dict(family='Arial, sans-serif', size=14, color='black')),
    xaxis=dict(title='Division Number', autorange=False, range=[1, 17],
titlefont=dict(family='Arial, sans-serif', size=14, color='black')),
    autosize=False,
    width=700,
    height=700,
```

```
    margin=dict(l=50, r=10, b=50, t=90, pad=4), paper_bgcolor='#ffffff',
plot_bgcolor='#ffffff')
# layout = go.Layout(title = Title)
fig = go.Figure(data=data1, layout=layout)
plotly.offline.plot(fig, auto_open=True)
```

ExFig5.py

```
# For Extended Data Fig.5, Dowsett et al.
import glob
import pandas as pd
import numpy as np
from collections import defaultdict
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import poisson
from scipy.stats import nbinom
from scipy.stats import gamma
import random
import math
from statistics import mean
from statistics import stdev
# Introduction
"""The following Python 3 code simulates the extent to which the mutation counts in
pairs of segregant groups (Da vs Db and Ma vs Mb) are correlated."""
# Output Files(s) folder:
from pathlib import Path
File_save_location  = Path.cwd()
# Defined functions
"""chr_mu_dict is a dictionary where the keys represent chromosomes in a diploid
cell and the values, their associated mutation rates. This assumes each chr
contributes proportionately to the genome-wide mutation rate."""
def mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a,chr_list_b):
    ch_mu = []
    for i in chr_lenlist:
        ch_mu.append(mu*(i/tot_bp))
    tuplesA=list(zip(chr_list_a,ch_mu))
    tuplesB=list(zip(chr_list_b,ch_mu))
    chr_mu_dict = {}
    for i in tuplesA:
        chr_mu_dict[i[0]]=i[1]
    for i in tuplesB:
```

```
        chr_mu_dict[i[0]]=i[1]
    return chr_mu_dict
# creates dict where keys are chrs, and values are arrays (length c) of fixed muts from a
Poisson.
def chr_mut_p(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=v, size=c)) for (k,v) in chr_mu_dict.items()}
# as above, except rate is doubled to represent actual rate of mispair formation.
def chr_mut_bp(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=(v*2), size=c)) for (k,v) in chr_mu_dict.items()}
# multiplies value by randomly selected 0 or 1 to mimic Mendelian segregation.
def binomialize(x):
    return x * (random.randint(0,1))
# returns index of dispersion for a distribution
def index_dis(df):
    return ((df["sum"].std())**2)/df["sum"].mean()
# simulates c iterations of fixed mutations counts arising by a Poisson process.
def mk_df_p(chr_mu_dict,c):
    data_poisson_p = chr_mut_p(chr_mu_dict,c)
    df_p = pd.DataFrame.from_dict(data_poisson_p)
    df_p["sum"]=df_p.sum(axis=1)
    return df_p
# simulates c iterations of fixed mutations counts arising and segregating to two cells.
def mk_df_bp(chr_mu_dict,c):
    data_poisson2_p = chr_mut_bp(chr_mu_dict,c)
    dfas_p = pd.DataFrame.from_dict(data_poisson2_p)
    df_binomialize(dfas_p) #produces two dfs of mutation counts from reciprocol
segregants
    df2_cell1["sum"]=df2_cell1.sum(axis=1)
    add_chr_totals(df2_cell1)
    df2_cell2["sum"]=df2_cell2.sum(axis=1)
    add_chr_totals(df2_cell2)
def df_binomialize(df_input):
    df_input_bintem = (df_input * 0) + 1 #makes a dataframe of 1's w/dimensions of
df_input
    df_input_binom = df_input_bintem.applymap(binomialize) #Randomly changes 1's to
0 for cell1
#     print(df_input_binom)
    df_input_binom_inv=abs(df_input_binom - 1) #Inverse of dfas_p_binom for cell2
#     print(df_input_binom_inv)
    df2_cell1 = df_input.mul(df_input_binom) #Multiplies mutations in cell1 by 1 or 0 to
mimic segregation.
    df2_cell2 = df_input.mul(df_input_binom_inv) #Mimics mutations in cell2 the inverse
to mimic segregation.
```

```
    df2_cell1["sum"] = df2_cell1.sum(axis=1)
    add_chr_totals(df2_cell1)
    df2_cell2["sum"] = df2_cell2.sum(axis=1)
    add_chr_totals(df2_cell2)
    return df2_cell1,df2_cell2
def add_chr_totals(df_cell): #adds muts on a and b sets of chromosomes to get
observed mut/chr.
    df_cell["I"] = df_cell["Ia"] + df_cell["Ib"]
    df_cell["II"] = df_cell["IIa"] + df_cell["IIb"]
    df_cell["III"] = df_cell["IIIa"] + df_cell["IIIb"]
    df_cell["IV"] = df_cell["IVa"] + df_cell["IVb"]
    df_cell["V"] = df_cell["Va"] + df_cell["Vb"]
    df_cell["VI"] = df_cell["VIa"] + df_cell["VIb"]
    df_cell["VII"] = df_cell["VIIa"] + df_cell["VIIb"]
    df_cell["VIII"] = df_cell["VIIIa"] + df_cell["VIIIb"]
    df_cell["IX"] = df_cell["IXa"] + df_cell["IXb"]
    df_cell["X"] = df_cell["Xa"] + df_cell["Xb"]
    df_cell["XI"] = df_cell["XIa"] + df_cell["XIb"]
    df_cell["XII"] = df_cell["XIIa"] + df_cell["XIIb"]
    df_cell["XIII"] = df_cell["XIIIa"] + df_cell["XIIIb"]
    df_cell["XIV"] = df_cell["XIVa"] + df_cell["XIVb"]
    df_cell["XV"] = df_cell["XVa"] + df_cell["XVb"]
    df_cell["XVI"] = df_cell["XVIa"] + df_cell["XVIb"]
# Input parameters from glm.nb in R for the combined distribution of Dm and Mm
counts.
munb = 4.927 # rate parameter for negative binomial (munb)
theta = 60.42 # shape parameter
# Conversions
muc = math.exp(munb) #mu corrected (muc) is munb exponentiated.
gv = (muc)**2/theta #variance of the Gamma distribution of lambda
gscale = gv/muc #scale paramater of the Gamma distribution of lambda.
hap_mu = muc/4 #rate of fixed mutations per haploid genome.
print("The haploid rate of fixed mutations is", hap_mu)
n = 100 # number of iterations to perform.
#Counts the number of unmasked nucleotides on each chr in the masked haploid yeast
genome
chr_list_a = [] #(list_a and list_b for two sets of chrs in 2n cell)
chr_list_b = []
chr_list = []
chr_lenlist = []
for chrfile in glob.glob(r"G:\My Drive\Volatility File Share\YeastGenome\*_rm.*"):
    chr_rm = chrfile.split(".")[4] #gets the chr number from the fasta file name.
    chr_list_a.append(chr_rm + "a")
```

```
    chr_list_b.append(chr_rm + "b")
    chr_list.append(chr_rm)
    with open(chrfile, 'r') as f:
        firstline = f.readline()
        count = int(firstline.split(":")[5]) #gets chr length from header of fasta file
        for line in f:  #counts "N"s in the fasta file and subtracts them from chr length.
            for i in line:
                if i == "N":
                    count = count - 1
    chr_lenlist.append(count)
tot_bp = sum(chr_lenlist)
#Models the occurrance and segregation of fixed mutations in pairs of segregating cells.
"""The following returns a gamma distribution of rates of mismatch formation
and then divides each value by 4, since we're looking at the number of fixed mutations
per haploid genome. This distribution will be our input for n divisions,
each using a different lambda. It then takes the dataframe of mismatch counts and
generates
the reciprocal mutation burdens expected in the resulting pairs of segregant cells."""
data_gamma = (gamma.rvs(a=theta, scale=gscale, size=n))/4
df_nb = pd.DataFrame(columns=chr_list_a+chr_list_b)
dfas_nbD = pd.DataFrame(columns=chr_list_a+chr_list_b) #D stands for Daughter cell
dfas_nbM = pd.DataFrame(columns=chr_list_a+chr_list_b) #M stands for Mother cell
for g in data_gamma: #with the same lambda we're going to call mutations twice and
compare
    chr_mu_dictD = mk_chr_mu_dict(g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    chr_mu_dictM = mk_chr_mu_dict(g,chr_lenlist,tot_bp,chr_list_a,chr_list_b)
    data_poisson2D = chr_mut_bp(chr_mu_dictD, 1)
    data_poisson2M = chr_mut_bp(chr_mu_dictM, 1)
    dfa2D = pd.DataFrame.from_dict(data_poisson2D)
    dfa2M = pd.DataFrame.from_dict(data_poisson2M)
    dfas_nbD = dfas_nbD.append(dfa2D, ignore_index=True)
    dfas_nbM = dfas_nbM.append(dfa2M, ignore_index=True)
df2_bnb_cell1D,df2_bnb_cell2D = df_binomialize(dfas_nbD)
df2_bnb_cell1M,df2_bnb_cell2M = df_binomialize(dfas_nbM)
df2totalbnbDsums=pd.DataFrame()
df2totalbnbMsums=pd.DataFrame()
df2totalbnbDsums["Segregant 1"]=df2_bnb_cell1D["sum"]
df2totalbnbDsums["Segregant 2"]=df2_bnb_cell2D["sum"]
df2totalbnbMsums["Segregant 1"]=df2_bnb_cell1M["sum"]
df2totalbnbMsums["Segregant 2"]=df2_bnb_cell2M["sum"]
df2totalbnbDsums
"""Creates data frame of actual data (Da v Db and Ma v Mb) for comparison."""
df_data = pd.read_csv("G:\My Drive\Volatility File Share\Data\MutInSegGroups.csv")
```

"""Plots the linear regressions of the bnb model and the actual data."""
fig1, ax = plt.subplots(figsize=(2,2))
sns.regplot(x="Segregant 1", y="Segregant 2", data=df2totalbnbDsums, marker='o',
color="silver", scatter_kws={'s':7},line_kws={'color':"peru"})
sns.regplot(x="Db", y="Da", data=df_data, color="green",marker='o',
scatter_kws={'s':11})
sns.regplot(x="Ma", y="Mb", data=df_data, color="blue",marker='o',
scatter_kws={'s':11})
ax.set(ylim=(0, 150))
ax.set(xlim=(0, 150))
ax.set(xlabel='Segregant 1', ylabel='Segregant 2')
ax.tick_params(bottom=True,left=True, labelsize=8)
fig1.savefig(f"{File_save_location}/lineRegSegsnb100v2.pdf",transparent =
True,bbox_inches='tight')

ExFig6.py

```
# For Supplementary Fig 6, Dowsett et al. (pol2-4 msh6 haploid cells)
import glob
import pandas as pd
import numpy as np
from collections import defaultdict
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import poisson
from scipy.stats import nbinom
from scipy.stats import gamma
import random
import math
from statistics import mean
from statistics import stdev
# Introduction
"""The following Python 3 code simulates the extent to which asymmetric segregation of
mutations leads to overdispersion
in a haploid strain with a modest mutator phenotype compared to simple Poisson
process."""
# Output Files(s) folder:
from pathlib import Path
File_save_location  = Path.cwd()


# Defined functions
```

```
"""chr_mu_dict is a dictionary where the keys represent chromosomes in a diploid
cell and the values, their associated mutation rates. This assumes each chr
contributes proportionately to the genome-wide mutation rate."""
def mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a):
    ch_mu = []
    for i in chr_lenlist:
        ch_mu.append(mu*(i/tot_bp))
    tuplesA=list(zip(chr_list_a,ch_mu))
    chr_mu_dict = {}
    for i in tuplesA:
        chr_mu_dict[i[0]]=i[1]
    return chr_mu_dict
# creates dict where keys are chrs, and values are arrays (length c) of fixed muts from a
Poisson.
def chr_mut_p(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=v, size=c)) for (k,v) in chr_mu_dict.items()}
# as above, except rate is doubled to represent actual rate of mispair formation.
def chr_mut_bp(chr_mu_dict,c):
    return {k:(poisson.rvs(mu=(v*2), size=c)) for (k,v) in chr_mu_dict.items()}
# multiplies value by randomly selected 0 or 1 to mimic mitotic segregation.
def binomialize(x):
    return x * (random.randint(0,1))

# returns index of dispersion for a distribution
def index_dis(df):
    return ((df["sum"].std())**2)/df["sum"].mean()
# simulates c iterations of fixed mutations counts arising by a Poisson process.
def mk_df_p(chr_mu_dict,c):
    data_poisson_p = chr_mut_p(chr_mu_dict,c)
    df_p = pd.DataFrame.from_dict(data_poisson_p)
    df_p["sum"]=df_p.sum(axis=1)
    return df_p
# simulates c iterations of fixed mutations counts arising by a Poisson-binomial process.
def mk_df_bp(chr_mu_dict,c):
    data_poisson2_p = chr_mut_bp(chr_mu_dict,c)
    dfas_p = pd.DataFrame.from_dict(data_poisson2_p)
    df2_p = dfas_p.applymap(binomialize) #Mimics segregation of each chromosome.
    df2_p["sum"]=df2_p.sum(axis=1)
    return df2_p
# Conversions
hap_mu = 1.75 #rate of fixed mutations per haploid genome by WGS.
print("The haploid rate of fixed mutations is", hap_mu)
n = 10000 # number of iterations to perform.
```

```
#Counts the number of unmasked nucleotides on each chr in the masked haploid yeast
genome
chr_list_a = [] #(list_a for 1 set of chrs in 1n cell)
chr_lenlist = []
for chrfile in glob.glob(r"G:\My Drive\Volatility File Share\YeastGenome\*_rm.*"):
    chr_rm = chrfile.split(".")[4] #gets the chr number from the fasta file name.
    chr_list_a.append(chr_rm + "a")
    with open(chrfile, 'r') as f:
        firstline = f.readline()
        count = int(firstline.split(":")[5]) #gets chr length from header of fasta file
        for line in f:  #counts "N"s in the fasta file and subtracts them from chr length.
            for i in line:
                if i == "N":
                    count = count - 1
    chr_lenlist.append(count)
tot_bp = sum(chr_lenlist)
#Generates Poisson and Poisson-binomial distributions.
mu=hap_mu
Dlist=[]
D2list=[]
chr_mu_dict = mk_chr_mu_dict(mu,chr_lenlist,tot_bp,chr_list_a)
df_p = mk_df_p(chr_mu_dict,n)
df2_p = mk_df_bp(chr_mu_dict,n)
print("Index of Dispersion for Poisson, n=",n,",",index_dis(df_p))
print("Index of Dispersion for Poisson-binomial, n=",n,",",index_dis(df2_p))
#simulates variation in index of dispersion given a small sample size
counter = 0
sample = 176
while counter < n:
    df_ps = mk_df_p(chr_mu_dict,sample)
    df2_ps = mk_df_bp(chr_mu_dict,sample)
    Dlist.append(index_dis(df_ps))
    D2list.append(index_dis(df2_ps))
    counter += 1
print("The mean index of dispersion for Poisson, n=",sample,",", mean(Dlist),"+/-
",stdev(Dlist))
print("The mean index of dispersion for Poisson-binomial, n=",
sample,",",mean(D2list),"+/-",stdev(D2list))
sns.set(style="white", palette="bright", color_codes=True, rc={"lines.linewidth": 1.0})
plt.rcParams['patch.linewidth'] = 0
# Plots Poisson vs Poisson-binomial distributions.
fig1, ax = plt.subplots(figsize=(2.5, 2.5))
sns.distplot(df_p["sum"],
```

```
                bins=list(range(0,10,1)),
                kde=False,
                norm_hist=True,
                color='gray')
sns.distplot(df2_p["sum"],
                bins=list(range(0,10,1)),
                kde=False,
                norm_hist=True,
                color='orange')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlabel='Mutations/Division', ylabel='Density')
fig1.savefig(f"{File_save_location}/ASmut_ye1n.pdf",transparent =
True,bbox_inches='tight')
#Plots the distributions of the indices of dispersion from Poisson and Poisson-binomial
simulations.
fig2, ax = plt.subplots(figsize=(2.5, 2.5))
sns.distplot(Dlist,
              bins=[i for i in np.arange(0.5,1.5,0.02)],
                kde=False, norm_hist=True,
                color='gray')
ax = sns.distplot(D2list,
                bins=[i for i in np.arange(0.5,1.5,0.02)],
                kde=False, norm_hist=True,
                color='orange')
ax.tick_params(bottom=True,left=True,labelsize=10)
ax.set(xlabel='Index of Dispersion', ylabel='Density')
fig2.savefig(f"{File_save_location}/IDmut_ye1n.pdf",transparent =
True,bbox_inches='tight')
```

Chapter Four Scripts

LOHsearch.py

```
#The below script generates a text document from a set of .noSNP.var files
#representing the initial, FY, and aged pairs of sequenced colonies from a given
#lineage. It can also interpret a list of such files.
import glob
import pandas as pd
from argparse import ArgumentParser
def Frameit(name, num, form): #Return the paired dataframes for the mother and
daughters
    file = (f"{name}{num}{form}.noSNP.var")
    with open(file,'r') as f:
        part1 = [i for i in f]
```

```
        col = part1[0].split('\n')[0].split('\t')
        part1 = part1[1:]
        dat = pd.DataFrame([i.split('\t')[:5] for i in part1], columns=col[:5])
    name = str(file)
    print(f"framing up {name}")
    dat.insert(0, "Strain", name, True)
    new = dat['Cons:Cov:Reads1:Reads2:Freq:P-value'].str.split(":", n = 5, expand =
True) # new data frame with split value columns
    adding_names = col[4].split(":")
    for i in range(0,len(adding_names)): # making separate column from each new data
frame (and looping through)
        dat[adding_names[i]] = new[i]
    dat.drop(columns =[col[4]], inplace = True) # Dropping old column
    if form == 'd':
        return(dat)
    elif form == '':
        return(dat)
def Unify(M,D): #converts the data frames to a unified dictionary of position : (Mfreq,
pvalue)(Dfreq, pvalue)
    Md, Dd, MDd = {}, {}, {}
    for row in M.itertuples():
        chrom = row[2]
        position = row[3]
        Md[chrom + ':' + position] = (float(row[10].split('%')[0]),row[11])
    for row in D.itertuples():
        chrom = row[2]
        position = row[3]
        Dd[chrom + ':' + position] = (float(row[10].split('%')[0]),row[11])
    for key, value in Md.items():
        if key in Dd:
            MDd[key] = ((value),(Dd[key]))
    return (MDd)
# 100% SNP fingerprint for identification/verification purposes:
def Find_Finger(paired): # builds a dictionary of all 90+ snps shared between both
colonies
    finger = {}
    for k,v in (paired.items()):
        if v[0][0] > 90 and v[1][0] > 90:
            finger[k] = v
    return (finger)
# now doing the same but just for the hets. (could easily make these run together...)
def Find_HetSNP(paired):
    hetSNP = {}
```

```
    for k,v in (paired.items()):
        if (20 < v[0][0] < 80) and (20 < v[1][0] < 80):
            hetSNP[k] = v
    return (hetSNP)
ID = 'IanWSAge-4-'
M, D = 'null', 'null'
pair = ['','d']
formatter = ['','o','fy']
for num in range(1,12):
# num = '15' #can change to range
    try:
        for form in pair: #initiates data frames for m/d pairs
            if form == '':
                init_M = Frameit(ID, num, form)
            elif form == 'd':
                init_D = Frameit(ID, num, form)
        init_paired = Unify(init_M,init_D)
        init_finger = Find_Finger(init_paired)
        init_het = Find_HetSNP(init_paired)
        # num = '14'
        # adjusting formatter list to find old lineage pair
        for form in pair:
            if form == '':
                o_M = Frameit((ID + formatter[1]), num, form)
            elif form == 'd':
                o_D = Frameit((ID + formatter[1]), num, form)
        o_paired = Unify(o_M,o_D)
        o_finger = Find_Finger(o_paired)
        o_het = Find_HetSNP(o_paired)
        # for k,v in init_het.items():
        #     if k in o_finger:
        #         print(f"O LOH at {k} with {v} and {o_finger[k]}")
        # and forever young lineage (SGS1 lines have no FY lineage, commenting out)
        for form in pair:
            if form == '':
                fy_M = Frameit((ID + formatter[2]), num, form)
            elif form == 'd':
                fy_D = Frameit((ID + formatter[2]), num, form)
        fy_paired = Unify(fy_M,fy_D)
        fy_finger = Find_Finger(fy_paired)
        fy_het = Find_HetSNP(fy_paired)
        # export for aged:
        with open(f"{ID}{num}LOH.txt", "w+") as file:
```

```
    for k,v in init_het.items():
        v1 = float(v[0][0])
        v2 = float(v[0][1])
        v3 = float(v[1][0])
        v4 = float(v[1][1])
        if k in o_finger:
            o1 = float(o_finger[k][0][0])
            o2 = float(o_finger[k][0][1])
            o3 = float(o_finger[k][1][0])
            o4 = float(o_finger[k][1][1])
            file.write(f"{k}\t{v1}\t{v2}\t{v3}\t{v4}\t{o1}\t{o2}\t{o3}\t{o4}\n")
            print(f"LOH at {k} with {v} and {o_finger[k]}")
# # export for forever young:
with open(f"{ID}{num}LOH_fy.txt", "w+") as file:
    for k,v in init_het.items():
    # for k,v in fy_het.items():
        v1 = float(v[0][0])
        v2 = float(v[0][1])
        v3 = float(v[1][0])
        v4 = float(v[1][1])
        if k in fy_finger:
        # if k in o_finger:
            o1 = float(fy_finger[k][0][0])
            o2 = float(fy_finger[k][0][1])
            o3 = float(fy_finger[k][1][0])
            o4 = float(fy_finger[k][1][1])
            file.write(f"{k}\t{v1}\t{v2}\t{v3}\t{v4}\t{o1}\t{o2}\t{o3}\t{o4}\n")
            # print(f"LOH at {k} with {v} and {o_finger[k]}")
# generating frequency distribution graph:
# Test using just old pairs:
# dist = Unify(o_M,o_D)
# distlist = []
# for k,v in dist.items():
#     distlist.append(v[0])
#     distlist.append(v[1])
#     if abs(v[0]-v[1]) > 30 :
#         print((k,v))
# import matplotlib.pyplot as plt
# distrplot = plt.hist(distlist, 300, density=True, facecolor='g', alpha=0.75)
# plt.hist(distlist, 300, density=True, facecolor='g', alpha=0.75)
# distrplot.savefig(frequency_distribution, dpi=300, facecolor='w', edgecolor='w',
#         orientation='portrait', papertype=None, format=None,
#         transparent=False, bbox_inches=None, pad_inches=0.1,
```

```
        #         frameon=None, metadata=None)
    except FileNotFoundError:
        print("error, file not found")
        continue
```

LOHgenome_graph.py

```
#This script imports sets of values and a masked genome and returns a graph of those
#values along vertically distributed chromosome lines
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from Bio import SeqIO
from matplotlib import collections  as mc
import pylab as pl
figsize = (24, 6)
fig, ax = plt.subplots(figsize=figsize)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.yticks(range (1,17))
plt.xlim(0, 1600000)
plt.ylim(0, 17)
ax.set_ylabel('Chromosome', fontname="Arial", fontsize=12)
ax.set_xlabel('Basepair Position', fontname="Arial", fontsize=12)
ax.set_title('LOH events, wildtype lineages', fontname="Arial", fontsize=16)
# Blank Chromosomes Construction
lengths = [230218,813184,316620,1531933,576874,
          270161,1090940,562643,439888,745751,666816,1078177,924431,784333,1091
291,948066]
output = []
for i in range(0, 16):
    output.append([  (0, i+1), (lengths[i], i+1)    ])
c = "black"
lc = mc.LineCollection([i for i in output], colors=c
, linewidths=2)
ax.add_collection(lc)

# Chromosome key lookup (For mutations and Masked genome construction)
dchrom = {'NC_001133' : 1, 'NC_001134' : 2, 'NC_001135' : 3, 'NC_001136' : 4,
'NC_001137' : 5 , 'NC_001138' : 6,
```

```
'NC_001139' : 7, 'NC_001140' : 8, 'NC_001141' : 9, 'NC_001142' : 10, 'NC_001143' :
11, 'NC_001144' : 12,
'NC_001145' : 13, 'NC_001146' : 14, 'NC_001147' : 15, 'NC_001148' : 16, 'NC_001224'
: 'M'}
# # Generate a means of looking up a position along a chromosome to see if it's
masked ('N'), requires the referenced file: S288C-masked-genome.fasta
# # This section builds the yeast masked genome positions ('N') as a list (N_list) in the
form (Chromosome, position) of all masked positions
genome_dict, record_dict = {}, {}
for record in SeqIO.parse("S288C-masked-genome.fasta", "fasta"):
    record_dict[record.id] = record.seq
position = 0
N_list = []
for key,value in record_dict.items():
    position = 0
    for i in value:
        if i == 'N' and key != 'Mito':
            N_list.append((dchrom[key.split('|')[1]], position))
        position += 1
color = 'grey'
mark = '|'
for i in range (1,17):
    ax.scatter([x[1] for x in N_list if x[0] == i], ([i for x in N_list if x[0] == i]), s=None,
c=color, marker=mark, label="Masked Base" if i ==1 else "", alpha=0.2,zorder=2)

# Generate a list of SNVs:
loh_file = 'LOHresults_compiled_woMito.txt'
lohs = []
lineages = []
with open(loh_file, 'r') as rr:
    loh_lines  = [(i.split('\n')[0].split('\t')) for i in rr]
    for i in loh_lines:
        if i[1].split(':')[0] != 'Mito':
            # print(i)
            chrm = dchrom[i[1].split(':')[0].split('|')[1]]
            pos = int(i[1].split(':')[1])
            lineage = i[0]
            snp = [chrm, pos, lineage]
            lohs.append(snp)
            lineages.append(lineage)
lineages = set(lineages)
lineages = list(lineages)
for lin in lineages:
```

```
    mark = '|'
    eventlist = []
    color_num = lineages.index(lin)
    print(color_num)
    colorlist = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'tab:purple']
    color = colorlist[color_num]
    offset = 0
    for event in lohs:
        if event[2] == lin:
            eventlist.append(event)
    for i in range(1,17):
        ax.scatter([x[1] for x in eventlist if x[0] == i], ([i+offset for x in eventlist if x[0] == i]),
s=None, c=color, marker=mark, label=(f"{lin} lineage") if i ==1 else "", alpha=1,zorder=3)
    offset = 0.4
    mark = 'triangle'
    triangle_down
    mark = 7
    for i in range(1,17):
        ax.scatter([x[1] for x in eventlist if x[0] == i], ([i+offset for x in eventlist if x[0] == i]),
s=None, c=color, marker=mark, label=(f"{lin} lineage") if i ==1 else "", alpha=1,zorder=3)
color = 'green'
mark = '|'
for i in range (1,17):
    ax.scatter([x[1] for x in snps if x[0] == i], ([i for x in snps if x[0] == i]), s=None, c=color,
marker=mark, label="SNVs" if i ==1 else "", alpha=1,zorder=3)
# Optional Centromere addition:
cent_file = 'centromeres.txt'
with open(cent_file, 'r') as cent:
    cent_lines = [i.split('\t') for i in cent]
cents = [(int(i[2].split('..')[0])+int(i[2].split('..')[1]))/2 for i in cent_lines]
for i in range (1,17):
    ax.scatter(cents[i-1],i, s=20, c='grey', marker='o',
    # label="Centromere" if i ==1 else "",
    alpha=0.8, zorder=4)
ax.legend()
plt.show()
```