Title
Analysis of the roles of TIN2 splice forms in human telomere regulation and maintenance

Author
Cimini, Beth

Publication Date
2016

Peer reviewed|Thesis/dissertation

Analysis of the roles of TIN2 splice forms in human telomere regulation and maintenance

by

Beth Cimini

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Biochemistry and Molecular Biology

in the

[redacted]

Dedication and acknowledgements

Chapter 1 is presented as in preparation for Nucleic Acids Research 5/2016

Chapter 2 is presented as in preparation for Bioinformatics 5/2016

A thesis is never created in a vacuum, but I've been incredibly fortunate to have a lot of people that I need to thank for helping me get this far.

First and foremost to Liz, for making me feel welcome as a recruit, feel wanted as a graduate student, and feel like you've always believed I had it in me most especially when I was sure I didn't. Thank you for always saying yes.

To Brad, for helping me get started, for never rolling your eyes at me, for being a mentor for a whole lot longer than 10 weeks, and for reminding me to always disbelieve my data.

To Francesca, Birna, Tet, and Kyle Jay, the finest friends and graduate students I could've hoped to have around me. I'll never forget the late nights together in lab, and the later nights together out of lab having fun.

To Dana and Jue, for seeing this journey through with me beginning to end, for listening when I yell crazy things across the lab, for always having cake, and for always being there no matter what.

To Imke and Morgan, for wise advice, coffee breaks, and much-needed frequent kicks in the behind. I hope I can be more like you when I grow up.

To Shang, Lifeng, Carol, Kyle L, Josh, Jason, Tracy, Jie, Stephanie, Ajay, and Florie, for being the best damn labmates (and adopted labmates) a person could ever work with. I learned so much from each of you, and yet I'm sure it was only a fraction of all you had to offer.

To Toni and Nancy, for whom juggling a thousand things seemed terribly straightforward and whose responses to requests for help was always "Of course!".

To Kurt and DeLaine, for teaching me about microscopes, for endless answering of "quick questions", and for making most of my experiments possible with your equipment and your knowledge.

To Pat, Orion, and Dyche, for doing their best to corral such a headstrong and willful charge.

To Stacey, for always being my partner in crime, for always having time for me, and for making me hide $20 bills in your purse.

To my "girls", without whom I truly could not have made it through graduate school. I could never have imagined having friends like you and I'll never stop feeling astonished that I do.

To Tetrad '07, the smartest and most fantastic group of kids I know. Thanks for being partners on a great adventure, for letting me be very inappropriate, and for not making last feel least.

To CellProfiler, for giving me data to analyze and a chance to figure out what I love.

To the city of San Francisco, for just being awe$ome.

To Mom, my first, last, and best champion, for liking me even if I weren't your daughter.

To Meghan, for thinking you love me more (you don't) and for bugging me to move home sooner.

To Michael and Angela, for being amazing role models and for never bugging me to move home sooner.

To Matthew, Christina, Enzo, Luca, and Lily, for forgiving me for missing a lot of years.

To the ones I've lost, part of you is always with me and generally pops up at the strangest and least appropriate times. Just like you'd want.

To the one that I've been lucky enough to find, thanks for taking a tremendous chance on me and for pointing out the wonder in things I'd started to take for granted.

To those without category or who've been shamefully omitted, thanks for everything you've done to get me here.

Here's to the quintessential graduate school experience: coffee, beer, therapy, coffee, late-night microscope dance parties, coffee, and every now and then an "aha!" that makes it all worthwhile.

Analysis of the roles of TIN2 splice forms in human telomere regulation and maintenance

Beth Cimini

Abstract

Mammalian telomeres are protected by a 6-protein unit called the shelterin complex; the scaffold protein, TIN2, regulates the stability and recruitment of all of the other members and additionally controls telomerase access. In humans TIN2 has been shown to have two major isoforms; the shorter, soluble TIN2S isoform and the longer, nuclear matrix-associated TIN2L. To determine whether the isoforms have separate functions we moderately expressed GFP-TIN2S or GFP-TIN2L either alone or alongside an shRNA to knock down the endogenous TIN2. We found that while either isoform supported telomerase access and telomere elongation, GFP-TIN2L showed a significant defect in protecting telomeres from DNA damage and that this defect was likely due to decreased ability to associate with the telomere. Both phenotypes could be modulated by a point mutation of the TIN2L S396 casein kinase II consensus phosphorylation site. We also found that the S270 ATM/ATR consensus phosphorylation site showed no effect when mutated in GFP-TIN2S but had significant effects on GFP-TIN2L's association with TRF1. TIN2L's access to the telomere is therefore much more regulated than TIN2S's and may hint at a role for TIN2L as a factor that can quickly modulate telomere behavior whenever the cell requires.

In the course of this work I began accumulating very large data sets in CellProfiler, running tens of thousands of measurements in each experiment. In order to cope with the analysis I began to write Python scripts with a basic GUI to automate some simpler tasks (selecting

parameters to analyze, pulling those columns from the spreadsheet of each image analysis set, and running statistical comparisons), then added the ability to save default parameter sets, generate graphs, to run a few extra analyses not offered by CellProfiler. I found myself wishing such a program had already existed to help me analyze my data and wanting to share these functions with labmates who ran similar experiments but didn't know how to program. In Chapter 2 I describe the culmination of this work - CellProfilerStats, a free open-source program to parse image analysis data into statistical analyses and high-quality graphs, no programming required.

Table of Contents

List of tables

List of figures

Introduction:

Chapter 1:

Introduction

As the famous expression goes, "nothing in biology makes sense except in the light of evolution"(1); the roles and functions of telomeres firmly fall within this category. While evolution is a fantastic adaptor of parts currently at hand, it cannot anticipate future needs and can only revise major systems with great difficulty. These limitations lead to many systems in biology where problems are not truly solved, only avoided. Two examples of systems requiring circumvention of serious issues are the faithful replication of DNA and the repair of damaged DNA, both of which are necessary for organisms to propagate their genomes and to ensure their genes live on.

Replication is carried out by DNA polymerases, which achieve their fidelity through a combination of a) promoting incorporation of only the correct nucleotide (which produces a mutation rate of 1 in 10^{-4}-10^{-5}) and b) subsequent error correction via a 3'->5' exonuclease which removes mispaired bases (decreasing the error rate to 1 in 10^{-7}). This high level of fidelity is so critical that the mechanism of DNA polymerase actually prevents it from attempting *de novo* synthesis of DNA when provided with only a single-stranded template and dNTPs, as it would not be able to properly sense if error correction was necessary in the initial bases (reviewed in (2)). In order to begin DNA replication therefore the cell must first synthesize an RNA primer at the origin of replication, which is then handed off to DNA polymerase to be extended and which will eventually be excised from the elongating DNA molecule(3)). Replication then continues around the circular chromosome until the whole genome (including the bases at the origin that were initially synthesized as RNA) is fully replicated (Fig 1A). While this use of RNA as a primer is an elegant work-around for the

limitations of DNA polymerase in organisms with circular chromosomes, it created a significant problem upon the transition to linear chromosomes: after excision of the initial RNA primer in a linear chromosome, there is no mechanism to replace the sequence that has been excised and those bases are lost forever from that DNA strand (Fig 1B). Cells with linear chromosomes therefore find themselves in a situation where genetic information is lost with every division, a critical issue known as the "end-replication problem"(4).

The transition to linear chromosomes was similarly problematic for the DNA damage sensing machinery. In a circular chromosome every free DNA end can and should be treated as a sign of DNA damage that should be quickly repaired (Fig 1A); failure to do so can lead to chromosome fragmentation and loss of genetic information after replication and cell division. In contrast, a linear chromosome naturally has free DNA ends that the cell should leave alone (Fig 1B); attempts to erroneously "fix" the ends of linear chromosomes at worst can lead to dicentric chromosomes, subsequent problems in mitosis, and eventual aneuploidy (5) and at best create re-circularized chromosomes that allow cells to grow vegetatively but that cannot properly undergo meiosis (6).

Rather than attempt to overhaul the critical DNA replication and DNA damage sensing systems, evolution introduced a work-around: the telomere. In the vast majority of eukaryotes telomeres consist of a TG-rich repeat(7) that can be added to the end of a chromosome by the telomerase reverse transcriptase(8), which is thought to be evolutionarily derived from parasitic retrotransposons(9). Since telomeres contain no unique genomic information and can be lengthened by telomerase, the cell can afford to lose the end of the chromosome during replication; since the sequence added is a repetitive element it allowed for the evolution of telomere binding-proteins that can create a privileged chromatin state at the chromosome end

that keeps the telomere from being sensed as DNA damage and erroneously "repaired" (reviewed in (10)).

In mammals the telomeres consist of hundreds to thousands of TTAGGG repeats bound by a 6 protein complex known as the shelterin complex (10). The shelterin complex contains 3 telomere-repeat binding proteins: POT1 (11), which binds ssDNA telomere repeats and is itself bound by the telomerase recruitment shelterin protein known as TPP1(12, 13); TRF2(14), which binds dsDNA telomere repeats, associates with the Rap1 shelterin protein(15), and is important for preventing the DNA damage response protein ATM from sensing the telomere as a substrate in need of repair(16); and TRF1(17), which also binds dsDNA telomere repeats and acts to limit telomerase association with the telomere. The final member of the shelterin complex is TIN2 (18), which binds to TRF1, TRF2, and TPP1 and acts as a scaffold to hold the complex together. When the shelterin complex is present in sufficient quantities the telomere evades detection by the DNA damage machinery and is said to be "capped"; loss of the complex due to telomere shortening or due to the deletion or mutation of members of the shelterin complex causes the DNA repair machinery to accumulate at the now "uncapped" telomere (10).

The presence of proteins that both increase (TPP1) and decrease (TRF1) telomerase recruitment in the shelterin complex hints that telomerase action must be both dynamic and highly regulated. While the addition of extra telomeric repeats may seem innocuous, studies have shown that due to the repetitive nature and high G content of telomeric repeats they present a particular challenge for the DNA replication machinery (19). In multicellular organisms the need to regulate telomerase access to the telomere is even more complex: telomere length is a major factor in determining how many times a cell can divide before it

undergoes senescence and helps to set the so-called "Hayflick limit" of cell division (reviewed in (20)). An organism can therefore regulate how proliferative each of its tissues can be via the regulation of telomerase expression. The importance of this control cannot be overstated; while most differentiated cells do not express telomerase, 85-90% of tumors re-activate telomerase expression (20). The need for precision in telomere length regulation is underscored by meta-analyses that show that having longer-than-average telomeres leads to increased risk of a certain subset of cancers, while possessing shorter-than-average telomeres shows increased risk of a different subset (21).

Given the evident importance of telomere length regulation it is unsurprising that mutations in telomerase components are not well tolerated; mutations in any of several telomerase-associated genes lead to a class of diseases previously referred to as dyskeratosis congenita but now simply referred to as "telomere syndromes". Haploinsufficiency of the main protein component of telomerase, TERT, leads to generationally progressive shortening of telomere length and a corresponding increase in disease severity. A person with an inherited TERT mutation may present with telomere lengths in the 1st percentile and be diagnosed with pulmonary fibrosis in middle age; those of their children who inherit the mutation possess even shorter telomeres, predisposing them to develop disorders such as aplastic anemia in early adulthood. The following generations are at risk of a wide variety of cancers, immunodeficiency, and even progressive organ failure in childhood or adolescence (22).

In contrast to telomere syndromes found in families carrying a telomerase mutation, *de novo* single point mutations or truncations of the TIN2 shelterin scaffold can cause very short telomeres and severe telomere syndromes in young children whose parents and siblings show completely wild-type telomere lengths(23). TIN2's central scaffolding role in the shelterin

complex combined with the rapid onset of telomere shortening in TIN2 mutant patients point to TIN2 as a telomere master regulator whose function is indispensable for proper telomere maintenance.

Despite TIN2's importance it is relatively little-studied; as of the writing of this thesis a PubMed search for TIN2 returns fewer results (~900) than any of the other 5 shelterin proteins (~1100-2400) and 30-fold fewer results than TERT(~28000). Given these numbers it is perhaps not terribly surprising that it took until 2009 for researchers to discover that the TIN2 gene is alternatively spliced and that all of the known work to that point had been done on the shorter (or TIN2S) isoform (24). In the longer TIN2L splice variant three additional introns are excised allowing translation of a 451 amino acid protein; these introns are included in TIN2S, resulting in an immediate stop codon and termination of the protein after amino acid 354 (Fig 2). All of the known functional domains of TIN2 as well as the area of the protein that contains most of the telomere syndrome-causing mutations are found in the region of the protein shared between the two isoforms, and the TIN2L tail contains no predicted structural domains. The only known difference between the two splice variants is that during cellular fractionation TIN2S and the other shelterins fractionate with the bulk chromatin while TIN2L is largely retained in the nuclear matrix with proteins such as lamin A (24). The authors note that the nuclear matrix attachment is so strong that endogenous TIN2L cannot be seen on a Western blot unless the protein preparation is modified to ensure its inclusion in the soluble fraction; they hypothesize that this may be why TIN2L had never previously been observed.

The association of a shelterin protein with the nuclear matrix was not surprising as it has been known for some time that telomeric DNA, unlike bulk genomic DNA, is associated with the nuclear matrix (25). The significance of this attachment, however, has never been fully

understood. Subsequent work showed that the connection to the nuclear matrix was likely through a protein link (26) and that cells lacking the DNA damage repair kinase ATM showed significantly higher telomeric attachment to the nuclear matrix (27).

This link between nuclear matrix attachment and ATM is intriguing given results from our and others' labs showing that the diffusion speed of fluorescently tagged telomeres increases upon induction of telomere damage in an ATM-dependent manner(28, 29); detachment from a scaffold could certainly lead to an apparent increase in speed. Furthermore, recent work on the DNA damage response protein 53BP1 has indicated that upon induction of DNA damage it goes from being nuclear matrix-associated to being soluble, echoing the theme of leaving the nuclear matrix upon sensing DNA damage (30). Given these links and the fact that TIN2S and TIN2L share all the same functional domains and therefore are easily comparable, we set out to investigate whether the soluble TIN2S and nuclear-matrix bound TIN2L played different roles in the regulation of telomere capping and telomerase recruitment.

References

1. Dobzhansky,T. (1973) Nothing in Biology Makes Sense except in the Light of Evolution. Am. Biol. Teach., 35, 125–129.
2. Alberts,B., Johnson,A., Lewis,J., Raff,M., Roberts,K. and Walter,P. (2002) DNA Replication Mechanisms. In Molecular Biology of the Cell. Garland Science.
3. Reichard,P., Eliasson,R. and Söderman,G. (1974) Initiator RNA in discontinuous polyoma DNA synthesis. Proc. Natl. Acad. Sci. U. S. A., 71, 4901–4905.
4. Zakian,V.A. (2012) Telomeres: the beginnings and ends of eukaryotic chromosomes. Exp. Cell Res., 318, 1456–1460.
5. Artandi,S.E., Chang,S., Lee,S.L., Alson,S., Gottlieb,G.J., Chin,L. and DePinho,R.A. (2000) Telomere dysfunction promotes non-reciprocal translocations and epithelial cancers in mice. Nature, 406, 641–645.
6. Naito,T., Matsuura,A. and Ishikawa,F. (1998) Circular chromosome formation in a fission yeast mutant defective in two ATM homologues. Nat. Genet., 20, 203–206.

7. Blackburn,E.H. (2010) Telomeres and telomerase: the means to the end (Nobel lecture). Angew. Chem. Int. Ed Engl., 49, 7405–7421.

8. Greider,C.W. and Blackburn,E.H. (1985) Identification of a specific telomere terminal transferase activity in Tetrahymena extracts. Cell, 43, 405–413.

9. Nakamura,T.M. and Cech,T.R. (1998) Reversing time: origin of telomerase. Cell, 92, 587–590.

10. Palm,W. and de Lange,T. (2008) How shelterin protects mammalian telomeres. Annu. Rev. Genet., 42, 301–334.

11. Baumann,P. and Cech,T.R. (2001) Pot1, the putative telomere end-binding protein in fission yeast and humans. Science, 292, 1171–1175.

12. Liu,D., Safari,A., O'Connor,M.S., Chan,D.W., Laegeler,A., Qin,J. and Songyang,Z. (2004) PTOP interacts with POT1 and regulates its localization to telomeres. Nat. Cell Biol., 6, 673–680.

13. Ye,J.Z.-S., Hockemeyer,D., Krutchinsky,A.N., Loayza,D., Hooper,S.M., Chait,B.T. and de Lange,T. (2004) POT1-interacting protein PIP1: a telomere length regulator that recruits POT1 to the TIN2/TRF1 complex. Genes Dev., 18, 1649–1654.

14. Bilaud,T., Brun,C., Ancelin,K., Koering,C.E., Laroche,T. and Gilson,E. (1997) Telomeric localization of TRF2, a novel human telobox protein. Nat. Genet., 17, 236–239.

15. Li,B., Oestreich,S. and de Lange,T. (2000) Identification of human Rap1: implications for telomere evolution. Cell, 101, 471–483.

16. Takai,H., Smogorzewska,A. and de Lange,T. (2003) DNA damage foci at dysfunctional telomeres. Curr. Biol., 13, 1549–1556.

17. Chong,L., van Steensel,B., Broccoli,D., Erdjument-Bromage,H., Hanish,J., Tempst,P. and de Lange,T. (1995) A human telomeric protein. Science, 270, 1663–1667.

18. Kim,S.-H., Kaminker,P. and Campisi,J. (1999) TIN2 , a new regulator of telomere length in human cells. Nature Genetics, 23, 405–412.

19. Sfeir,A., Kosiyatrakul,S.T., Hockemeyer,D., MacRae,S.L., Karlseder,J., Schildkraut,C.L. and de Lange,T. (2009) Mammalian telomeres resemble fragile sites and require TRF1 for efficient replication. Cell, 138, 90–103.

20. Stewart,S. a. and Weinberg,R. a. (2006) Telomeres: cancer to human aging. Annu. Rev. Cell Dev. Biol., 22, 531–557.

21. Zhu,X., Han,W., Xue,W., Zou,Y., Xie,C., Du,J. and Jin,G. (2016) The association between telomere length and cancer risk in population studies. Sci. Rep., 6, 22243.

22. Armanios,M. and Blackburn,E.H. (2012) The telomere syndromes. Nat. Rev. Genet., 13, 693–704.

23. Savage,S. a., Giri,N., Baerlocher,G.M., Orr,N., Lansdorp,P.M. and Alter,B.P. (2008) TINF2, a Component of the Shelterin Telomere Protection Complex, Is Mutated in Dyskeratosis Congenita. Am. J. Hum. Genet., 82, 501–509.

24. Kaminker,P.G., Kim,S.H., Desprez,P.Y. and Campisi,J. (2009) A novel form of the telomere-associated protein TIN2 localizes to the nuclear matrix. Cell Cycle, 8, 1–9.

25. de Lange,T. (1992) Human telomeres are attached to the nuclear matrix. EMBO J., 11, 717–724.

26. Ludérus,M.E.E., Van Steensel,B., Chong,L., Sibon,O.C.M., Cremers,F.F.M. and De Lange,T. (1996) Structure, subnuclear distribution, and nuclear matrix association of the mammalian telomeric complex. J. Cell Biol., 135, 867–881.

27. Smilenov,L.B., Dhar,S. and Pandita,T.K. (1999) Altered Telomere Nuclear Matrix Interactions and Nucleosomal Periodicity in Ataxia Telangiectasia Cells before and after Ionizing Radiation Treatment. Molecular and Cellular Biology, 19, 6963–6971.

28. Wang,X., Kam,Z., Carlton,P.M., Xu,L., Sedat,J.W. and Blackburn,E.H. (2008) Rapid telomere motions in live human cells analyzed by highly time-resolved microscopy. Epigenetics Chromatin, 1, 4.

29. Dimitrova,N., Chen,Y.-C.M., Spector,D.L., de Lange,T. and Lange,T.D. (2008) 53BP1 promotes non-homologous end joining of telomeres by increasing chromatin mobility. Nature, 456, 524–528.

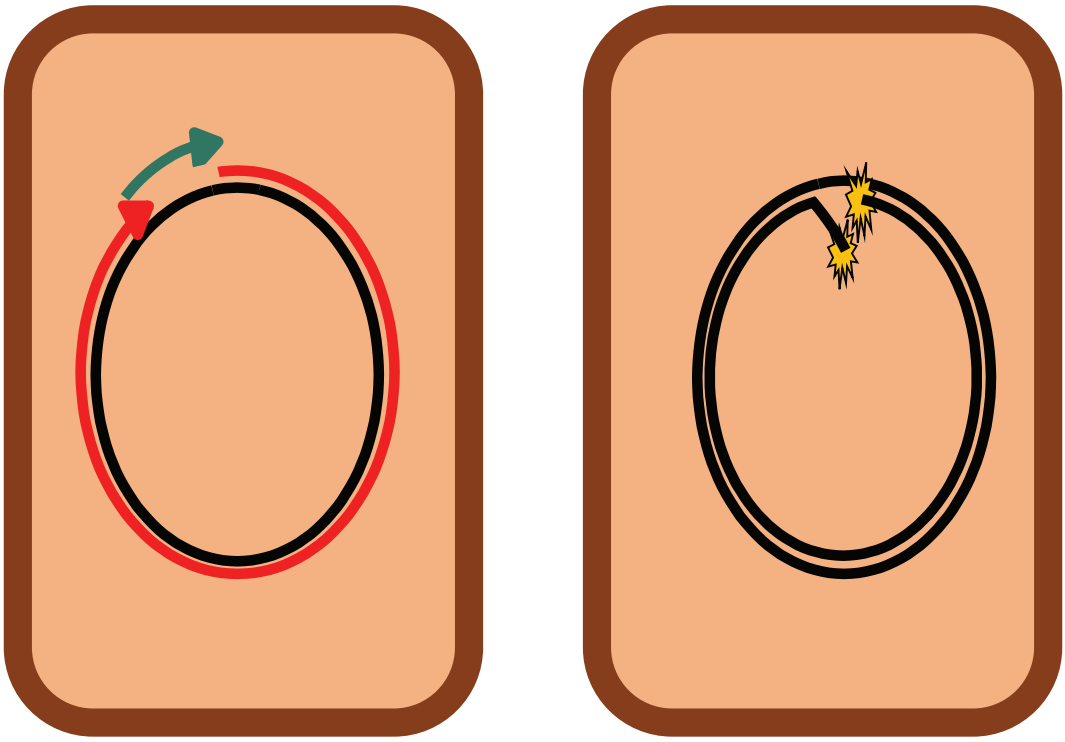
30. Gibbs-Seymour,I., Markiewicz,E., Bekker-Jensen,S., Mailand,N. and Hutchison,C.J. (2015) Lamin A/C-dependent interaction with 53BP1 promotes cellular responses to DNA damage. Aging Cell, 14, 162–169.

Figure Legends

Figure 1: DNA replication and DNA damage sensing in circular vs linear chromosomes. A) Illustration of DNA replication and DNA damage sensing in an ancestral organism with a circular chromosome. B) Illustration of the particular challenges of linear chromosomes that are solved by the use of telomeres.

Figure 2: Comparison of TIN2S and TIN2L with major features included. TIN2L has three additional splicing events leading to a longer protein product (albeit from a shorter mRNA). While the TPP1 and TRF1 binding domains of TIN2 have been narrowed to small regions, TRF2 binding has only been broadly mapped to the N-terminal half of the protein. The telomere syndrome mutation cluster shown in red accounts for a very high portion of the known TIN2 mutant patients. Scale bar=100

A)



B)

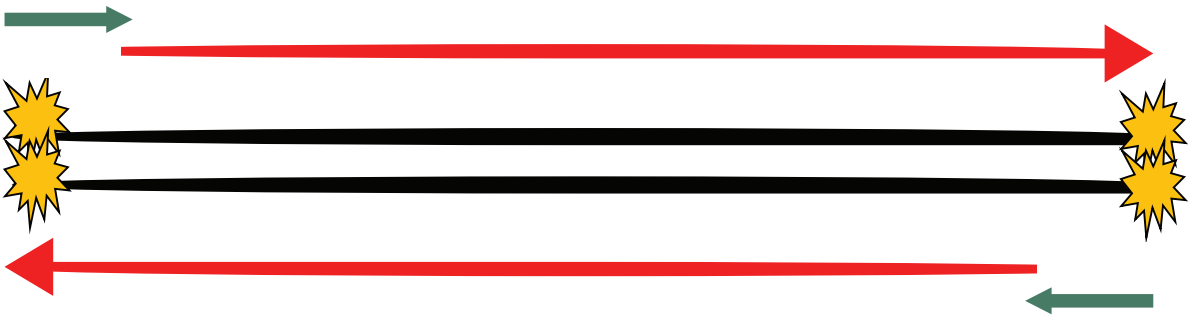


Figure 1: DNA replication and DNA damage sensing in circular vs linear chromosomes

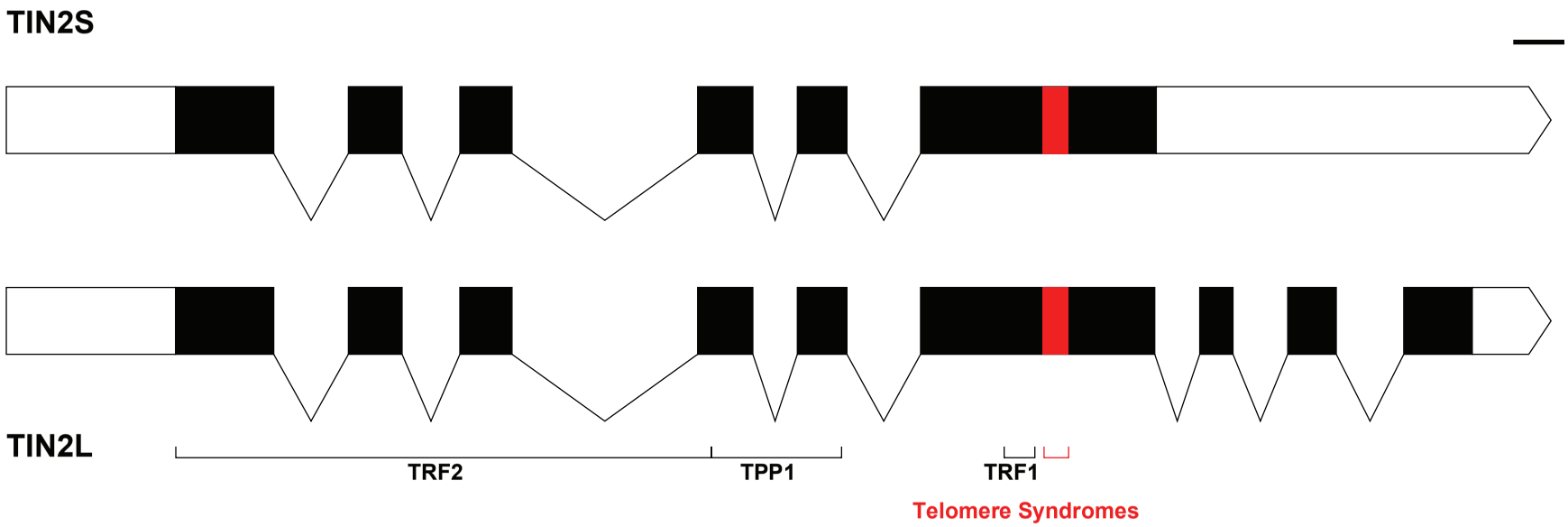


Figure 2: Comparison of TIN2S and TIN2L with major features included

Splice Variants of the Telomere Master Regulator Protein TIN2 Differentially Localize to and Protect the Telomere

Beth A Cimini[1] and Elizabeth H Blackburn[1*]

[1] Department of Biochemistry and Biophysics, University of California-San Francisco, San Francisco, CA 94158 USA

[*] To whom correspondence should be addressed. Tel: 1-858-453-4100; Fax: 1-858-546-0838; Email: eblackburn@salk.edu

Present Address: Elizabeth Blackburn, Salk Institute for Biological Studies, La Jolla, CA 92037, USA

ABSTRACT

Mammalian telomeres are protected by a 6-protein unit called the shelterin complex; the scaffold protein, TIN2, regulates the stability and recruitment of all of the other members and additionally controls telomerase access. In humans TIN2 has been shown to have two major isoforms; the shorter, soluble TIN2S isoform and the longer, nuclear matrix-associated TIN2L. To determine whether the isoforms have separate functions we moderately expressed GFP-TIN2S or GFP-TIN2L either alone or alongside an shRNA to knock down the endogenous TIN2. We found that while either isoform supported telomerase access and telomere elongation, GFP-TIN2L showed a significant defect in protecting telomeres from DNA damage

and that this defect was likely due to decreased ability to associate with the telomere. Both phenotypes could be modulated by a point mutation of the TIN2L S396 casein kinase II consensus phosphorylation site. We also found that the S270 ATM/ATR consensus phosphorylation site showed no effect when mutated in GFP-TIN2S but had significant effects on GFP-TIN2L's association with TRF1. TIN2L's access to the telomere is therefore much more regulated than TIN2S's and may hint at a role for TIN2L as a factor that can quickly modulate telomere behavior whenever the cell requires.

INTRODUCTION

Mammalian telomeres consist of hundreds to thousands of TTAGGG repeats bound by a 6 protein complex known as the shelterin complex (reviewed in (1)). The shelterin complex contains 3 telomere-repeat binding proteins: POT1 (2), which binds ssDNA telomere repeats and is itself bound by the telomerase recruitment shelterin protein known as TPP1(3, 4); TRF2(5), which binds dsDNA telomere repeats, associates with the Rap1 shelterin protein(6), and is important for preventing the DNA damage response protein ATM from sensing the telomere as a substrate in need of repair(7); and TRF1(8), which also binds dsDNA telomere repeats and acts to limit telomerase association with the telomere. The final member of the shelterin complex is TIN2(9), which binds to TRF1, TRF2, and TPP1 and acts as a scaffold to hold the complex together. When the shelterin complex is present in sufficient quantities the telomere evades detection by the DNA damage machinery and is said to be "capped"; loss of the complex due to telomere shortening or due to the deletion or mutation of members of the shelterin complex causes the DNA repair machinery to accumulate at the now "uncapped" telomere(1).

The presence of proteins that both increase (TPP1) and decrease (TRF1) telomerase recruitment in the shelterin complex hints that telomerase action must be both dynamic and highly regulated. While the addition of extra telomeric repeats may seem innocuous, studies have shown that due to the repetitive nature and high G content of telomeric repeats they present a particular challenge for the DNA replication machinery (10). In multicellular organisms the need to regulate telomerase access to the telomere is even more complex: telomere length is a major factor in determining how many times a cell can divide before it undergoes senescence and helps to set the so-called "Hayflick limit" of cell division (reviewed in (11)). An organism can therefore regulate how proliferative each of its tissues can be via the regulation of telomerase expression. The importance of this control cannot be overstated; while most differentiated cells do not express telomerase, 85-90% of tumors re-activate telomerase expression (11).

Given the evident importance of telomere length regulation it is unsurprising that mutations in telomerase components are not well tolerated; mutations in any of several telomerase-associated genes lead to a class of diseases previously referred to as dyskeratosis congenita but now simply referred to as "telomere syndromes". Haploinsufficiency of the main protein component of telomerase, TERT, leads to generationally progressive shortening of telomere length and a corresponding increase in disease severity (12). In contrast, de novo single point mutations or truncations of the TIN2 shelterin scaffold can cause very short telomeres and severe telomere syndromes in children whose parents and siblings show completely wild-type telomere lengths (13). TIN2's central scaffolding role in the shelterin complex combined with the rapid onset of telomere shortening in TIN2 mutant patients point to TIN2 as a telomere master regulator whose function is indispensable for proper telomere maintenance.

Ten years after the initial description of TIN2 it was discovered that in human cells the TIN2 gene is alternatively spliced and that all of the known work to that point had been done on the shorter (or TIN2S) isoform (14). In the longer TIN2L splice variant three additional introns are excised allowing translation of a 451 amino acid protein; these introns are included in TIN2S, resulting in an immediate stop codon and termination of the protein after amino acid 354 (Fig 2). All of the known functional domains of TIN2 as well as the area of the protein that contains most of the telomere syndrome-causing mutations are found in the region of the protein shared between the two isoforms, and the TIN2L tail contains no predicted structural domains. The only known difference between the two splice variants is that during cellular fractionation TIN2S and the other shelterins fractionate with the bulk chromatin while TIN2L is largely retained in the nuclear matrix with proteins such as lamin A (14). The authors note that the nuclear matrix attachment is so strong that endogenous TIN2L cannot be seen on a Western blot unless the protein preparation is modified to ensure its inclusion in the soluble fraction; they hypothesize that this may be why TIN2L had never previously been observed.

The association of a shelterin protein with the nuclear matrix was not surprising as it has been known for some time that telomeric DNA, unlike bulk genomic DNA, is associated with the nuclear matrix (15). The significance of this attachment, however, has never been fully understood. Subsequent work showed that the connection to the nuclear matrix was likely through a protein link (16) and that cells lacking the DNA damage repair kinase ATM showed significantly higher telomeric attachment to the nuclear matrix (17).

The link between nuclear matrix attachment and ATM is intriguing given results from our and others' labs showing that the diffusion speed of fluorescently tagged telomeres increases upon induction of telomere damage in an ATM-dependent manner(18, 19); detachment from a

scaffold could certainly lead to an apparent increase in speed. Furthermore, recent work on the DNA damage response protein 53BP1 has indicated that upon induction of DNA damage it goes from being nuclear matrix-associated to being soluble, echoing the theme of leaving the nuclear matrix upon sensing DNA damage (20). Given these links and the fact that TIN2S and TIN2L share all the same functional domains and therefore are easily comparable, we set out to investigate whether the soluble TIN2S and nuclear-matrix bound TIN2L played different roles in the regulation of telomere capping and telomerase recruitment.

MATERIAL AND METHODS

Cell culture and cell line construction

UM-UC-3 bladder cancer cells were grown in Dulbecco's modified Eagle's medium supplemented with 10% (vol/vol) fetal bovine serum, 1% (vol/vol) GlutaMAX-1 (Gibco) and 1% (vol/vol) penicillin–streptomycin (Gibco). Cells were grown at 37°C in 5%CO_2. FACS sorted cell lines were created using a FACSAriaII (BD Biosciences). Selection of WT hTR, 47A hTR and all shRNA constructs was accomplished via 24-48hrs selection in 1μg/mL puromycin; TSQ1 expressing cells were selected via FACS sorting for mApple, which was encoded on the same vector.

Lentiviral plasmids and production

The lentiviral vector system was provided by Didier Trono (University of Geneva, Geneva, Switzerland (21)). Vectors for introducing a control shRNA (5'-GTTCTACAACGTAACGAGGTT-3') as well as for WT, 47A, and TSQ1 versions of hTR have

been described previously (22, 23). TIN2 shRNAs were generated against the following sequences: TIN2shRNA1 (5'-GCAAGGAAGAACATGCGATATACA-3'), TIN2shRNA2 (5'-GGCACACATCTTCCTCAGGGACCAT-3'), TIN2shRNA3 (5'-GTCAGAGGCTCCTGTGGATTTGG-3'), TIN2shRNA4 (5'-GAACAGAATCCTCCTCAGCAACAAA-3'). Plasmids expressing the following ORFs were received from the following sources: mApple-C1: Addgene 54631; iBlueberry-C1: generous gift of Dr. Xiokun Shu via the UCSF Nikon Imaging Center; mTagBFP2: Mike Davidson plasmid collection via the UCSF NIC; Histone 2B: Addgene 21044; TIN2L cDNA: OpenBiolabs MHS1011-74557; 53BP1 cDNA: OpenBiolabs 8327629. GFP-TRF1 expressing lentiviral constructs have been previously described (18). Construction of lentiviral shRNA plasmids has been previously described (23), and production of lentiviral fluorescent protein fusions was done via Gibson assembly cloning (24). Creation of TIN2 point mutants was carried out by QuikChange Lightning Site-Directed Mutagenesis Kit (Agilent) using the recommended primers (http://www.genomics.agilent.com/primerDesignProgram.jsp).

Cell viability and cell cycle analysis

Apoptosis was measured with Caspase-Glo 3/7 kit (Promega) according to the manufacturer's instructions and was read by the Veritas Microplate Luminometer (Turner BioSystems). Cell cycle analysis was measured by staining for DNA content with Vybrant DyeCycleRuby (Life Technologies) per manufacturer's instructions and was measured alongside GFP expression on a FACSCalibur (BD Biosciences).

Immunofluorescence and FISH

Cells used for IF-FISH analysis were grown on Lab-Tek II chamber slides (Nunc) then fixed 15' in 2% paraformaldehyde. After permeabilization and blocking slides were incubated overnight at 4° in 1:500 rabbit anti-53BP1 (NB100-304; Novus Biologicals) and 1:300 mouse anti-ATM pS1981 (Rockland) followed by incubation for one hour at room temperature in the appropriate Alexa 488 and Alexa 647 -labeled secondary antibodies. After fixation of secondary antibodies FISH with Cy3 labeled telomere PNA probes (Panagene) proceeded as in (22) without the pepsin treatment. Cells used for TSQ1-FISH were grown on Lab-Tek II chamber slides then fixed and processed exactly as in (22).

Microscopy, image processing, and image analysis

Imaging of FISH and IF-FISH slides was performed on a Metafer microscope Metasystems using a 60X/1.42NA Olympus Plan-Apo objective and denoised using Huygens (Scientific Volume Imaging). FRAP was performed on a Nikon Ti widefield microscope with a Zyla 5.2 (Andor) sCMOS camera using a 100X/1.4NA Plan-Apo objective, Rapp UGA-40 photobleaching system and an Okolab incubator to maintain 37°C and 5%CO_2 . Analysis of shelterin localization to telomeres was performed on a Nikon Ti-E widefield microscope with a Flash 4.0 (Hamamatsu) sCMOS camera using 40X/0.95NA PlanApo objective and an Okolab incubator to maintain 37°C and 5%CO_2, followed by deconvolution using Huygens. FRAP analysis was performed by ROI measurement in FIJI (25) followed by curve fitting in Python using SciPy (http://www.scipy.org/) . All other image analysis was performed in CellProfiler (26) (pipelines available upon request) followed by graphing and statistical analysis in custom Python software (Cimini and Blackburn, in preparation). When measurements of individual telomeres were

made using fluorescently tagged shelterin, values were normalized to mean nuclear intensity to control for any heterogeneity of expression levels.

Western blotting

Cell pellets were frozen at -80 and the volume of RIPA buffer added was calculated by cell counts obtained before freezing. Protein was isolated via lysing cells in RIPA buffer (1% NP-40, 150mM sodium chloride, 25mM Tris ph7.8, 0.1% SDS, 1% sodium deoxycholate plus cOmplete EDTA-free Protease Inhibitor Cocktail (Roche)), followed by treatment with DNase (New England Biolabs) for 1-2hours before addition of 1 volume of 4X Odyssey loading buffer (LI-COR). After isolation, protein was boiled for 10' then spun 5' at maximum speed to clear the lysate. Samples were loaded on 4-12% NuPage Bis-Tris gels (LifeTechnologies) and then run in MOPS buffer and transferred to Immobilon-FL PVDF (0.45µm pore size) per manufacturer's instructions. Blocking and antibody dilution was carried out in Odyssey blocking buffer (LI-COR). The following antibodies and dilutions were used: Abcam rabbit anti TIN2 EPR15319 (1:1000), Roche mouse anti GAPDH (1:20000), Pierce donkey anti rabbit DyeLight 680 (1:5000), Pierce donkey anti mouse DyeLight 800 (1:5000). Primaries were incubated overnight at 4 degrees in the stated dilutions plus 0.1% Tween, and after washing blots were incubated in secondary antibodies in the stated dilutions plus 0.1% Tween and 0.02% SDS for 30-60 minutes. Blots were read on an Odyssey scanner (LI-COR) and quantified in FIJI relative to a loading control.

RESULTS

A large increase in telomere damage caused by TIN2 shRNA knockdown can occur without altering cell cycling or proliferation

As TIN2S and TIN2L are very similar there are very few sites to target for splice form-specific shRNA construction, none of which were successful in our hands. We therefore decided to create isoform-enriched lines by overexpression of fluorescently tagged TIN2S or TIN2L. By using fluorescently tagged TIN2 we could not only ensure equal starting levels of exogenous expression but could tune the expression levels to give us a population with a more modest and physiological level of TIN2. Subsequent expression of TIN2 shRNAs could then enrich the desired isoform even further by removing most of the endogenous TIN2, leaving us with a population of cells with relatively normal overall levels of TIN2 but enriched for a single splice form.

GFP-TIN2S or GFP-TIN2L were introduced into UM-UC-3 bladder cancer cells, which have a telomere length close to that of normal human cells and that our lab has previously shown are excellent for telomere imaging (18). In addition, this cell line abundantly expresses the telomerase protein component hTERT but is limited in its expression of the telomerase RNA hTR, which allows easy telomere length manipulation and the ability to directly test telomerase activity by introduction of mutant hTR (23). 4 shRNAs were designed and created across the shared region of the two splice variants (Fig 1A). TIN2 shRNAs were then introduced into these lines as well as into the parental UM-UC-3 cells. TIN2 shRNAs 1-4 all showed strong ability to knock down GFP-TIN2S and GFP-TIN2L (Fig S1A-B). However as TIN2 shRNAs

which show significant TIN2 protein reduction on a Western blot may not actually remove enough TIN2 to induce telomere damage (28), we additionally assayed the parental UM-UC-3 cells for uncapped telomeres by testing for colocalization with 53BP1. By this metric, only TIN2 shRNAs 1 and 4 were effective at stripping enough TIN2 from the telomeres to cause them to become uncapped (Fig 1B-C), with TIN2 shRNA1 the most effective and TIN2 shRNA4 presenting an intermediate effect. While the combined level of endogenous TIN2S and TIN2L can be difficult to quantify due to the extreme difficulty of solubilizing TIN2L (14), by our best estimate in UM-UC-3 TIN2S has 5 fold greater expression than TIN2L and the GFP-tagged copies of TIN2 are 20 fold overexpressed relative to endogenous TIN2 when coexpressed with a control shRNA, 2 fold overexpressed with TIN2 shRNA1, and 5 fold overexpressed with TIN2 shRNA4.

Curiously, despite the significant numbers of TIFs induced by both shRNAs, neither induced a growth defect (Fig S2A) or caused more than a modest induction of caspase 3/7 activity (Fig S2B) in UM-UC-3, though a decrease in cell growth was observed for TIN2 shRNA1 when introduced into CD4+ primary T cells (F. Gazzaniga, personal communication). This is contrary to previous TIN2 knockdown results in mouse cells (29), but is reminiscent of a study of a graded series of TRF2 shRNAs that showed that certain levels of shelterin depletion are severe enough to cause a local uncapped-like state but not quite strong enough to trigger the cell's full DNA damage response (30). These "intermediate-state" cells showed cellular activation of pS1981-ATM but not the downstream CHK2 kinase (30). While we see a small increase in pATM localization to telomeres upon treatment with TIN2 shRNAs it is not as robust as the pATM association caused by uncapping telomeres with the hTR mutant 47A (Fig S3), a process known to require ATM(31). Previous work in mouse has shown that uncapping due to TIN2

removal is only partially transduced through ATM and has a significant ATR-mediated component (32), which aligns with our findings; this indicates that the induction of intermediate-state uncapping may be a more general phenomenon that can be caused by many forms of telomere damage and not just via a TRF2-ATM signal transduction pathway.

Exogenous TIN2S and TIN2L are equally recruited to telomeres and are equally competent at telomerase recruitment and telomere elongation

We first wished to examine whether both GFP-TIN2S and GFP-TIN2L were equally capable of being recruited to telomeres. We imaged cells expressing either a GFP-TIN2 isoform or GFP-TRF1, defined the nucleus in an unbiased fashion via a co-expressed tagged H2B, and then measured total nuclear levels of GFP as well as the amount of GFP specifically found in telomeric foci (Fig S4A). GFP-TIN2S and GFP-TIN2L were comparable on each measure we examined: total nuclear brightness, number of telomeres identified in each nucleus, brightness of individual telomeres, and fraction of the nuclear GFP located inside telomeric foci (Fig S4B-E). We concluded that therefore GFP-TIN2S and GFP-TIN2L seem to be comparably able to localize onto telomeres at these expression levels, allowing us to make direct comparisons of their ability to carry out each of TIN2's various roles in telomere maintenance.

In order to measure the ability of each splice form to recruit telomerase, we took advantage of an assay using a template-mutant version of hTR called Tolerated SeQuence 1, or TSQ1. This version of hTR introduces an alternate repeat sequence that can be probed via PNA-FISH, but unlike other template mutants its incorporation does not trigger a DNA damage response at the telomere; this therefore allows visualization of which telomeres were visited by telomerase over the course of the assay (Fig 2A)(22). GFP-TIN2S and GFP-TIN2L both

allowed incorporation of TSQ1 after 4 days of expression (Fig 2B); in each of three repeats GFP-TIN2S allowed slightly more incorporation than GFP-TIN2L but only once was the difference statistically significant, leading us to conclude that both are broadly capable of recruiting telomerase, though TIN2L may have a small defect. Introducing TIN2 shRNA 1 3 days before TSQ1 addition significantly and equally reduced both GFP-TIN2S' and GFP-TIN2L's ability to incorporate TSQ1 (Fig S5), as expected given TIN2's known role in telomerase recruitment (33).

We next set out to study whether once telomerase was recruited to the telomere GFP-TIN2S and GFP-TIN2L would allow equal amounts of telomere elongation. Using qFISH, we determined that GFP-TIN2S and GFP-TIN2L expressing cells were both able to elongate their telomeres after 7 days overexpression of hTR (Fig 2C). When we examined the ability of elongated telomeres to recruit telomere binding proteins, both GFP-TIN2S and GFP-TIN2L show a comparable increase in recruitment as judged by fraction of nuclear GFP incorporated into telomeric foci, though neither increased as much as GFP-TRF1, which directly binds telomeric repeats(Fig 2D).

Elimination of endogenous TIN2 shows that TIN2L is less effective at telomere capping than TIN2S due to decreased recruitment

As opposed to the similarities GFP-TIN2S and GFP-TIN2L showed in their ability to allow telomerase access to the telomere, the two splice variants showed very different abilities to carry out TIN2's telomere capping functions. In all but 1 of 6 replicates, GFP-TIN2L showed a dramatic defect relative to GFP-TIN2S in preventing 53BP1 association with the telomere after introduction of TIN2 shRNAs (Fig 3A). When we examined the splice forms' recruitment to

the telomere after knockdown, we saw that despite similar nuclear levels of GFP-TIN2S and GFP-TIN2L, (Fig S6A), GFP-TIN2L showed much less association with telomeres (Fig 3B). This was evident both by directly examining the percent of GFP recruitment to telomeres (Fig 3C) and by measuring the mean GFP-TIN2 intensity of telomeres identified by mApple-TRF1 (Fig S6B). This splice form-specific difference in ability to associate with the telomeres could also be observed when the two isoforms were attached to different fluorescent tags and then simultaneously overexpressed in the same cells; a much smaller percentage of the nuclear GFP was incorporated into telomeric foci when GFP was attached to TIN2L rather than TIN2S (Fig S7).

While most of this decreased capping ability was likely due to the weaker association of GFP-TIN2L with the telomere, it is worth noting that in GFP-TIN2S expressing cells the uncapped telomeres have either equal or slightly lower than average TIN2 intensity (less TIN2 leads to a greater chance of association with 53BP1), while in GFP-TIN2L expressing cells the uncapped telomeres show slightly higher than average TIN2 (Fig 3D). We also observed that when we compared FRAP recovery speeds of various shelterins on capped vs artificially uncapped telomeres that GFP-TIN2L, but not GFP-TRF1 or GFP-TIN2S, showed slower recovery at uncapped telomeres (Fig S8). This may indicate that, similar to GFP-TIN2L's slightly reduced ability to recruit telomerase, GFP-TIN2L may also be somewhat worse at preventing association of DNA damage response proteins even when it is properly recruited to the telomere.

TIN2L phosphosite point mutants can increase shelterin recruitment to telomeres and improve capping ability

In order to dissect how TIN2 may be posttranscriptionally regulated, we made serine-to-alanine mutations at two potential phosphorylation sites- S270, the only ATM/ATR consensus site in TIN2, and S396, a residue present only in the TIN2L tail which is well conserved via ConSurf (34)(Fig S9) and is a consensus site for Casein Kinase II (CK2) (A. Bertuch, personal communication). When we examined the ability of these point mutants to promote telomere capping, we saw that all 3 point mutants of GFP-TIN2L examined (S270A, S396A, and the double S270A/S396A mutant) showed increased capping compared to the GFP-TIN2L WT(Fig 4A). No consistent difference was found between GFP-TIN2S WT and S270A variants. Consistent with increased capping ability, all the GFP-TIN2L point mutants showed an increased level of TRF1 recruitment to their telomeric foci (Fig S10A).

Given the increase in telomere capping and TRF1 recruitment, we were therefore surprised when we examined telomere recruitment of the GFP-TIN2L variants after shRNA treatment and saw that while all the point mutants showed slightly better recruitment only the GFP-TIN2L double point mutant S270A/S396A showed statistically significant improvement in its localization to telomeres (Fig 4B-C). However when we examined the distribution of TRF1 in the lines expressing different forms of TIN2, we found that GFP-TIN2L S270A had an extremely high ability to recruit TRF1 into telomeric foci (Fig 4D). As S270 is immediately downstream of the crystallized TIN2-TRF1 interaction domain, which ends at V268 (35), it is be ideally placed to modulate that interaction. Overexpression of TRF1 also allows GFP-TIN2L S396A to partially overcome its recruitment defect and to match the recruitment level of the GFP-TIN2L S270A/S396A double point mutant (Fig S10B), further strengthening the case that the S270A mutation functions to increase the interaction of the telomere with GFP-TIN2L via an enhanced TRF1 interaction. We therefore favor a model where both the S270 and S396

phosphorylation sites are used in separate pathways *in vivo* to regulate the amount of TIN2L present at the telomere- S270 via regulation of TRF1 affinity and S396 via decreasing ability to localize to telomeres in a way that has yet to be determined.

DISCUSSION

In this report, we have shown in UM-UC-3 cells that overexpressed GFP-TIN2S and GFP-TIN2L are equally competent at associating with telomeres and allowing telomerase access when overexpressed. Upon telomere elongation GFP-TIN2S and GFP-TIN2L both increase their association with the telomere; this stands in contrast to previous results in HMECs (14), which showed an additional recruitment of only TIN2S but not TIN2L after telomere elongation. This discrepancy is analogous to previous work on TRF1, which found after artificial elongation of the telomeres in cancer cells and primary cells that the cancer cells were more efficient at recruiting additional TRF1 to the telomere and showed increased association of TRF1 with the nuclear matrix (36). These results together imply that the regulation of TIN2 isoform expression and/or association in primary cells versus cancer cells may be fruitful ground for future experiments.

GFP-TIN2L shows significantly less recruitment to the telomere than GFP-TIN2S does when that overexpression is counteracted with a TIN2 shRNA that reduces its levels to near-WT amount or when tagged versions of both isoforms are introduced into the same cells and forced to compete for finite numbers of binding sites. This lack of association at the telomere explains why GFP-TIN2L is significantly worse than GFP-TIN2S at capping the telomere when expressed at comparable low levels; previous studies that removed TIN2 from the telomere

(by disrupting its association with TRF1) showed decreased levels of TRF2 bound at the telomere as well (37).

The presence of a well conserved CK2 phosphorylation site at S396 in the TIN2L tail implied that TIN2L specifically may be posttranslationally regulated; we have shown here that mutating that residue causes increased association with the telomere and more robust capping. CK2 has a wide variety of roles in the cell, including DNA damage-specific phosphorylation of p53 and XRCC1 (38, 39) as well as phosphorylation of BRCA1 (40). If telomeres, like 53BP1, are released from the nuclear matrix upon uncapping, phosphorylation of a residue in the domain that confers nuclear matrix binding by a kinase known to be triggered by DNA damage would certainly be a reasonable mechanism.

The ATM/ATR consensus phosphorylation site at S270 is slightly more puzzling due to its lesser degree of conservation and the lack of effect that its mutation has on GFP-TIN2S; S270 itself is well conserved but Q271, which dictates the consensus site, is not (Fig S9). The conservation of S270 itself may simply mean that in other organisms (such as mouse) a different kinase targets S270; alternately it may not be phosphorylated at all but may be structurally important for either the TRF1 binding site (which ends at V268), the HP1γ binding site/telomere syndrome hotspot (which begins at K280), or both. The difference in effect that it has on GFP-TIN2S vs GFP-TIN2L similarly has several possible explanations, none of which are mutually exclusive: TIN2S and TIN2L may have different tertiary structures which cause the residue to interact differently with one of the local functional motifs, the different localizations of TIN2S and TIN2L may mean that a spatially restricted kinase or phosphatase acts on one isoform and not the other, or there may in fact be an effect that is simply masked by the stronger interaction observed between WT TIN2S and TRF1.

Though the details vary, telomere tethering to the nuclear envelope is observed in budding yeast (41), fission yeast (42), and C elegans (43) ; this suggests though the mechanism and location of tethering vary that telomere attachment to a nuclear structure is a general phenomenon. Mutations in the nuclear matrix protein lamin A cause telomere shortening and the premature senescence associated with these mutations can be alleviated by overexpression of telomerase (44, 45), indicating that appropriate interaction with the nuclear matrix is important for telomere maintenance. The finding that the TIN2 isoform that is tethered to the nuclear matrix actually has a lower affinity for the telomere than the soluble isoform, coupled with the fact that this isoform has multiple potential sites of posttranslational modification, implies that modulation of the degree and timing of tethering is critical to the system. Further investigation of these dynamics may lead to increased understanding of the regulation of telomeres and the diseases associated with them.

ACKNOWLEDGEMENT

We would like to thank members of the Blackburn and Stohr labs for helpful discussions and for critical reading of the manuscript. We would like to acknowledge the use of the UCSF Nikon Imaging Center's microscopes and software and thank Kurt Thorn and DeLaine Larsen for guidance on microscopy techniques. We thank Alison Bertuch for valuable discussion and for sharing unpublished findings on TIN2.

FUNDING

This research was supported by the National Institutes of Health [CA96840] to EHB. Funding for open access charge: National Institutes of Health.

REFERENCES

1. Palm,W. and de Lange,T. (2008) How shelterin protects mammalian telomeres. *Annu. Rev. Genet.*, **42**, 301–334.

2. Baumann,P. and Cech,T.R. (2001) Pot1, the putative telomere end-binding protein in fission yeast and humans. *Science*, **292**, 1171–1175.

3. Liu,D., Safari,A., O'Connor,M.S., Chan,D.W., Laegeler,A., Qin,J. and Songyang,Z. (2004) PTOP interacts with POT1 and regulates its localization to telomeres. *Nat. Cell Biol.*, **6**, 673–680.

4. Ye,J.Z.-S., Hockemeyer,D., Krutchinsky,A.N., Loayza,D., Hooper,S.M., Chait,B.T. and de Lange,T. (2004) POT1-interacting protein PIP1: a telomere length regulator that recruits POT1 to the TIN2/TRF1 complex. *Genes Dev.*, **18**, 1649–1654.

5. Bilaud,T., Brun,C., Ancelin,K., Koering,C.E., Laroche,T. and Gilson,E. (1997) Telomeric localization of TRF2, a novel human telobox protein. *Nat. Genet.*, **17**, 236–239.

6. Li,B., Oestreich,S. and de Lange,T. (2000) Identification of human Rap1: implications for telomere evolution. *Cell*, **101**, 471–483.

7. Takai,H., Smogorzewska,A. and de Lange,T. (2003) DNA damage foci at dysfunctional telomeres. *Curr. Biol.*, **13**, 1549–1556.

8. Chong,L., van Steensel,B., Broccoli,D., Erdjument-Bromage,H., Hanish,J., Tempst,P. and de Lange,T. (1995) A human telomeric protein. *Science*, **270**, 1663–1667.

9. Kim,S.-H., Kaminker,P. and Campisi,J. (1999) TIN2 , a new regulator of telomere length in human cells. *Nature Genetics*, **23**, 405–412.

10. Sfeir,A., Kosiyatrakul,S.T., Hockemeyer,D., MacRae,S.L., Karlseder,J., Schildkraut,C.L. and de Lange,T. (2009) Mammalian telomeres resemble fragile sites and require TRF1 for efficient replication. *Cell*, **138**, 90–103.

11. Stewart,S. a. and Weinberg,R. a. (2006) Telomeres: cancer to human aging. *Annu. Rev. Cell Dev. Biol.*, **22**, 531–557.

12. Armanios,M. and Blackburn,E.H. (2012) The telomere syndromes. *Nat. Rev. Genet.*, **13**, 693–704.

13. Savage,S. a., Giri,N., Baerlocher,G.M., Orr,N., Lansdorp,P.M. and Alter,B.P. (2008) TINF2, a Component of the Shelterin Telomere Protection Complex, Is Mutated in Dyskeratosis Congenita. *Am. J. Hum. Genet.*, **82**, 501–509.

14. Kaminker,P.G., Kim,S.H., Desprez,P.Y. and Campisi,J. (2009) A novel form of the telomere-associated protein TIN2 localizes to the nuclear matrix. *Cell Cycle*, **8**, 1–9.

15. de Lange,T. (1992) Human telomeres are attached to the nuclear matrix. *EMBO J.*, **11**, 717–724.

16. Ludérus,M.E.E., Van Steensel,B., Chong,L., Sibon,O.C.M., Cremers,F.F.M. and De Lange,T. (1996) Structure, subnuclear distribution, and nuclear matrix association of the mammalian telomeric complex. *J. Cell Biol.*, **135**, 867–881.

17. Smilenov,L.B., Dhar,S. and Pandita,T.K. (1999) Altered Telomere Nuclear Matrix Interactions and Nucleosomal Periodicity in Ataxia Telangiectasia Cells before and after Ionizing Radiation Treatment. *Molecular and Cellular Biology*, **19**, 6963–6971.

18. Wang,X., Kam,Z., Carlton,P.M., Xu,L., Sedat,J.W. and Blackburn,E.H. (2008) Rapid telomere motions in live human cells analyzed by highly time-resolved microscopy. *Epigenetics Chromatin*, **1**, 4.

19. Dimitrova,N., Chen,Y.-C.M., Spector,D.L., de Lange,T. and Lange,T.D. (2008) 53BP1 promotes non-homologous end joining of telomeres by increasing chromatin mobility. *Nature*, **456**, 524–528.

20. Gibbs-Seymour,I., Markiewicz,E., Bekker-Jensen,S., Mailand,N. and Hutchison,C.J. (2015) Lamin A/C-dependent interaction with 53BP1 promotes cellular responses to DNA damage. *Aging Cell*, **14**, 162–169.

21. Zufferey,R., Nagy,D., Mandel,R.J., Naldini,L. and Trono,D. (1997) Multiply attenuated lentiviral vector achieves efficient gene delivery in vivo. *Nat. Biotechnol.*, **15**, 871–875.

22. Diolaiti,M.E., Cimini,B.A., Kageyama,R., Charles,F.A. and Stohr,B.A. (2013) In situ visualization of telomere elongation patterns in human cells. *Nucleic Acids Res.*, **41**, e176.

23. Li,S., Rosenberg,J.E., Donjacour,A. a., Botchkina,I.L., Hom,Y.K., Cunha,G.R. and Blackburn,E.H. (2004) Rapid inhibition of cancer cell growth induced by lentiviral delivery and expression of mutant-template telomerase RNA and anti-telomerase short-interfering RNA. *Cancer Res.*, **64**, 4833–4840.

24. Gibson,D.G., Young,L., Chuang,R.-Y., Venter,J.C., Hutchison,C.A.,3rd and Smith,H.O. (2009) Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat. Methods*, **6**, 343–345.

25. Schindelin,J., Arganda-Carreras,I., Frise,E., Kaynig,V., Longair,M., Pietzsch,T., Preibisch,S., Rueden,C., Saalfeld,S., Schmid,B., *et al.* (2012) Fiji: an open-source platform for biological-image analysis. *Nat. Methods*, **9**, 676–682.

26. Kamentsky,L., Jones,T.R., Fraser,A., Bray,M.-A., Logan,D.J., Madden,K.L., Ljosa,V., Rueden,C., Eliceiri,K.W. and Carpenter,A.E. (2011) Improved structure, function and compatibility for CellProfiler: modular high-throughput image analysis software. *Bioinformatics*, **27**, 1179–1180.

27. Listerman,I., Gazzaniga,F.S. and Blackburn,E.H. (2014) An investigation of the effects of the core protein telomerase reverse transcriptase on Wnt signaling in breast cancer cells. *Mol. Cell. Biol.*, **34**, 280–289.

28. Chen,L.Y., Zhang,Y., Zhang,Q., Li,H., Luo,Z., Fang,H., Kim,S.H., Qin,L., Yotnda,P., Xu,J., *et al.* (2012) Mitochondrial Localization of Telomeric Protein TIN2 Links Telomere Regulation to Metabolic Control. *Mol. Cell*, **47**, 839–850.

29. Kim,S.H., Davalos,A.R., Heo,S.J., Rodier,F., Zou,Y., Beausejour,C., Kaminker,P., Yannone,S.M. and Campisi,J. (2008) Telomere dysfunction and cell survival: Roles for distinct TIN2-containing complexes. *J. Cell Biol.*, **181**, 447–460.

30. Cesare,A., Hayashi,M., Crabbe,L. and Karlseder,J. (2013) The Telomere deprotection response is functionally distinct from the Genomic DNA damage response. *Mol. Cell*, **51**, 141–155.

31. Stohr,B.A. and Blackburn,E.H. (2008) ATM mediates cytotoxicity of a mutant telomerase RNA in human cancer cells. *Cancer Res.*, **68**, 5309–5317.

32. Takai,K.K., Kibe,T., Donigian,J.R., Frescas,D. and de Lange,T. (2011) Telomere protection by TPP1/POT1 requires tethering to TIN2. *Mol. Cell*, **44**, 647–659.

33. Abreu,E., Aritonovska,E., Reichenbach,P., Cristofari,G., Culp,B., Terns,R.M., Lingner,J. and Terns,M.P. (2010) TIN2-tethered TPP1 recruits human telomerase to telomeres in vivo. *Mol. Cell. Biol.*, **30**, 2971–2982.

34. Ashkenazy,H., Erez,E., Martz,E., Pupko,T. and Ben-Tal,N. (2010) ConSurf 2010: calculating evolutionary conservation in sequence and structure of proteins and nucleic acids. Nucleic Acids Res., 38, W529–33.

35. Chen,Y., Yang,Y., van Overbeek,M., Donigian,J.R., Baciu,P., de Lange,T. and Lei,M. (2008) A shared docking motif in TRF1 and TRF2 used for differential recruitment of telomeric proteins. Science, 319, 1092–1096.

36. Okabe,J., Eguchi,A., Wadhwa,R., Rakwal,R., Tsukinoki,R., Hayakawa,T. and Nakanishi,M. (2004) Limited capacity of the nuclear matrix to bind telomere repeat binding factor TRF1 may restrict the proliferation of mortal human fibroblasts. Hum. Mol. Genet., 13, 285–293.

37. Frescas,D. and de Lange,T. (2014) TRF2-tethered TIN2 can mediate telomere protection by TPP1/POT1. Mol. Cell. Biol., 34, 1349–1362.

38. Keller,D.M., Zeng,X., Wang,Y., Zhang,Q.H., Kapoor,M., Shu,H., Goodman,R., Lozano,G., Zhao,Y. and Lu,H. (2001) A DNA damage-induced p53 serine 392 kinase complex contains CK2, hSpt16, and SSRP1. Mol. Cell, 7, 283–292.

39. Loizou,J.I., El-Khamisy,S.F., Zlatanou,A., Moore,D.J., Chan,D.W., Qin,J., Sarno,S., Meggio,F., Pinna,L.A. and Caldecott,K.W. (2004) The protein kinase CK2 facilitates repair of chromosomal DNA single-strand breaks. Cell, 117, 17–28.

40. O'Brien,K.A., Lemke,S.J., Cocke,K.S., Rao,R.N. and Beckmann,R.P. (1999) Casein kinase 2 binds to and phosphorylates BRCA1. Biochem. Biophys. Res. Commun., 260, 658–664.

41. Hediger,F., Florence,H., Neumann,F.R., Van Houwe,G., Karine,D. and Gasser,S.M. (2002) Live Imaging of Telomeres. Curr. Biol., 12, 2076–2089.

42. Chikashige,Y., Haraguchi,T. and Hiraoka,Y. (2010) Nuclear envelope attachment is not necessary for telomere function in fission yeast. Nucleus, 1, 481–486.

43. Ferreira,H.C., Towbin,B.D., Jegou,T. and Gasser,S.M. (2013) The shelterin protein POT-1 anchors Caenorhabditis elegans telomeres through SUN-1 at the nuclear periphery. J. Cell Biol., 203, 727–735.

44. Benson,E.K., Lee,S.W. and Aaronson,S. a. (2010) Role of progerin-induced telomere dysfunction in HGPS premature cellular senescence. J. Cell Sci., 123, 2605–2612.

45. Chojnowski,A., Ong,P.F., Wong,E.S.M., Lim,J.S.Y., Mutalif,R.A., Navasankari,R., Dutta,B., Yang,H., Liow,Y.Y., Sze,S.K., et al. (2015) Progerin reduces LAP2α-telomere association in Hutchinson-Gilford progeria. Elife, 4, e07759.

FIGURE LEGENDS

Figure 1 - TIN2 shRNAs 1 and 4 are effective against endogenous and overexpressed TIN2. A) Map of TIN2S and TIN2L detailing target sites for the various shRNAs. B) IF for 53BP1 (green) and FISH for telomeric repeats (magenta) demonstrates induction of TIFs after TIN2 shRNA infection. Scale bar=10μm. C) Quantification of the percentage of 53BP1-associated telomeres in each cell shows large induction of TIFs by TIN2 shRNA1 and TIN2 shRNA4. D) Western blotting for TIN2 shows that TIN2S is the major splice form isolated and that GFP-TIN2S and GFP-TIN2L are expressed at equivalent levels and knocked down equally.

Figure 2 - GFP-TIN2S and GFP-TIN2L are capable of recruiting telomerase and elongating telomeres. A) Demonstration of TSQ1 assay for telomeres (green) and TSQ1 expression (magenta). Scale bar = 10 μm. B) Measurement of the percentage of telomeres in each nucleus that colocalize with TSQ1 foci shows overexpression of GFP-TIN2S and GFP-TIN2L do not significantly affect TSQ1 incorporation at telomeres. C) Analysis of telomere length with PNA-FISH shows that GFP-TIN2S and GFP-TIN2L expressing cells both elongate their telomeres after expression of hTR. D) Analysis of GFP-TRF1, GFP-TIN2S, and GFP-TIN2L telomere spot intensities show that all the shelterins show increases in foci brightness after telomere elongation with hTR.

Figure 3 - TIN2L has a capping defect after knockdown and shows decreased telomere recruitment. A) IF-FISH analysis of the percentage of telomeres in each nucleus colocalizing with 53BP1 shows that GFP-TIN2L is less competent at protecting telomeres after expression of TIN2 shRNAs. B-C) GFP-TIN2L shows much less concentration into telomeric

foci than GFP-TIN2S in cells expressing TIN2 shRNAs. Scale bar=10μm. D) Measurement of mApple-TRF1 telomeres shows no large difference in the amount of GFP-TIN2 present at uncapped telomeres versus the amount present across all telomeres.

Figure 4 - TIN2L phosphosite mutants partially restore shelterin recruitment and capping ability. A) IF-FISH TIF analysis shows that all GFP-TIN2L point mutants are better at telomere capping after TIN2 shRNA1 treatment than the GFP-TIN2L WT. B-C) GFP-TIN2L S396A point mutants (particularly the S270A/S396A double point mutant) show significantly better recruitment to telomeres after TIN2 shRNA treatment. D) Measurement of the ability of mApple-TRF1 to form telomere foci in the presence of various forms of GFP-TIN2 shows that while mutation of S270A has no effect on GFP-TIN2S, its mutation in GFP-TIN2L significantly improves telomeric recruitment.

Figure S1-Quantification of TIN2 knockdown via GFP-TIN2 fluorescence. Measurement of the total nuclear GFP intensity of GFP-TIN2S (A) and GFP-TIN2L (B) show that all 4 shRNAs show a substantial decrease in the measured GFP-shelterin expressed.

Figure S2-TIN2 shRNAs 1 and 4 do not slow cell growth or induce substantial caspase 3/7 activity. A) Growth curves show that despite the large increase in TIFs caused by TIN2 shRNA1 and TIN2 shRNA4, cell growth is indistinguishable from scramble control. B) Measurement of caspase 3/7 activation at day 7 shows that TIN2 shRNAs 1 and 4 show a very slight level of caspase activation but far less than the level induced by introduction of the 47A mutant telomerase.

Figure S3-TIN2 knockdown causes less telomere colocalization with pATM than uncapping via the hTR mutant 47A. Simultaneous measurement of the percentage of

telomeres in each nucleus that colocalize with 53BP1 (A) and pATM (B) shows that while TIN2 shRNA1 causes a comparable amount of 53BP1 activation as 47A the percentage of telomeres that are pATM positive is much lower, indicating that the telomere uncapping response in these cells is driven primarily through ATR activation.

Figure S4- Overexpressed GFP-TIN2S and GFP-TIN2L are equally recruited to telomeres. A) Illustration of the identification of nuclei (yellow) and telomeres (red) in a GFP-TIN2L expressing cell. B-E)GFP-TIN2S and GFP-TIN2L show comparable distribution in mean total nuclear GFP (B), number of telomeres identified per cell (C), the brightness of each individual telomere (D), or the percentage of the total nuclear GFP signal contained within the sum of the telomere spots (E).

Figure S5- TIN2 shRNA1 decreases the cell's ability to incorporate TSQ1 at telomeres. Measurement of the percentage of telomeres incorporating TSQ1 in each nucleus shows a significant reduction in the number of TSQ1-postitive telomeres both in cells that express GFP-TIN2S and GFP-TIN2L.

Figure S6- TRF1 telomeres recruit less TIN2L than TIN2S after TIN2 shRNA treatment despite similar nuclear expression levels A) Measurement of total nuclear GFP intensity in cells expressing GFP-TIN2 and mApple-TRF1 shows that GFP-TIN2S and GFP-TIN2L show comparable expression levels across isoforms after TIN2 shRNA treatment.
B) Measurement of the normalized GFP intensity of mApple-TRF1 telomeres shows much less GFP-TIN2L present on telomeres than GFP-TIN2S after TIN2 shRNA treatment.

Figure S7- TIN2L shows impaired recruitment to telomeres when co-expressed with TIN2S. Cells expressing either GFP-TIN2S or GFP-TIN2L had an mApple version of the

other TIN2 isoform added and FACS sorting was carried out to ensure comparable expression levels of both fluorescent proteins. GFP-TIN2L shows much less accumulation into foci than GFP-TIN2S does when both isoforms are simultaneously overexpressed and forced to compete for binding sites.

Figure S8-GFP-TIN2L shows slower FRAP recovery after telomere uncapping. Analysis of FRAP recovery of various GFP-tagged shelterins shows that while the $t_{1/2}$ of recovery for GFP-TRF1 and GFP-TIN2S do not change in cells with induced telomere damage vs control cells, GFP-TIN2L telomeres show a much slower recovery when telomere damage has been induced. GFP-TRF1 also shows a significantly increased recovery rate in cells with TIN2 knockdown, consistent with TIN2's role in stabilizing TRF1.

Figure S9-Conservation of individual residues in TIN2. Conservation analysis of individual residues in TIN2 shows good conservation in the known TPP1 and TRF1 binding site as well as S270 and S396.

Figure S10- Interaction of overexpressed TRF1 with GFP-TIN2 point mutants. A) Quantification of the amount of mApple-TRF1 present at all GFP-TIN2 telomeres shows that GFP-TIN2L WT recruits less TRF1 than GFP-TIN2S but all GFP-TIN2L point mutants recruit more than GFP-TIN2S. B) When mApple-TRF1 is overexpressed in cells with GFP-TIN2, the S396A and S270A/S396A GFP-TIN2L mutants now both show a partial rescue in ability to localize to telomeres after TIN2shRNA4 treatment.

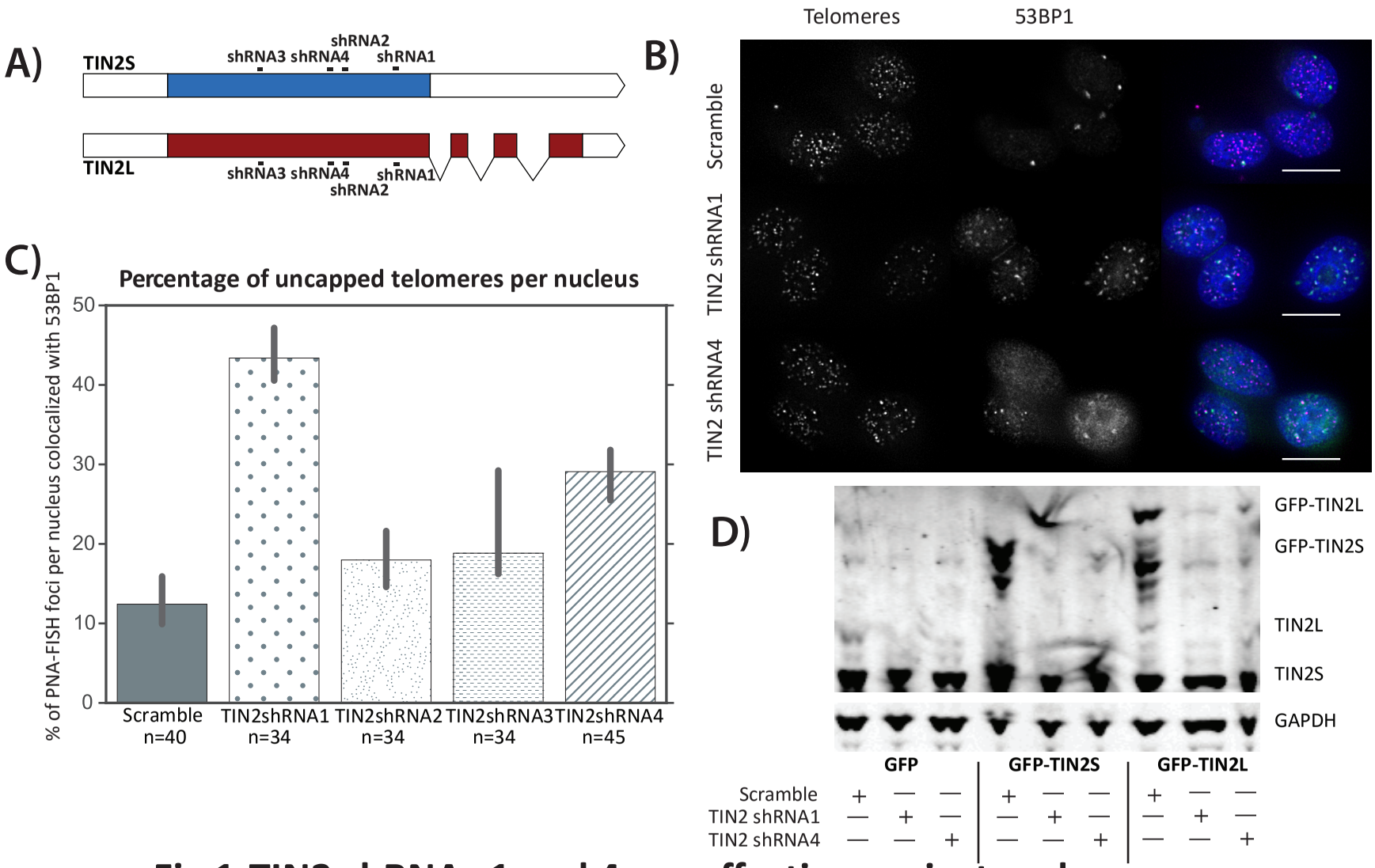


Fig 1-TIN2 shRNAs 1 and 4 are effective against endogenous and overexpressed TIN2

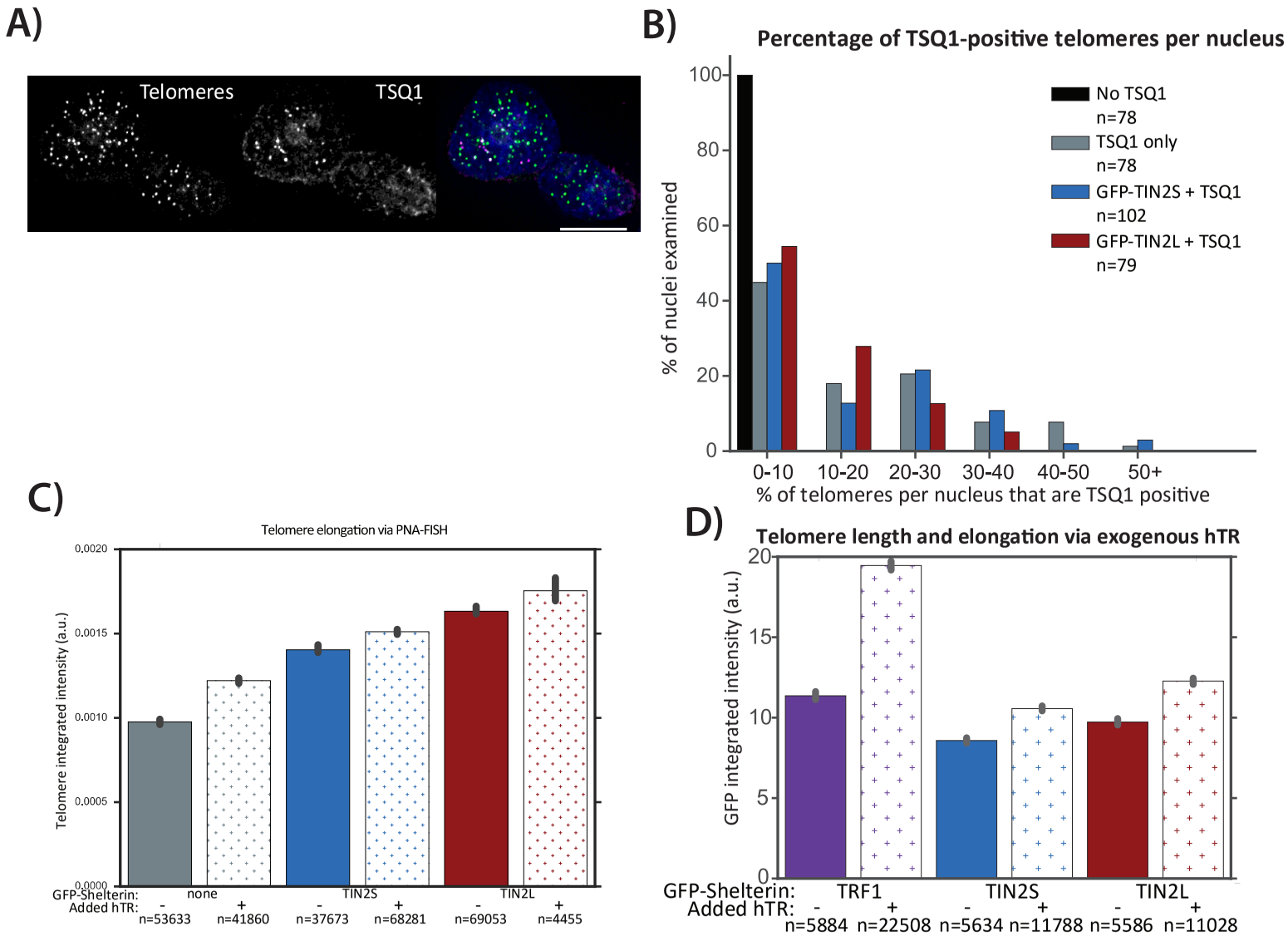


Fig 2-GFP-TIN2S and GFP-TIN2L are capable of recruiting telomerase and elongating telomeres

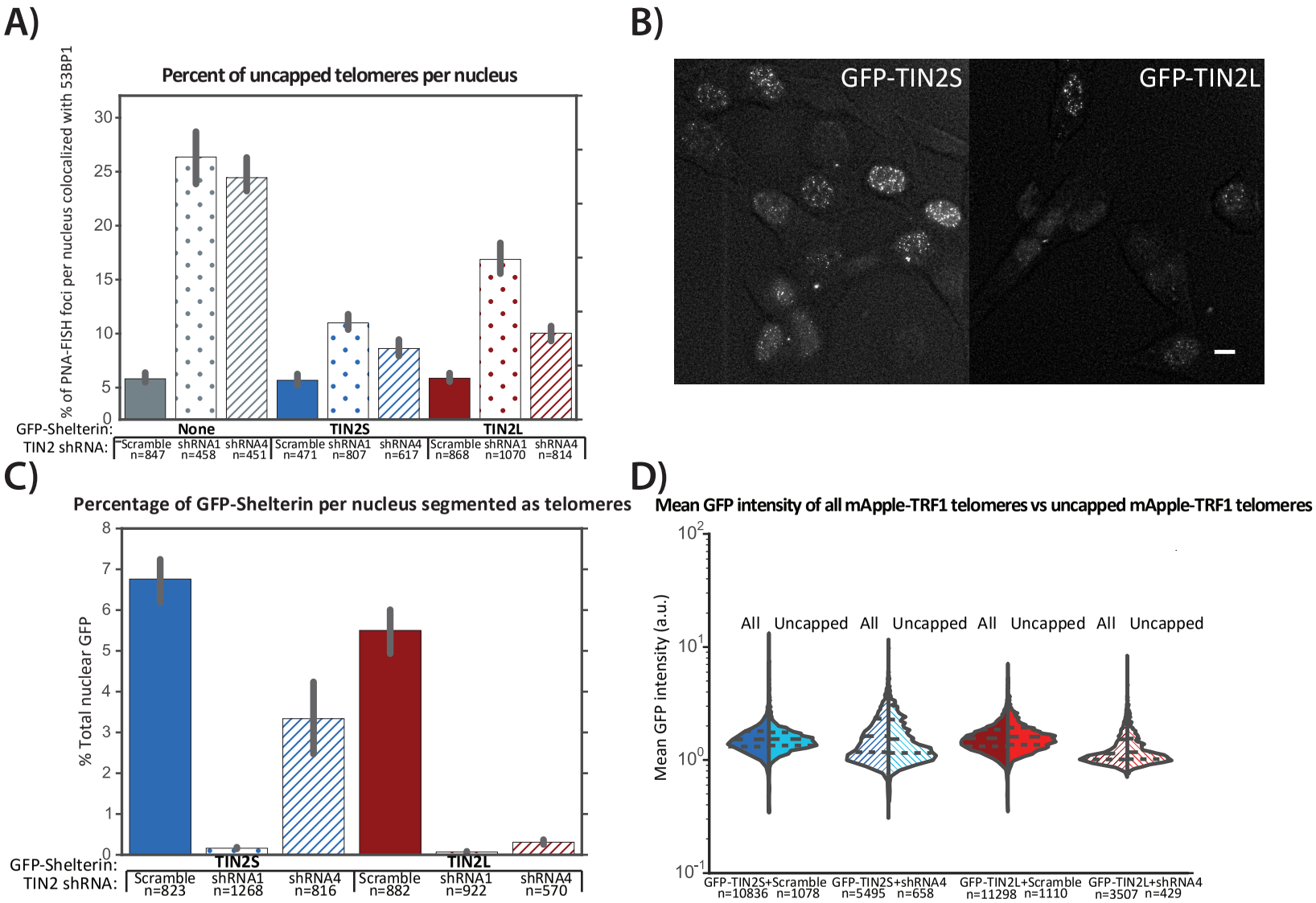


Fig 3- TIN2L has a capping defect after knockdown and shows decreased telomere recruitment

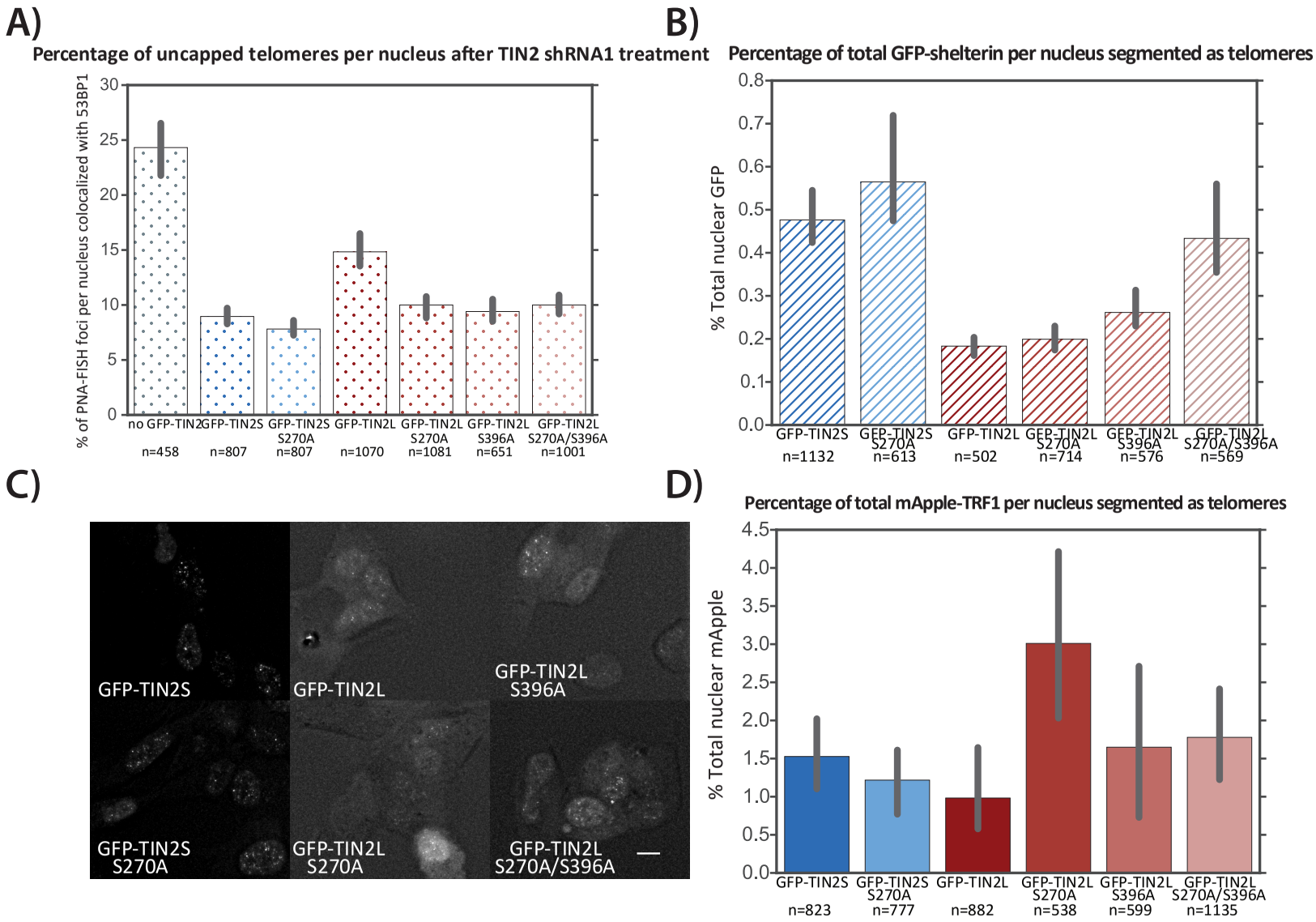
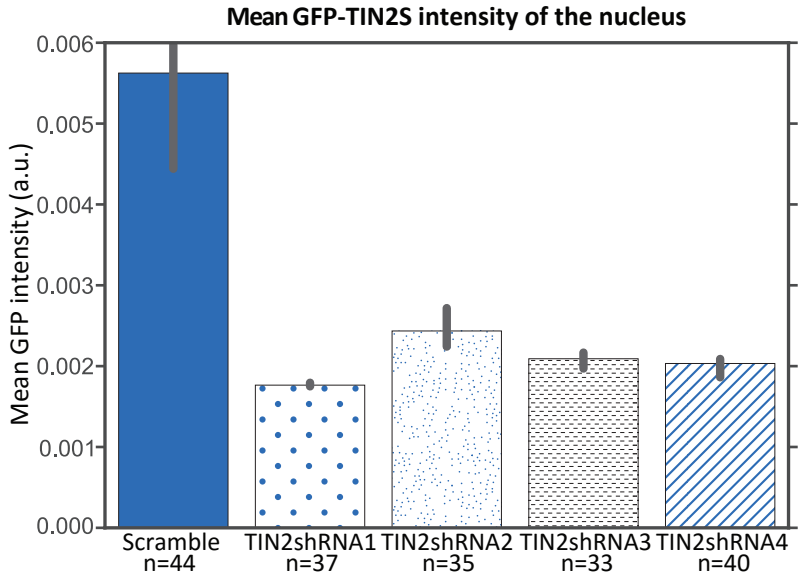
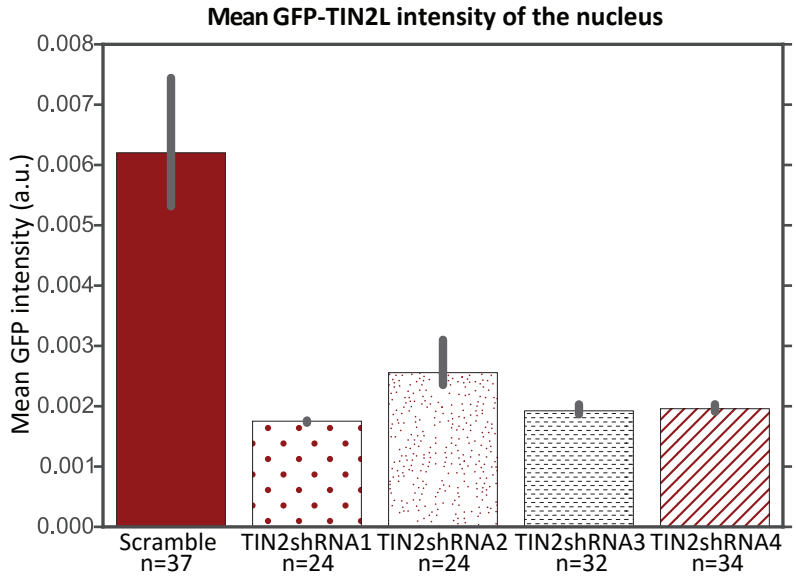


Fig 4- TIN2L phosphosite mutants partially restore shelterin recruitment and capping ability

A)

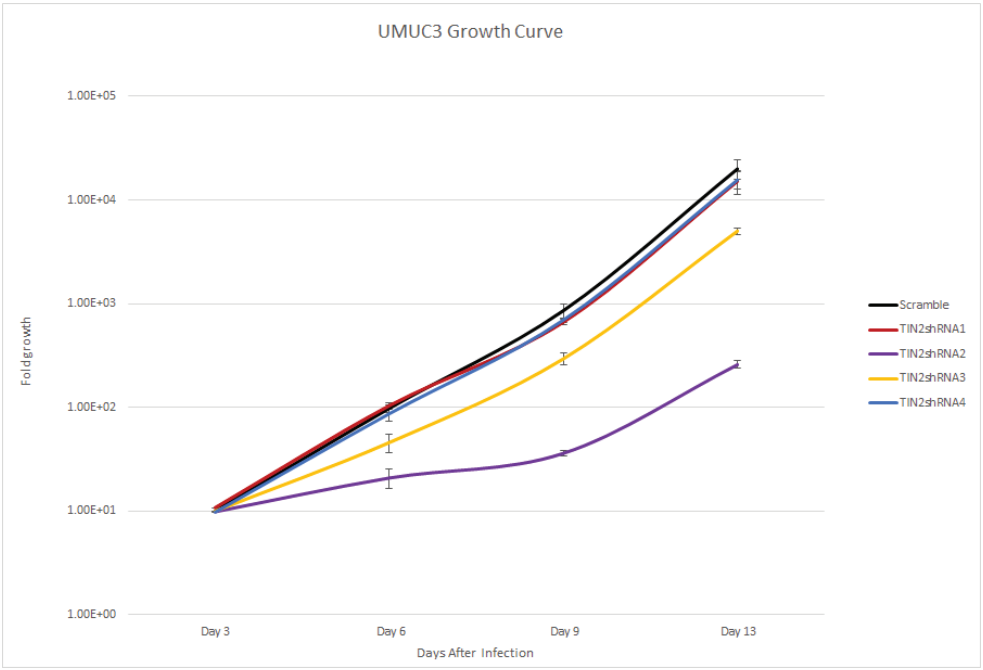


B)

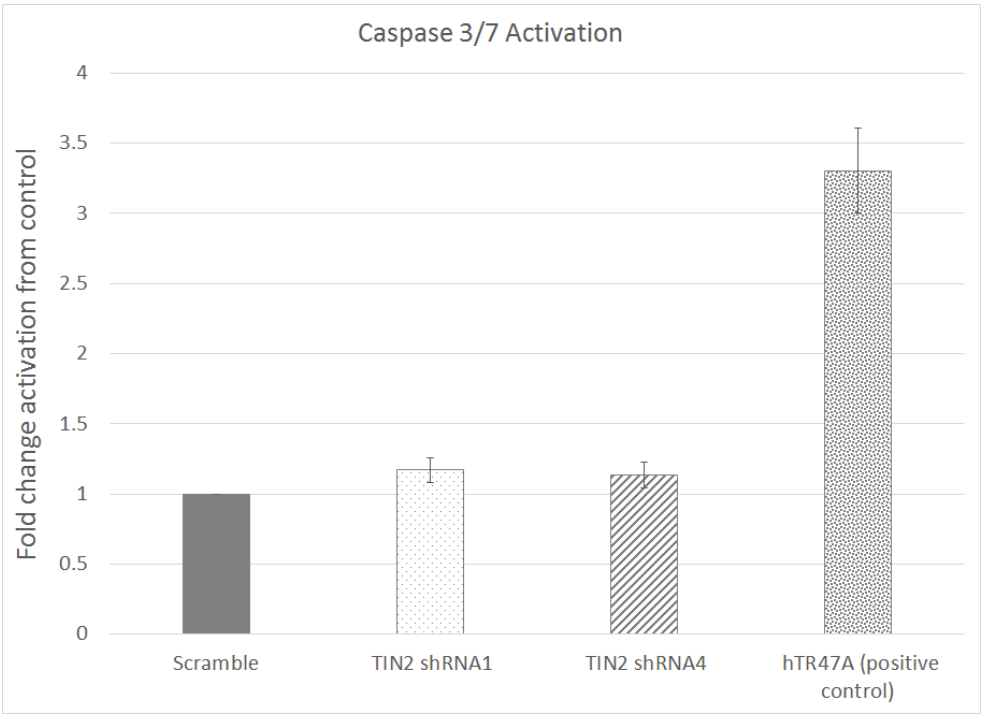


S1-Quantification of TIN2 knockdown via GFP-TIN2 fluorescence

A)

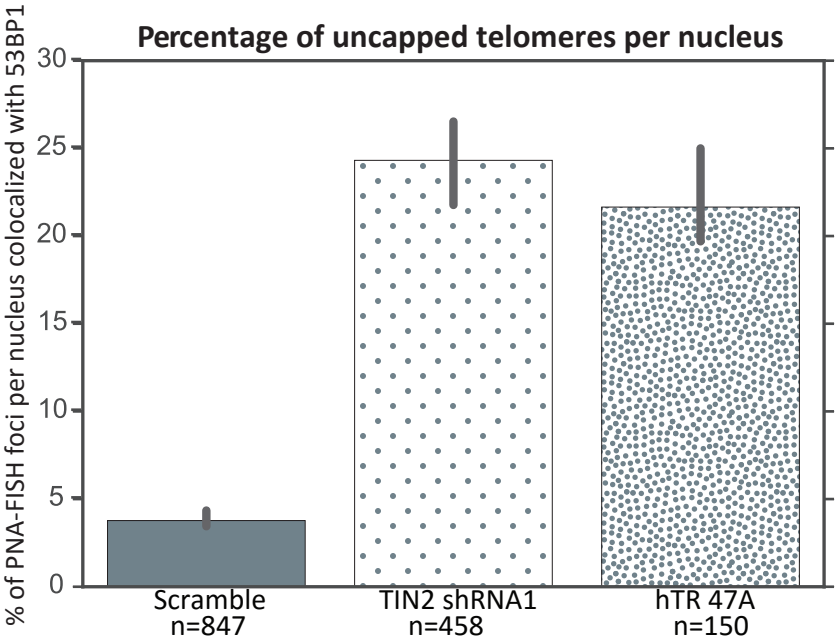


B)

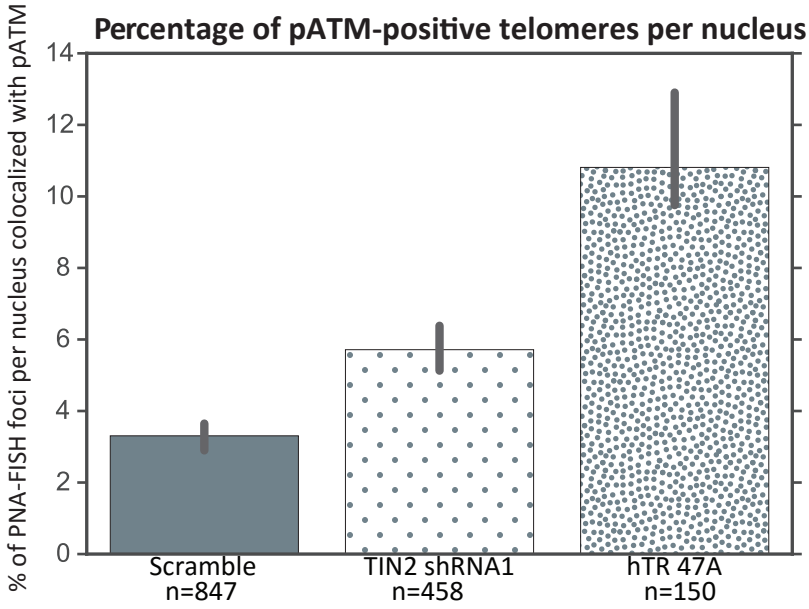


S2-TIN2 shRNAs 1 and 4 do not slow cell growth or induce substantial caspase 3/7 activity

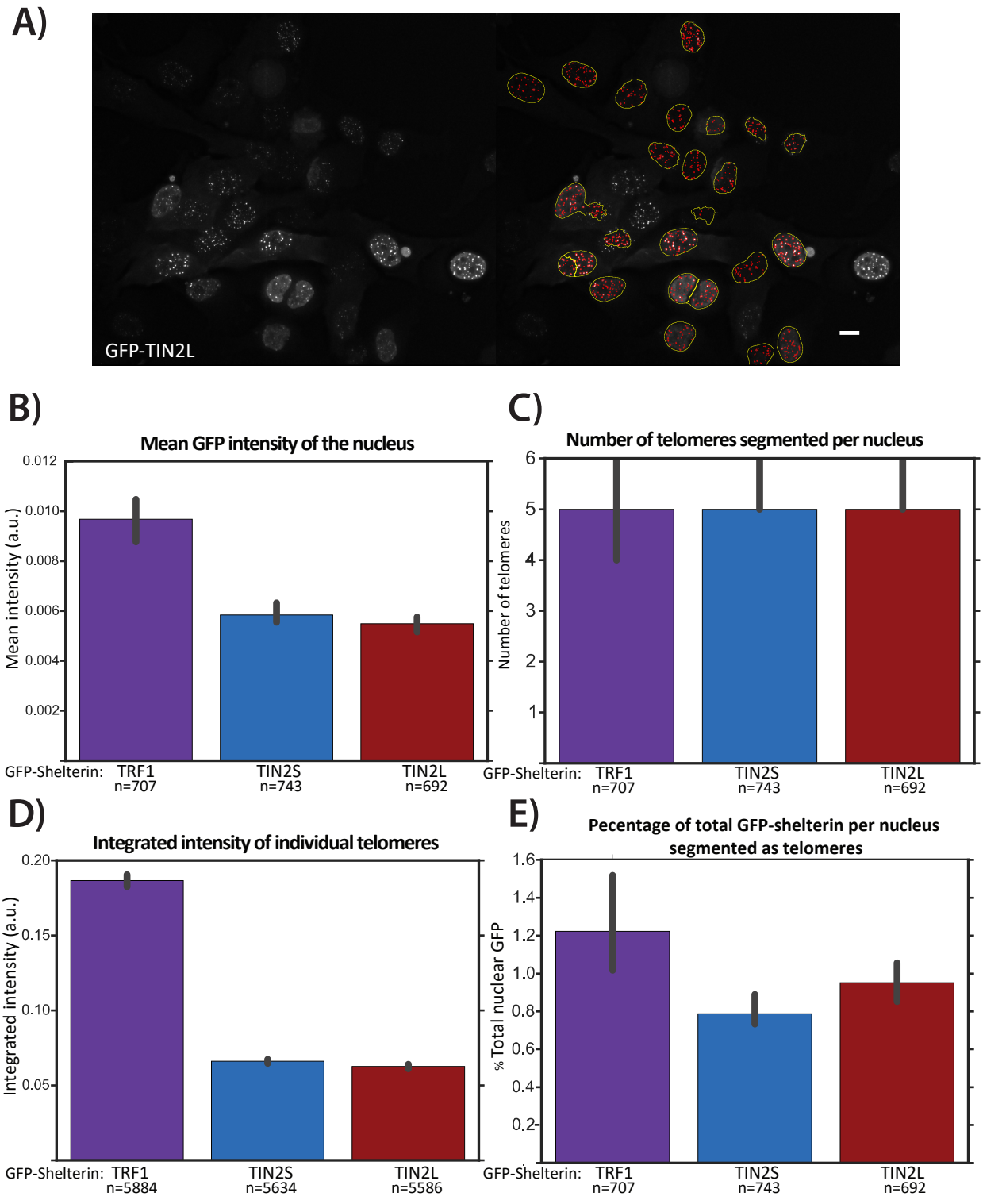
A)



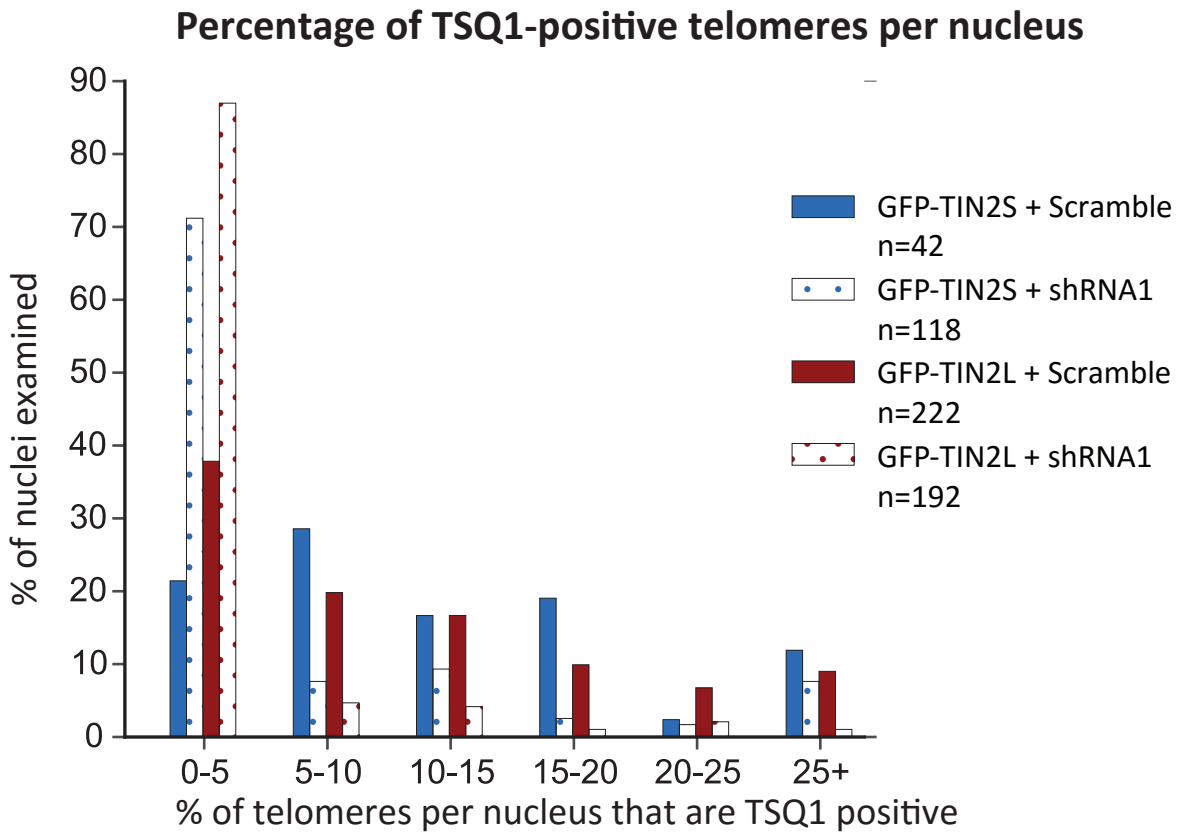
B)



S3-TIN2 knockdown causes less telomere colocalization with pATM than uncapping via the hTR mutant 47A

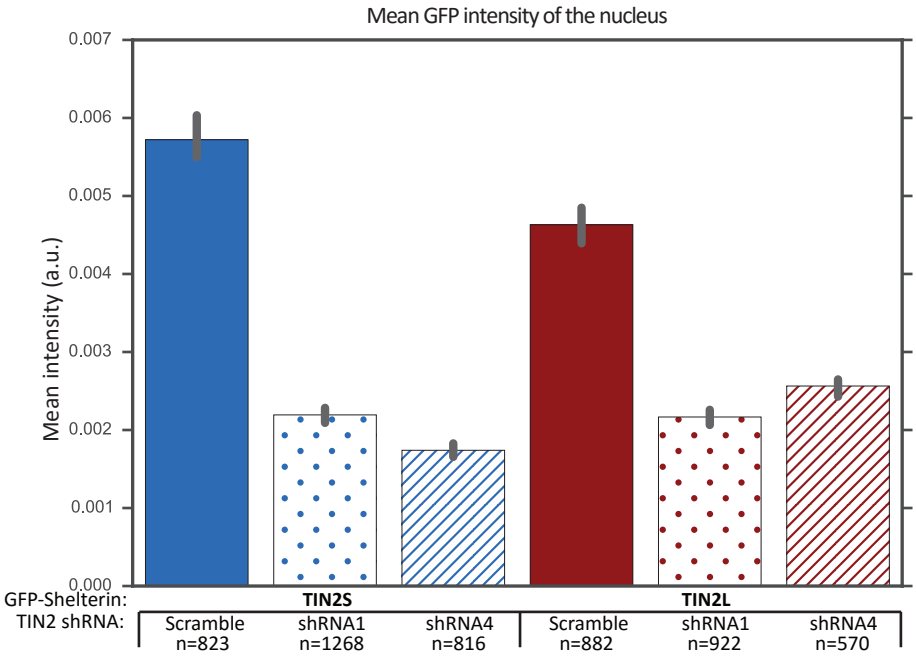


S4- Overexpressed GFP-TIN2S and GFP-TIN2L are equally recruited to telomeres

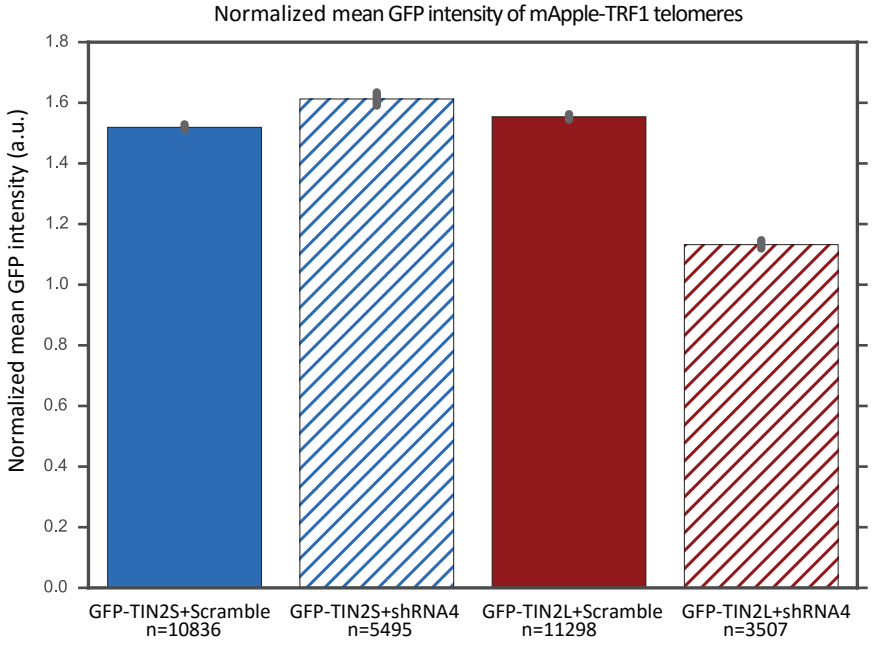


S5- TIN2 shRNA1 decreases the cell's ability to incorporate TSQ1 at telomeres

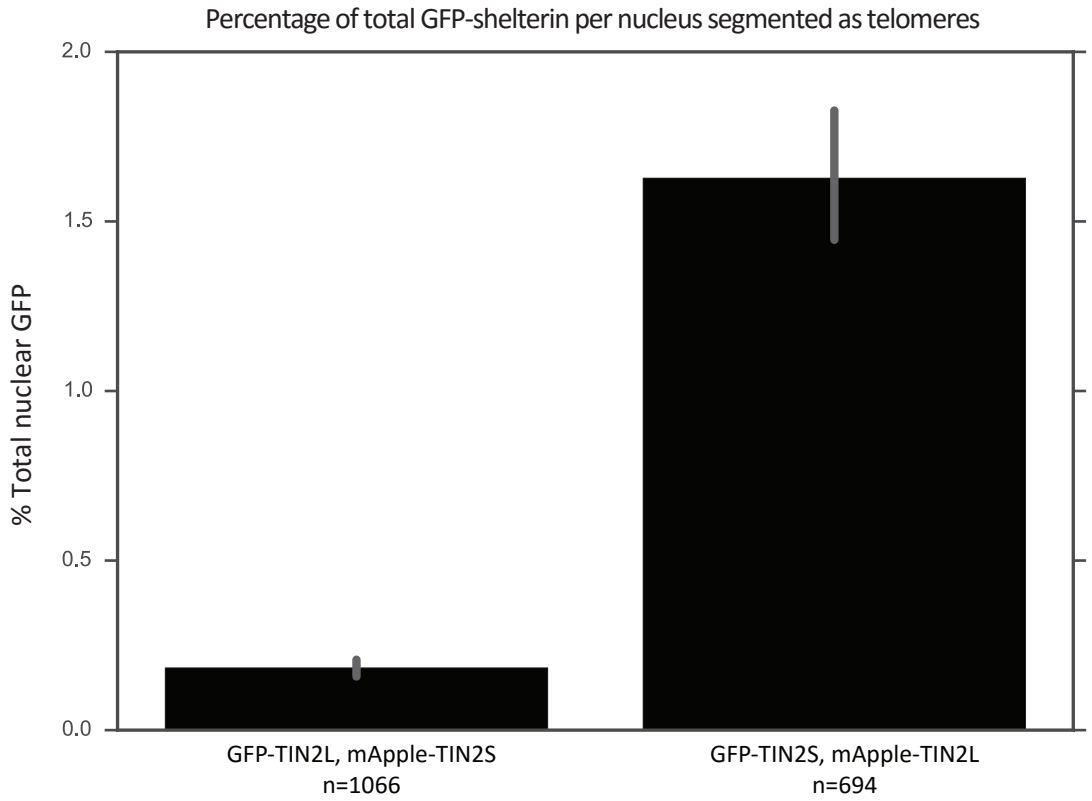
A)



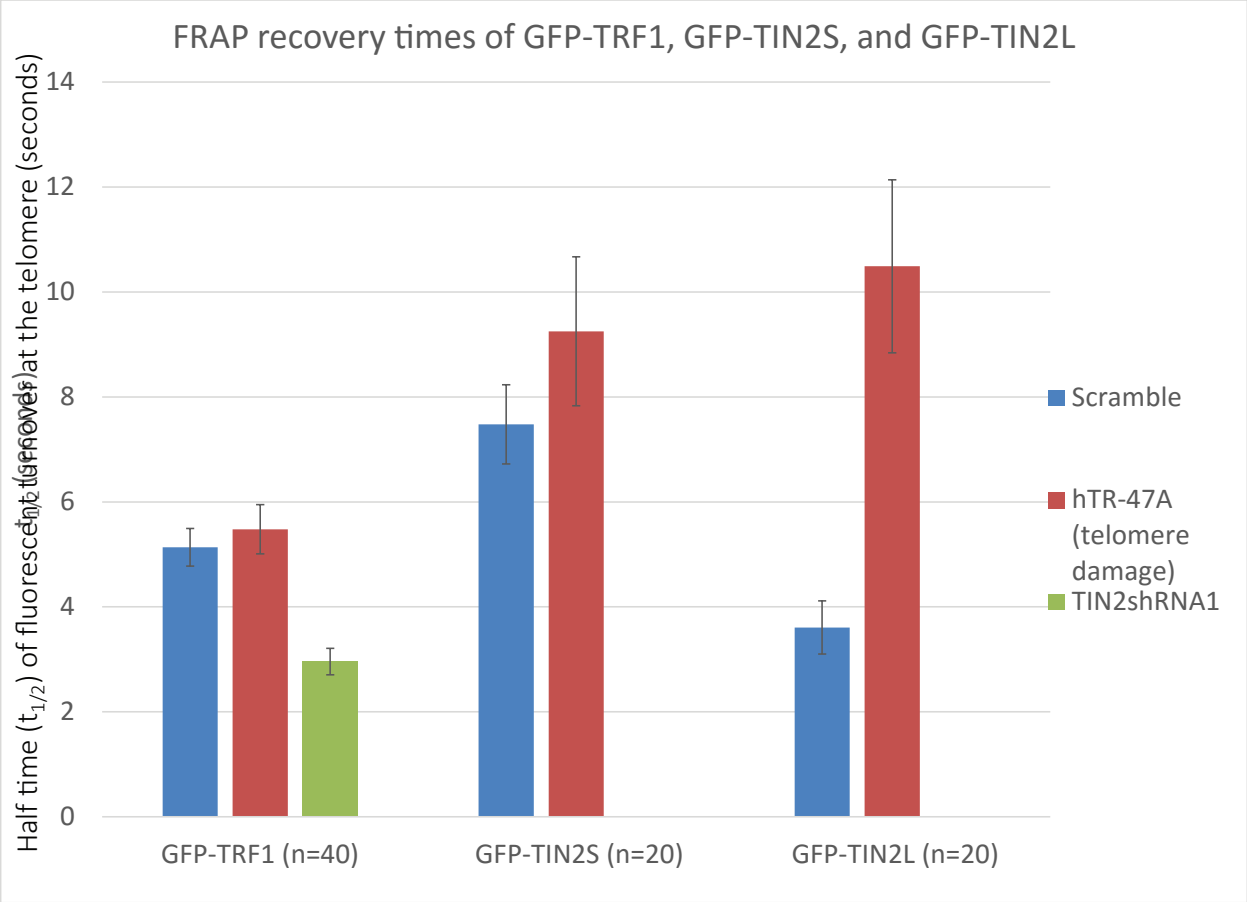
B)



S6- TRF1 telomeres recruit less TIN2L than TIN2S after TIN2 shRNA treatment despite similar nuclear expression levels



S7- TIN2L shows impaired recruitment to telomeres when co-expressed with TIN2S

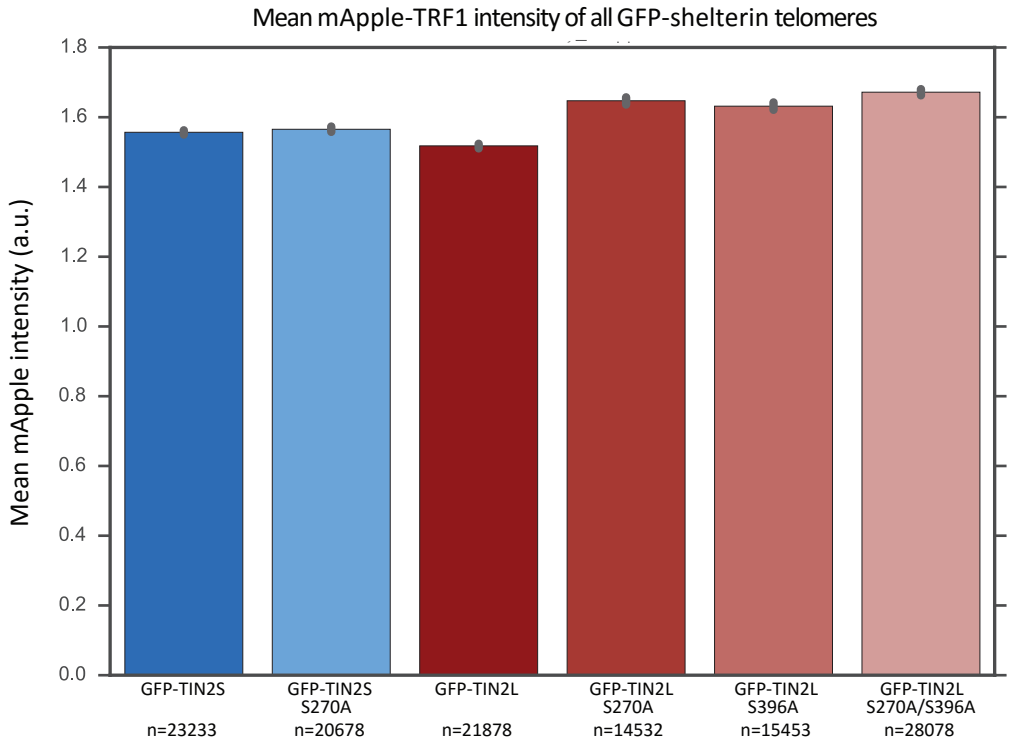


S8-GFP-TIN2L shows slower FRAP recovery after telomere uncapping

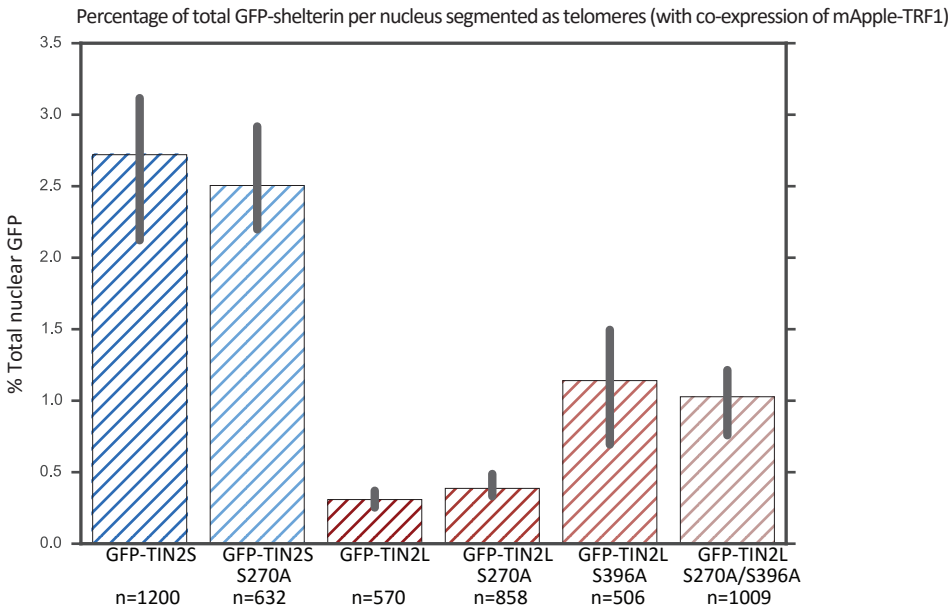
ConSurf Results

S9-Conservation of individual residues in TIN2

A)



B)



S10- Interaction of overexpressed TRF1 with GFP-TIN2 point mutants

CellProfilerStats simplifies and automates analysis of CellProfiler data and opens deep image mining to a broader sector of biologists

Abstract

Summary: As biology moves firmly into the age of high throughput screening, knowledge of how to handle large data sets has not kept pace. One area in which this is true is in image analysis, where a typical experiment may include several dozen parameters and several hundred images. CellProfilerStats is designed as a backend to the popular image analysis program CellProfiler and allows easy generation of graphs and statistics along with interrogation of specific subpopulations in an easily installed package and without the need for any ability to program by the end-user.

Availability and implementation: Source code with list of dependencies along with Windows and Mac executables can be found at https://github.com/bethac07/CellProfilerStats/releases

Contact: beth.cimini@ucsf.edu

1 Introduction

As high content screening becomes cheaper, researchers are tasked with analyzing larger data sets than even before (Perkel, 2016). Unfortunately, many biologists lack the training to properly analyze such data sets (Smolinski, 2010)- when the National Science Foundation solicited ideas for improving STEM graduate education in 2013, teams from multiple disciplines identified a need for increased training in programming and large data sets (http://www.nsf.gov/news/special_reports/gradchallenge/), indicating that the problem is both far-reaching and far from being solved.

Image analysis can easily create unexpectedly large data sets, particularly as automated plate-scanning microscopes become more common. A recent meta-analysis of large microscopy screens showed that 60-80% of studies reported measurements of only one or two features of the cells (Singh *et al.*, 2014), despite the large number of measurable parameters in image data and that increasing the number of features can increase accuracy of classification(Loo *et al.*, 2007). Much of the data from these screens is therefore being "left on the table", in some cases likely due to lack of knowledge of how to better mine the results.

CellProfiler is a popular open source image analysis program that can measure large numbers of features and operate on large image sets (Kamentsky *et al.*, 2011); it incorporates a flexible module structure so that users can customize their analysis. From the GUI users can add any number of steps to their analysis "pipeline", perform measurements, then export the measurements to SQL, MatLab, or .csv files. All of these steps can be carried out by someone with no experience in programming or large data structures whatsoever; the resulting analysis of even a few hundred images can easily create output files which contain dozens of

measurement columns and tens of thousands of rows, which often proves daunting or insurmountable to the large numbers of biologists without advanced computer training.

2 Program overview

CellProfilerStats provides a simple GUI-based program for users to begin exploring measurement data output by CellProfiler without requiring advanced computer skills. For users with a working knowledge of Python and who desire greater customization, the source code is provided along with a list of dependencies.

For analysis of still images:

CellProfilerStats (Fig 1A) asks users how they'd like to begin, asks them to select the folder that contains the several .csv files exported by CellProfiler, then asks if they want to add any additional analyses that can be derived from the CellProfiler output, such as measuring the distance to the closest neighbor object, normalizing fluorescence intensities of objects within a cell by the cellular expression level of the fluorescent protein, and calculating the fraction of the area or intensity of a larger parent object (ie a cell) occupied by all its children objects (ie the ribosomes). The selections made at this step can be saved as defaults for easy replicability and can be applied to many folders simultaneously using "batch mode". The original and newly generated measurements are output as a single .xlsx file per experiment, simplifying data management.

In a second step, users can select the .xlsx file or files they want to further analyze; if multiple files are selected, the user is prompted to select a "baseline" condition for further statistical analyses. Users are guided through a series of menus to select a list of measurements

and how they want their data graphed (linear vs log); these again may be saved as a default for easy reuse. Multiple graph types for each measurement are created using the seaborn graphing package (https://stanford.edu/~mwaskom/software/seaborn/) and saved as vector graphics in .pdf format (Fig 1B). If multiple data sets are being compared then a statistical heatmap is created to see whether any groups are different from the baseline on any selected measurement by either the Mann-Whitney or Kolmogorov-Smirnov 2 sample test; the entire resulting array is corrected for multiple comparisons and presented in Excel with color coding so the results can be quickly and easily interpreted (Fig 1C).

Most importantly, users can also choose "filters" to analyze specific subsets of their data, allowing easy testing of questions such as "is DNA damage staining particularly intense in cells above the 90th percentile for the number of mitochondria?" or "do cells with an area of <10000 pixels show unique patterns of actin localization?". The correlation of markers in different channels across particular subpopulations can answer some of the most interesting scientific questions but is often prohibitively tedious and error-prone without programming skills; CellProfilerStats allows users to accomplish it by following a few simple prompts (Fig 1D).

For analysis of time lapse movies:

CellProfilerStats can also be used downstream of CellProfiler's "TrackObjects" module for analysis of object motion in timelapse movies. Comparisons between multiple treatments can be made for measures of speed and displacement; if the user inputs the framerate and pixel size of their movies the mean squared displacement is calculated as well. Speed and displacement are correlated with the particle's size and integrated intensity, and per-track and per-movie measurements are graphed for easy detection of outliers and subpopulations (Fig 1E).

3 Conclusions

CellProfilerStats opens high-content image analysis to the large group of biologists who may want to ask detailed questions about subpopulations of their cells but who lack the training to begin. It generates robust statistical analyses with appropriate corrections for multiple comparison, automatedly generates vector graphics, and allows users to quickly and easily analyze subgroups within their data to test more complicated hypotheses. The ability to save analysis settings as defaults increases the ease of use while ensuring analysis-to-analysis comparability. Additionally it can be used in both the analysis of single images and timelapse movies, making it versatile and useful for a wide range of experiments.

Programming is valuable and necessary skill, as evidenced by the success of groups like Software Carpentry (http://software-carpentry.org/) that teach introductory programming workshops for researchers. Until these skills are more ubiquitous in the larger biology community, however, there is a need for programs that can create gateways into large data sets for researchers without computer science training. It is our hope that showing researchers the power of granular analysis of large image sets may even help demystify the process and encourage them to learn to manipulate these data sets on their own in MatLab, R, or Python, and to pass these skills on in their own circles.

Funding

This research was supported by NCI-CA96840 to EHB

Conflict of Interest: none declared.

References

Kamentsky,L. *et al.* (2011) Improved structure, function and compatibility for CellProfiler: modular high-throughput image analysis software. *Bioinformatics*, **27**, 1179–1180.
Loo,L.-H. *et al.* (2007) Image-based multivariate profiling of drug responses from single cells. *Nat. Methods*, **4**, 445–453.
Perkel,J. (2016) Making Sense of Big Data. *Biotechniques*, **60**, 08–112.
Singh,S. *et al.* (2014) Increasing the Content of High-Content Screening: An Overview. *J. Biomol. Screen.*, **19**, 640–650.
Smolinski,T.G. (2010) Computer literacy for life sciences: helping the digital-era biology undergraduates face today's research. *CBE Life Sci. Educ.*, **9**, 357–363.

Figure Legends

Figure 1: Interface and outputs of CellProfilerStats A) CellProfilerStats presents an interactive interface that asks users which kinds of analysis they'd like to do at each step of the process. B) Example of statistical heatmap produced by CellProfilerStats. Each row is a feature, each column is a treatment group, % change of the median is reported as well as color coding of which treatments were statistically significantly different after correction. C) Results are reported as cumulative frequencies, violin plots, and bar graphs to allow maximum exploration of the data. D) Analysis of subgroups is automated and shown as violin plots; two-sample T tests are presented to show where subgroups may be different from the total population. E) Output from the tracking analysis modules is broken down by tracked object and by cell to allow for identification of outliers and interesting subpopulations.

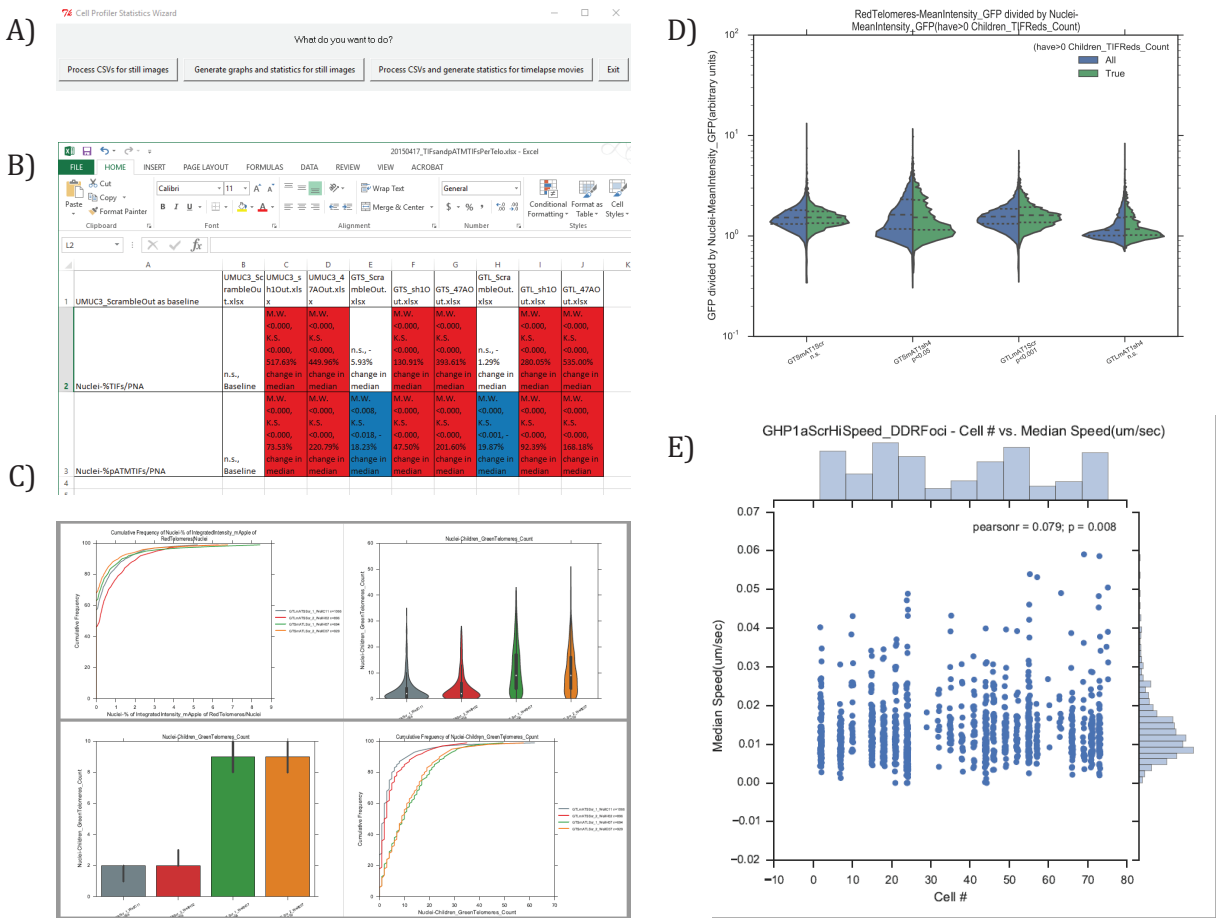


Figure 1: Interface and outputs of CellProfilerStats

Appendix A: Other useful modules

These modules and others can be found on the Blackburn Lab NAS in the "Public->Emeritus->Beth" directory or by contacting beth.cimini@gmail.com

Name/Category	Program	List of modules	Interactive interface?	Description
FRAP analysis	FIJI+Python	_FRAPAnalysisHighSpeedPlusROISav.py, FRAPAnalysis.py	FIJI yes, python no	Run the first in FIJI- pulls bleach coordinate data from microscope, user can adjust bleach ROI if necessary, then manually draw a nuclear ROI and an "outside the cell" ROI. It'll save a crop of just the cell and just the ROI if you need to go back to it later or want to analyze it in another program like CellProfiler. Feed the resulting data table for all images into the python module- calculates diffusion rates +/- SEM, calculates $t_{1/2}$.
RTPCR	Python, standalone .exe	RTPCR.py	Yes	For LightCycler analysis- If you've designated your replicates in the LC software and exported the CTs as text, you can describe your experiment (what primers, which one is the housekeeping gene, what your groups are, which group is your negative control, even has limited plate-map guessing if your setup is simple but if not you can manually assign each replicate to a primer set and a group); normalizes every primer set to its housekeeping gene, then normalizes every group to the baseline/negative control, all with appropriate error propagation- spits out an excel file and automated bar graph .pdfs (vector graphic)
FileBlinder	Python	FileBlinder.py	Yes	Point it at a folder of images, it'll create a copy of it called __Randomized and then randomize all your images with just a 4 digit number for blind scoring; creates an Excel file for decoding at the end that you can also use as your scoring worksheet by just making the "FileName" column only a pixel or two wide

Name/Category	Program	List of modules	Interactive interface?	Description
MetaSystemsTreatmentGrouper	Python	MetaSystemsTreamentGrouper.py	Yes	If you associate the wells in your multi-well chamber with a spot number in the Metafer software and export the cell positions at the end, you can feed it the folder of images and the position list, it'll ask you what to name the subfolder for each well, and then poof all your images are sorted
MetaSystemsRenameForHuygens	Python	MetaSystemsRenameForHuygens.py	Yes	Renames the dumb dumb file names from Metafer to something that CellProfiler and Huygens can work with. By default 568/"Y" is ch00, 488/"G" is ch01, 405/"B" is ch02, 647/"R" is ch03
AppendOrRemoveRand	Python	AppendOrRemoveRand.py	Yes	Lets you append and then remove random numbers (001-999) to the beginning of your images if you want to scramble them for manual work in CellProfiler or anything else
PlateRandomizer	Python	PlateRandomizer.py	No	Can give it a plate size in rows and columns, names of your cell lines and treatments, how many replicates of each, and whether or not it should avoid edge and corner wells, then creates an Excel with your plate map- handy to avoid always using the same plating order and potentially introducing bias
BatchMakeTrackMovies	FIJI, downstream of CellProfilerStats	_BatchMakeTrackMoviesNoBckSub.py	Yes	If you've run the "Analyze timelapse movies" module of CellProfilerStats and told it to draw the tracks, this is the last step- in FIJI run this then point it at your top experiment folder (which should have subfolders for each treatment in your experiment, each of which should have subfolders for each movie made by CPS); it'll make side-by-side movies with the cell on the left and the cell plus tracks on the right and then save them all in the top folder.

Name/Category	Program	List of modules	Interactive interface?	Description
MulticolorTimelapseRegistration	FIJI	,_2ColorMovieRegister.py,_3ColorMovieRegister.py	No	For cellular/nuclear registration of timelapse movies. Best/most up to date one is the "from Huygens" one- it shows you the movie, you crop a box around the cell you're going to be analyzing as close as you can while still having all of it, and it'll do the registration. You do have to tell it how many channels, z planes, time points etc in the source code though.
Flatfield correction of large image sets	FIJI	_3ColorTimelapseFlatfieldCorre.py	No	For flatfield correction using a correction image. Semi-automated in that it'll ask you what your folder is and tell you to get your correction images open, but you need to tell it what the names of your correction images are and which one goes to each channel.
IDT	Python	IDTFiles.py	Yes after initial setup	Nice little utility to keep your oligos organized- have to do initial setup to tell it the name of your folder where you keep your oligos and the name of the file to write them to but then you just click it every time you add new .pdf's of your oligo orders and it pulls the name, date ordered, ID# if you give it one, sequence into an excel file. Good for downstream integration with FileMaker, etc
File utilities	Python	Renamer.py	No	Small utilities for renaming, moving, making copies of, and deleting files.
Image merge/split/change type utilities	FIJI	_BatchDVtoTIF.py, _BatchTIFtoJPG.py, _BatchSplitChannels.py, etc etc etc	Yes	Will convert your file, split its channels, merge its channels, etc depending on the specific one. Anything with "Nested" in the name and you give it a folder with subfolders and it'll do _____ to all your subfolders, if not otherwise specified it's everything in the folder you point it at.
CellProfilerPipelines	CellProfiler	Too many to name (~300)	No	If you name it and it has telomeres I've pipelined it- pipelines for live/fixed/timelapse movies/metaphase spreads/yeast/DAB staining etc. Hopefully can be helpful to start your own pipelines in the future.

Appendix B: CellProfilerStats source code

CellProfilerStatsGUI.py

```
“””The main user interface to CPS"""

import easygui as eg
import sys

while 1:
    message='What do you want to do?'
    title='Cell Profiler Statistics Wizard'
    choices=('Process CSVs for still images',' Generate graphs and statistics for still images',
            'Process CSVs and generate statistics for timelapse movies','Exit')
    mainmenu=eg.indexbox(message,title,choices)

    if mainmenu==0:
        import CSVCombinerPandas as CSVP
        CSVP.runcsvcombiner()

    elif mainmenu==1:
        import OutputSigsXL as OutSXL
        OutSXL.dothestuff()

    elif mainmenu==2:
        import SimpleTrackingCleanup as STC
        STC.dothestuff()


    else:

        if eg.ccbox('Exit?','Exit?',choices=('No','Yes')):

            pass

        else:

            sys.exit(0)
```

HandyXLModulesPruned.py

```
“””A dump for userful shortcut functions and graphing defaults"""

import matplotlib.pyplot as plt
from scipy import stats
import textwrap
import numpy
import os
import seaborn as sns
import pandas as pd
import matplotlib.lines as mlines

colorbrewerlist=['#6f828a','#e31a1c','#33a02c','#ff7f00','#1f78b4','#6a3d9a','#b15928','#f4d054','
#ac1db8','#c5c9c7','#fb9a99','#b2df8a','#fdbf6f','#7bc8f6','#cab2d6','#cba560','#fffe71','#fb5ffc','#
010fcc','#9d0759']
def pythag(xb,xa,yb,ya): #the pythagorean theorem
    return (((xb-xa)**2)+((yb-ya)**2))**(0.5)

def unziprezip(inlist):
    "'turns [[1,2,3],[a,b,c]] into [[1,a],[2,b],[3,c]]"'
    outlist=[]
    for i in range(len(inlist[0])):
        temp=[]
        for j in range(len(inlist)):
            temp.append(inlist[j][i])
        outlist.append(temp)
    return outlist

def deepunziprezip(inlist):
    "'turns [[[1,2],[3,4]],[[5,6],[7,8]]] into [[[1,5],[2,6]],[[3,7],[4,8]]]"'
    outlist=[]
    for i in range(len(inlist[0])):
        outtemp=[]
        length=[]
        for k in inlist[0]:
            length.append(len(k))
        for j in range(max(length)):
            intemp=[]
            for m in range(len(inlist)):
                try:
                    intemp.append(inlist[m][i][j])
                except:
                    pass
            outtemp.append(intemp)
        outlist.append(outtemp)
    return outlist
```

```
def partialcount(list,query):
    """totals instances of a query in all members of a list (ie 4 a's in a list of ['apple','banana'])"""
    count=0
    for i in list:
        subcount=i.count(query)
        count+=subcount
    return count

def stripheaderonce(t,delimiter='_'):
    '''removes the header before and up to the first underscore (or whatever delimiter chosen) from a string'''
    if delimiter in t:
        t=t[t.index(delimiter)+1:]
    return t

def stripheaderagg(t,delimiter='_'):
    '''removes the header before and up to the last underscore (or whatever delimiter chosen) from a string'''
    while delimiter in t:
        t=t[t.index(delimiter)+1:]
    return t

def middleofdelim(t,delimiter='_'):
    '''pulls any text between (the first two) instances of a delimiter'''
    if delimiter in t:
        t=t[t.index(delimiter)+1:]
    if delimiter in t:
        t=t[:t.index(delimiter)]
    return t

def stripheaderlist(t,delimiter='_'):
    '''removes the header before and up to an underscore (or whatever delimiter chosen) from each string in a list'''
    s=[]
    for i in t:
        i=str(i)
        if delimiter in i:
            s.append(i[(i.index(delimiter)+1):])
        else:
            s.append(i)
    return s

def axisvaluefinder(columnheading):
    '''Function to try to clean up axes a bit for graphs'''
    if type(columnheading)!=str:
```

```
        columnheading=str(columnheading)
    if '_Area' in columnheading:
        value='Area (pixels^2)'
    elif 'Eccentricity' in columnheading:
        value='Eccentricity'
    elif 'Intensity' in columnheading:
        value=stripheaderlist([columnheading])[0]+'(arbitrary units)'
    elif 'Distance' in columnheading:
        value='Distance (pixels)'
    elif 'To' in columnheading:
        value='Distance (pixels)'
    elif 'Length' in columnheading:
        value='Length (pixels)'
    else:
        value=columnheading
    return value
```

def graphhistsaslines(listoflists, listoflabels,title,xlabel='',ylabel='% of values',normed=True, nbins=20,PDF=False,log=False):

```
    sns.set(style='ticks')
    plt.ioff() #turn off interactive mode to run more quickly
    plt.figure() #start a new figure
    listoflabelscopy=[]
    for j in range(len(listoflabels)):
        listoflabelscopy.append(listoflabels[j]+' n='+str(len(listoflists[j])))
    if normed==True:
        weightlist=[]
        for eachlist in listoflists:
            setlen=len(eachlist)
            weightlist.append(setlen*[100.0/float(setlen)])
    ax=plt.subplot(111)
    if len(listoflabelscopy)<=20:
        colorlist=colorbrewerlist[:len(listoflabelscopy)]
    else:

colorlist=(colorbrewerlist*(len(listoflabelscopy)/20))+colorbrewerlist[:len(listoflabelscopy)%20]
    for j in range(len(listoflists)):
        if normed==False:
            n,bins=numpy.histogram(listoflists[j],bins=nbins)
            #plt.hist(listoflists,label=listoflabelscopy,cumulative=False,
alpha=0.8,histtype=histtype,bins=nbins,log=log,color=colorlist) #create a 20-binned histogram
<---------More parameters can be changed here if desired
        else:
            n,bins=numpy.histogram(listoflists[j],bins=nbins,weights=weightlist[j])
```

```
            #plt.hist(listoflists,label=listoflabelscopy,cumulative=False,
weights=weightlist,alpha=0.8,histtype=histtype,bins=nbins,log=log,color=colorlist) #create a 20-
binned histogram <---------More parameters can be changed here if desired
        # Shink current axis by 20%
        bincenters=[]
        for i in range(len(bins)-1):
            bincenters.append((bins[i]+bins[i+1])/2)
        ax.plot(bincenters,n,label=listoflabelscopy[j],color=colorlist[j%20],zorder=0)
    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
    # Put a legend to the right of the current axis
    formtitle="\n".join(textwrap.wrap(title))
    plt.title(formtitle,stretch='ultra-condensed')#title the histogram with what it is
    plt.xlabel(axisvaluefinder(xlabel))
    if log==True:
        plt.xscale('log')
    plt.ylabel(axisvaluefinder(ylabel))
    leg=plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    for t in leg.get_texts():
        t.set_fontsize('xx-small')
    #plt.show()
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()
```

def graphscumhist(listofxs,listoflabels,xlabel,mantitle=None,markersize=6,colorlist=colorbrewerlist,PDF=False,log=False):

```
    '''function for graphing stuff in matplotlib'''
    sns.set(style='ticks')
    plt.ioff() #turn off interactive mode
    fig=plt.figure(1)
    listoflabelscopy=[]
    for j in range(len(listoflabels)):
        listoflabelscopy.append("\n".join(textwrap.wrap(listoflabels[j]+' n='+str(len(listofxs[j])))))
    ax=plt.subplot(111)
    for i in range(len(listofxs)):
        #print i%6,i%7
        try:
            percents=range(0,100)
            percentvals=[]
            for j in percents:
                percentvals.append(stats.scoreatpercentile(listofxs[i],j))
            ax.plot(percentvals,percents,label=listoflabelscopy[i],color=colorlist[i%20])
        except:
```

```
            pass
    if log==True:
        plt.xscale('log')
    if mantitle==None:
        mantitle='Cumulative Frequency of '+xlabel #add the title and axis labels
    formtitle="\n".join(textwrap.wrap(mantitle))
    plt.title(formtitle,stretch='ultra-condensed')
    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
    # Put a legend to the right of the current axis
    leg=ax.legend(numpoints=1,loc='center left', bbox_to_anchor=(1, 0.5))
    for t in leg.get_texts():
        t.set_fontsize('xx-small')
    plt.xlabel(xlabel)
    plt.ylabel('Cumulative Frequency')
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    fig.clear()
    plt.close(fig)
    plt.clf()
    #gc.collect()
```

def graphscatterpandas(listofxsprelim,listofysprelim,listoflabels,xlabelprelim,ylabelprelim,mantitle=None,markersize=6,markerlist=['o','^','s','p','H','D'],colorlist=['b','r','g','y','k','m','c'],PDF=False,log=False):

```
    '''function for graphing scatter plots in matplotlib'''
    if log==False:
        listofxs=listofxsprelim
        listofys=listofysprelim
        xlabel=xlabelprelim
        ylabel=ylabelprelim
    else:
        if 'x' in log:
            xlabel=xlabelprelim+' (log)'
            listofxs=[]
            for i in listofxsprelim:
                listofxs.append(list(numpy.log10(i)))
        else:
            listofxs=listofxsprelim
            xlabel=xlabelprelim
        if 'y' in log:
            ylabel=ylabelprelim+' (log)'
            listofys=[]
            for i in listofysprelim:
                listofys.append(list(numpy.log10(i)))
```

```
        else:
            listofys=listofysprelim
            ylabel=ylabelprelim
    plt.ioff() #turn off interactive mode
    plt.figure()
    ax=plt.subplot(111)
    if len(listofxs)==1:
        plt.close()
        plt.clf()
        sns.set(style='ticks')
        #print listofxs[0],listofys[0]
        if 'Cell #' not in xlabel:
            if len(listofxs[0])<4000:

sns.jointplot(x=pd.Series(listofxs[0],name=axisvaluefinder(xlabel)),y=pd.Series(listofys[0],name=
axisvaluefinder(ylabel)),kind='reg',marginal_kws=dict(kde=False))
            else:

sns.jointplot(x=pd.Series(listofxs[0],name=axisvaluefinder(xlabel)),y=pd.Series(listofys[0],name=
axisvaluefinder(ylabel)),kind='hex')
        else:

sns.jointplot(x=pd.Series(listofxs[0],name=axisvaluefinder(xlabel)),y=pd.Series(listofys[0],name=
axisvaluefinder(ylabel)))
    else:
        fulllabellist=[]
        for i in range(len(listofxs)):
            fulllabellist+=[listoflabels[i]]*len(listofxs[i])
        if len(fulllabellist)>8000:
            sns.set(style="darkgrid")
            forlegend=[]
            for i in range(len(listofxs)):
            #print i%6,i%7

sns.kdeplot(pd.Series(listofxs[i],name=axisvaluefinder(xlabel)),pd.Series(listofys[i],name=axisvalu
efinder(ylabel)),bw=0.05,cmap=sns.light_palette(colorbrewerlist[i%20],as_cmap=True,reverse=
True))

forlegend.append(mlines.Line2D([],[],color=colorbrewerlist[i%20],label=listoflabels[i]))
            ax.legend(handles=forlegend)
        else:
            pandaframe=pd.DataFrame()
            pandaframe=pandaframe.append(pd.Series(fulllabellist,name='Treatment'))
            masterxlist=[]
            masterylist=[]
            for i in listofxs:
```

```
            if type(i)!=list:
                i=list(i)
            masterxlist+=i
        for i in listofys:
            if type(i)!=list:
                i=list(i)
            masterylist+=i
        pandaframe=pandaframe.append(pd.Series(masterxlist,name=axisvaluefinder(xlabel)))
        pandaframe=pandaframe.append(pd.Series(masterylist,name=axisvaluefinder(ylabel)))
        pandaframe=pandaframe.transpose()

sns.lmplot(x=axisvaluefinder(xlabel),y=axisvaluefinder(ylabel),data=pandaframe,hue='Treatment',
palette=sns.color_palette(colorbrewerlist),truncate=True,robust=True)
    if mantitle!=False:
        plt.subplots_adjust(top=0.95)
        plt.suptitle(mantitle)
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()

def
graphtracksinacell(trackdict,mantitle,colorlist=['LightSkyBlue','DeepSkyBlue','Blue'
,'Black'],PDF=False):
    '''function for graphing cumulative track movements in matplotlib'''
    sns.set(style='ticks')
    plt.ioff() #turn off interactive mode
    plt.figure()
    ax=plt.subplot(111)
    ax.set_rasterization_zorder(1)
    for i in trackdict:
        ax.plot(i[1],i[2],color=colorlist[i[0]],zorder=0)
    if mantitle==None:
        mantitle='Integrated Distance of Tracks' #add the title and axis labels
    formtitle="\n".join(textwrap.wrap(mantitle))
    plt.title(formtitle,stretch='ultra-condensed')
    plt.xlabel('Frame #')
    plt.ylabel('Integrated distance traveled (pixels)')
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()
    #gc.collect()
```

```
def
graphmsds(listoftimepoints,listofmsds,labels=False,xunits='sec',yunits='(um)',manti
tle='Per-Track MSDs',PDF=False,each=False,havestdev=False):
    '''function for graphing MSD plots in matplotlib'''
    sns.set(style='ticks')
    plt.ioff() #turn off interactive mode
    plt.figure()
    ax=plt.subplot(111)
    ax.set_rasterization_zorder(1)
    if each==False:
        if havestdev==False:
            for i in range(len(listoftimepoints)):
                if labels==False:
                    ax.plot(listoftimepoints[i],listofmsds[i])
                else:
                    ax.plot(listoftimepoints[i],listofmsds[i],label=labels[i],color=colorbrewerlist[i%20])
        else:
            for i in range(len(listoftimepoints)):
                if labels==False:
                    ax.errorbar(listoftimepoints[i],listofmsds[i],yerr=havestdev[i])
                else:

ax.errorbar(listoftimepoints[i],listofmsds[i],label=labels[i],color=colorbrewerlist[i%20],yerr=hav
estdev[i])
    else:
        for i in range(len(listoftimepoints)):
            unzippeddata=unziprezip(listoftimepoints[i])
            ax.plot(unzippeddata[0],unzippeddata[1],zorder=0)
    if labels!=False:
        box = ax.get_position()
        ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
        # Put a legend to the right of the current axis
        leg=ax.legend(numpoints=1,loc='center left', bbox_to_anchor=(1, 0.5))
        for t in leg.get_texts():
            t.set_fontsize('xx-small')
    formtitle="\n".join(textwrap.wrap(mantitle))
    plt.title(formtitle,stretch='ultra-condensed')
    plt.xlabel('Time ('+xunits+')')
    plt.ylabel('MSD '+yunits[:-1]+'^2)')
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()
    #gc.collect()
```

```
def graphmanyhists(listoflists, listoflabels,title,xlabel='',ylabel='% of values',normed=True, nbins=20,histtype='bar',PDF=False,log=False,setbins=False):
    sns.set(style='ticks')
    plt.ioff() #turn off interactive mode to run more quickly
    plt.figure() #start a new figure
    listoflabelscopy=[]
    for j in range(len(listoflabels)):
        listoflabelscopy.append(listoflabels[j]+' \nn='+str(len(listoflists[j])))
    if normed==True:
        weightlist=[]
        for eachlist in listoflists:
            setlen=len(eachlist)
            weightlist.append(setlen*[100.0/float(setlen)])
    ax=plt.subplot(111)
    if len(listoflabelscopy)<=20:
        colorlist=colorbrewerlist[:len(listoflabelscopy)]
    else:

colorlist=(colorbrewerlist*(len(listoflabelscopy)/20))+colorbrewerlist[:len(listoflabelscopy)%20]
    if normed==False:
        plt.hist(listoflists,label=listoflabelscopy,cumulative=False,
histtype=histtype,bins=nbins,log=log,color=colorlist) #create a 20-binned histogram <---------
More parameters can be changed here if desired
    else:
        if setbins==False:
            plt.hist(listoflists,label=listoflabelscopy,cumulative=False,
weights=weightlist,histtype=histtype,bins=nbins,log=log,color=colorlist) #create a 20-binned
histogram <---------More parameters can be changed here if desired
        else:
            clippedlist=[]
            for eachlist in listoflists:
                clippedlist.append(list(numpy.clip(eachlist,setbins[0],setbins[-1])))
            plt.hist(clippedlist,label=listoflabelscopy,cumulative=False,
weights=weightlist,histtype=histtype,bins=setbins,log=log,color=colorlist)
            bincenters=[]
            xlabels=[]
            for i in range(len(setbins)-1):
                bincenters.append((setbins[i]+setbins[i+1])/2)
                xlabels.append(str(setbins[i])+'-'+str(setbins[i+1]))
            xlabels[-1] = str(setbins[-2])+'+'
            plt.xticks(bincenters)
            ax.set_xticklabels(xlabels)
            ymin, ymax = plt.ylim()
            if ymax>105:
                plt.ylim(0,105)
```

```
    # Shink current axis by 20%
    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
    # Put a legend to the right of the current axis
    formtitle="\n".join(textwrap.wrap(title))
    plt.title(formtitle,stretch='ultra-condensed')#title the histogram with what it is
    plt.xlabel(axisvaluefinder(xlabel))
    plt.ylabel(axisvaluefinder(ylabel))
    leg=plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    for t in leg.get_texts():
        t.set_fontsize('xx-small')
    #plt.show()
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()
```

def graphsubgrouppanda(pandadf,mantitle,parentstatname,howfilter,graphlog,readablelabellist,PDF=False):

```
    '''function for graphing subgroups as violinplots using seaborn'''
    sns.set(style='ticks')
    #[z,stattouse,findunfiltered,namefilter,readablelabellist,True]
    readablelabellist.sort()

#sns.violinplot(x='Treatment',y='Value',data=z,hue='Subgroup',split=True,scale='count',inner='quartile',order=readablelabellist.sort())
    labelcopy=[]
    #pandadf=pandadf.replace([numpy.inf, -numpy.inf], numpy.nan)
    #pandadf=pandadf.dropna()
    for i in range(len(readablelabellist)):

#stats.ttest_ind(z['Value'][z.isin(['sh1']).any(axis=1)*z.isin([6]).any(axis=1)],[z['Value'][z.isin(['Scr']).any(axis=1)*z.isin([5]).any(axis=1)]]*3)

pvalue=stats.ttest_ind(pandadf[parentstatname][pandadf.isin([readablelabellist[i]]).any(axis=1)*pandadf.isin(['All']).any(axis=1)],pandadf[parentstatname][pandadf.isin([readablelabellist[i]]).any(axis=1)*pandadf.isin(['True']).any(axis=1)],equal_var=False)[1]
        if pvalue<0.001:
            labelcopy.append(readablelabellist[i]+'\n p<0.001')
        elif pvalue<0.01:
            labelcopy.append(readablelabellist[i]+'\n p<0.01')
        elif pvalue<0.05:
            labelcopy.append(readablelabellist[i]+'\n p<0.05')
        else:
            labelcopy.append(readablelabellist[i]+'\n n.s.')
```

```
ax=sns.violinplot(x='Treatment',y=parentstatname,data=pandadf,hue=howfilter,split=True,inner
='quartile',order=readablelabellist,bw=0.05,cut=0)
    if graphlog==True:
        ax.set_yscale('log')
    else:
        ax.set_ylim(ymin=0)
    if mantitle==None:
        mantitle=parentstatname #add the title and axis labels
    formtitle="\n".join(textwrap.wrap(mantitle))
    ax.set_title(formtitle,stretch='ultra-condensed')
    ax.set_ylabel(axisvaluefinder(parentstatname))
    ax.set_xticklabels(labelcopy,fontsize='xx-small',rotation=30)
    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()
```

def graphswarmpandas(listofvalues,listoflabels,ylabel,bandwidth=0.05,mantitle=None,PDF=False,log=False,bargraph=False,clipoutliers=False,scale='area'):

```
    '''function for graphing violinplots (with overlaid scatters if there aren't too many points) in
seaborn'''

#newpal=['#1f78b4','#33a02c','#e31a1c','#ff7f00','#6a3d9a','#b15928','#a6cee3','#b2df8a','#fb9a9
9','#fdbf6f','#cab2d6','#ffff99']
    sns.set(style='ticks')
    sns.set_palette(sns.color_palette(colorbrewerlist))
    pandaframe=pd.DataFrame()
    labelsplusn=[]
    for i in range(len(listofvalues)):
        labelsplusn.append(listoflabels[i]+'\nn='+str(len(listofvalues[i])))
        pandaseries=pd.Series(listofvalues[i])
        pandaseries=pandaseries.replace([numpy.inf, -numpy.inf], numpy.nan)
        pandaseries=pandaseries.dropna()
        if log==False:
            if clipoutliers==True:
                pandaseries=pandaseries[~((pandaseries-
pandaseries.mean()).abs()>3*pandaseries.std())]
        pandaframe=pandaframe.append(pandaseries,ignore_index=True)
    pandaframe=pandaframe.transpose()
    if bargraph==False:
        ax=sns.violinplot(data=pandaframe,bw=bandwidth,scale=scale,cut=0)
        if pandaframe.shape[0]<100:
```

```
            sns.stripplot(data=pandaframe,jitter=True,size=2,zorder=0)
    else:
        ax=sns.barplot(data=pandaframe,estimator=numpy.median,ci=95)

    if log==True:
        ax.set_yscale('log')
    else:
        ax.set_ylim(ymin=0)
    if mantitle==None:
        mantitle=ylabel #add the title and axis labels
    formtitle="\n".join(textwrap.wrap(mantitle))
    ax.set_title(formtitle,stretch='ultra-condensed')
    ax.set_ylabel(axisvaluefinder(ylabel))
    ax.set_xticklabels(labelsplusn,fontsize='xx-small',rotation=30)

    if PDF!=False:
        plt.savefig(PDF, format='pdf')
    plt.close()
    plt.clf()

def findexcel(directory):
    '''Finds all files with a .xlsx extension in a given directory'''
    if not os.path.exists(directory): #Warn if directory doesn't exist
        print 'error: no such directory'
    else: #Create a list of the files
        t=[]
        for i in os.listdir(directory): #for all the files in the directory...
            if '.xls' in i: #if they have a .xls extension...
                j=os.path.join(directory,i) #create a full directory+filename string
                t.append([i,j]) #move to master list
    #print t
    return t
```

Spots.py

```
"""Creates class of tracked spot objects and the utilities to determine things like MSD"""

import numpy
import random
from scipy import stats
from HandyXLModulesPruned import pythag

class spots(list):
    def __init__(self,filename,movielen,minlength=0,maxlen=100000):
        """Read an Excel file, pick out all the
        instances of a given spot (as identified by Label)
        and give them a list of all the attributes measured
        The overall container is a list, with the headings as
        index 0 and the rest of the spots in order as lists of lists
        (one list for each timepoint, containing all the data
        from that timepoint)
        minlength allows the user to gate out instances that
        don't persist for a certain length of time"""

        array=numpy.genfromtxt(filename,delimiter=',',dtype=None)
        array2=numpy.genfromtxt(filename,delimiter=',',dtype=numpy.float64)
        headings=list(array[0,:]) #read headings

        self.append(headings) #save headings to self
        for i in range(len(headings)): #identify the label column
            if 'TrackObjects_Label' in headings[i]:
                labelcolumn=i
            if 'ImageNumber' == headings[i]:
                imnumcolumn=i
            if 'TrackObjects_Lifetime' in headings[i]:
                lifecolumn=i

        alllabels=list(array2[:,labelcolumn]) #a list of all the spot instances at all timepoints
        imnumlist=list(array2[:,imnumcolumn])
        #print imnumlist, list(array[:,imnumcolumn])
        lifelist=list(array2[:,lifecolumn])
        for w in range(1,len(alllabels)):
            alllabels[w]=int(alllabels[w])
            imnumlist[w]=int(imnumlist[w])
            lifelist[w]=int(lifelist[w])
        #print imnumlist
        starts=[]
        stops=[]
        for z in range(imnumlist[1],max(imnumlist[1:])):
```

```
    c=[]
    for i in range(1,len(imnumlist)):
        if imnumlist[i]==z:
            c.append([i,alllabels[i],lifelist[i]])
    #print c
    if len(c)>0:
        first=True
        while first==True:
            if c[0][1]!=1:
                first=False
            for j in range(len(c)):
                if c[j][2]!=1:
                    first=False
                """if c[j][0]-c[j-1][0]!=1:
                    first=False"""
            if first==True:
                starts.append(c[0][0])
                if c[0][0]!=1:
                    stops.append(c[0][0])
                first =False
stops.append(len(imnumlist))
#print starts, stops

for m in range(len(starts)):
    sublabels=alllabels[starts[m]:stops[m]]
    #print sublabels, "sublabels"
    sublife=lifelist[starts[m]:stops[m]]
    #print sublife, "sublife"
    """Add screening for only one telomere per identifier"""
    subarray=array2[starts[m]:stops[m],:]
    #print subarray
    maxlabel=max(sublabels) #find the number of spot instances
    for r in range(1,(maxlabel+1)): #for each spot identifier
        if sublabels.count(r)>=minlength: #checks if the spot lasts long enough
            #print "ident#", r
            a=[]  #the list of all information for an individual spot
            setcount=1
            for k in range(len(sublabels)): #if so, add all the information about that spot
                                        #at each timepoint
                if sublabels[k]==r:
                    if sublife[k]==setcount:
                        #while setcount<=maxlen:
                        fromstart=(int(subarray[k][0])-1)%movielen
                        if fromstart<maxlen:
                            a.append([fromstart]+list(subarray[k])[1:]+[int(subarray[k][0])])
                        setcount+=1
```

```
                #print "a",a
                if len(a)>=minlength: #see if spot is still long enough (if only considering a subset
of the movie)
                    self.append(a) #add the compiled spot information to the master list
                #print starts[m],r,'added'


    def realmsd(self,pixelsize,framerate,identifier,maxlentrack):
        """because apparently I am dumb"""
        x=self[0].index('Location_Center_X')
        y=self[0].index('Location_Center_Y')

        tracklist=[]
        alltracks={}
        count=0
        trackcount=0
        maxlencount=[]
        for i  in self[1:]:
            trackcount+=1
            maxlencount.append(len(i))
        for i  in self[1:]:
            trackframepos={}
            startframe=i[0][-1]
            for frame in range(len(i)):
                trackframepos[i[frame][-1]-startframe]=(i[frame][x],i[frame][y])
            allframes=trackframepos.keys()
            #print 'allframes',allframes
            allframes.sort()
            squareddisp={}
            for sep in range(1,allframes[-1]):
                if sep<maxlentrack:
                    squareddisp[sep]=[]
                    for j in allframes:
                        if j+sep in allframes:

squareddisp[sep].append((pixelsize*pythag(trackframepos[j+sep][0],trackframepos[j][0],trackfra
mepos[j+sep][1],trackframepos[j][1]))**2)
            #print squareddisp
            allsep=squareddisp.keys()
            allsep.sort()
            for i in allsep:
                if len(squareddisp[i])==0:
                    allsep=allsep[:allsep.index(i)]
                    break
            trackfinal=[]
            for eachsep in allsep:
```

```
trackfinal.append((eachsep*framerate,numpy.mean(squareddisp[eachsep]),len(squareddisp[eachsep])))
                if eachsep not in alltracks.keys():
                    alltracks[eachsep]=squareddisp[eachsep]
                else:
                    alltracks[eachsep]+=squareddisp[eachsep]
                #print trackfinal
            if len(trackfinal)>0:
                tracklist.append(trackfinal)
            count+=1
        finalsep=alltracks.keys()
        finalsep.sort()
        alltracklist=[]
        for eachframe in finalsep:

alltracklist.append((eachframe*framerate,numpy.mean(alltracks[eachframe]),numpy.std(alltracks[eachframe])/((len(alltracks[eachframe]))**0.5),len(alltracks[eachframe])))
        #print tracklist
        return tracklist, alltracklist

    def xyandrealt(self):
        """Pull the x and y coordinates for each spot at each
        timepoint- index 0 gives the user the frame at which
        the spot first appeared, and the rest are tuples
        containing the coordinate information for each
        timepoint"""
        x=self[0].index('Location_Center_X')
        y=self[0].index('Location_Center_Y')

        a=[]
        for i in self[1:]:
            #print i[0]
            b=[i[0][-1]]
            for j in i:
                b.append((j[-1]-b[0],j[x],j[y]))
            a.append(b)
        #print a
        return a

    def trackstonow(self,movielen):
        """Description"""
        x=self[0].index('Location_Center_X')
        y=self[0].index('Location_Center_Y')

        count=0
```

```
        tracksdict={}
        tracksonlyfinaldict={}
        for i in self[1:]:
            trackinprog=[[],[],count%13]
            cell=((i[0][-1]-1)/movielen)+1
            zlist=[]
            relzlist=[]
            lastframe=i[-1][-1]
            for j in i:
                zlist.append(j[-1])
                relzlist.append(j[-1]%200)
                trackinprog[0]=list(trackinprog[0])+[j[x]]
                trackinprog[1]=list(trackinprog[1])+[j[y]]
                if j[-1] not in tracksdict.keys():
                    tracksdict[j[-1]]=[list(trackinprog)]
                else:
                    tracksdict[j[-1]].append(list(trackinprog))
                if j[-1]==lastframe:
                    zlist=list(relzlist)
                    if cell not in tracksonlyfinaldict.keys():
                        tracksonlyfinaldict[cell]=[list(trackinprog)[:2]+[relzlist]]
                        #tracksonlyfinaldict[cell]=[[map(lambda x:x+1,trackinprog[0]),map(lambda
x:x+1,trackinprog[1]),zlist]]
                    else:
                        tracksonlyfinaldict[cell].append(list(trackinprog)[:2]+[relzlist])
                        #tracksonlyfinaldict[cell].append([map(lambda
x:x+1,trackinprog[0]),map(lambda x:x+1,trackinprog[1]),zlist])
                #print j[-1], trackinprog,tracksdict
            count+=1
        #print tracksdict[1],tracksdict[2],tracksdict[3]
        return tracksdict,tracksonlyfinaldict

    def intint(self,channelname):
        for i in self[0]:
            if 'Intensity_IntegratedIntensity' in i:
                if channelname in i:
                  k=self[0].index(i)
        a=[]
        medvals=[]
        for i in self[1:]:
            b=[i[0][0]]
            for j in i:
                b.append(j[k])
            a.append(b)
            medvals.append(numpy.median(b))
        return a, medvals
```

```
def intmean(self,channelname):
    for i in self[0]:
        if 'Intensity_MeanIntensity' in i:
            if channelname in i:
                k=self[0].index(i)
    a=[]
    medvals=[]
    for i in self[1:]:
        b=[i[0][0]]
        for j in i:
            b.append(j[k])
        a.append(b)
        medvals.append(numpy.median(b))
    return a, medvals

def intmeanframe(self,channelname):
    for i in self[0]:
        if 'Intensity_MeanIntensity' in i:
            if channelname in i:
                k=self[0].index(i)
    frames={}
    for i in self[1:]:
        for j in i:
            if j[0] not in frames.keys():
                frames[j[0]]=[j[k]]
            else:
                frames[j[0]]+=[j[k]]
    b=frames.keys()
    b.sort()
    meanframes=[]
    stdevframes=[]
    n=[]
    for i in b:
        n.append(len(frames[i]))
        meanframes.append(numpy.mean(frames[i]))
        stdevframes.append(numpy.std(frames[i]))
    return b,n,meanframes,stdevframes

def intnormframe(self,channel1name,channel2name):
    for i in self[0]:
        if 'Intensity_MeanIntensity' in i:
            if channel1name in i:
                k=self[0].index(i)
            if channel2name in i:
                m=self[0].index(i)
```

```
    frames={}
    for i in self[1:]:
        for j in i:
            if j[0] not in frames.keys():
                frames[j[0]]=[j[k]/j[m]]
            else:
                frames[j[0]]+=[j[k]/j[m]]
    b=frames.keys()
    b.sort()
    meanframes=[]
    stdevframes=[]
    for i in b:
        meanframes.append(numpy.mean(frames[i]))
        stdevframes.append(numpy.std(frames[i]))
    return b,meanframes,stdevframes

def size(self,pixelsize):
    for heading in self[0]:
        if 'AreaShape_Area' in heading:
            area=self[0].index(heading)

    if pixelsize==False:
        pixelsize=1
    a=[]
    medvals=[]
    for i in self[1:]:
        b=[i[0][0]]
        for j in i:
            b.append(j[area]*(pixelsize**2))
        a.append(b)
        medvals.append(numpy.median(b))
    return a,medvals

def diffco(self,pixelsize,framerate):
    for heading in self[0]:
        if 'DistanceTraveled' in heading:
            dist=self[0].index(heading)
    spotnum=[]
    calcdiffco=[]
    count=1
    for i in self[1:]:
        startframe=i[0][-1]
        lastframe=i[-1][-1]
        distinpix=i[-1][dist]*pixelsize
        diff=(distinpix**2)/(4*framerate*(lastframe+1-startframe))
        spotnum.append(count)
```

```
        calcdiffco.append(diff)
        count+=1
    return spotnum,calcdiffco


def speedanddisp(self,pixelsize,framerate):
    for heading in self[0]:
        if 'DistanceTraveled' in heading:
            dist=self[0].index(heading)
        if 'TrajectoryX' in heading:
            trajx=self[0].index(heading)
        if 'TrajectoryY' in heading:
            trajy=self[0].index(heading)
    if pixelsize==False:
        pixelsize=1
    if framerate==False:
        framerate=1
    speed=[]
    disp=[]
    integ=[]
    medspeed=[]
    maxdisp=[]
    for i in self[1:]:
        currframe=i[0][-1]
        instdisp=[0]
        instspeed=[0]
        instintegdist=[0]
        instspeednumonly=[0]
        for j in range(1,len(i)):
            pythag=((i[j][trajx]**2)+(i[j][trajy]**2))**(0.5)
            instdisp.append(i[j][dist]*pixelsize)
            instintegdist.append(instintegdist[-1]+pythag*pixelsize)
            if i[j][-1]-currframe==1:
                instspeed.append(pythag*pixelsize/framerate)
                instspeednumonly.append(pythag*pixelsize/framerate)
            else:
                instspeed.append('')
            currframe=i[j][-1]
        disp.append(instdisp)
        speed.append(instspeed)
        integ.append(instintegdist)
        maxdisp.append(numpy.max(instdisp))
        medspeed.append(numpy.median(instspeednumonly))

    return speed,disp,integ,medspeed,maxdisp
```

```
    def speedanddisppercell(self,movielen,pixelsize,framerate):
        if pixelsize==False:
            pixelsize=1
        if framerate==False:
            framerate=1
        for heading in self[0]:
            if 'DistanceTraveled' in heading:
                dist=self[0].index(heading)
            if 'TrajectoryX' in heading:
                trajx=self[0].index(heading)
            if 'TrajectoryY' in heading:
                trajy=self[0].index(heading)

        medspeed=[]
        meddisp=[]
        maxspeed=[]
        maxdisp=[]
        allsumspeedx=[]
        allsumspeedy=[]
        sumspeed={}
        allpercentsums=[]
        for i in self[1:]:
            cell=((i[0][-1]-1)/movielen)+1
            currframe=i[0][-1]
            disp=[0]
            speed=[0]
            scat=cell+random.uniform(-.25,.25)
            for j in range(1,len(i)):
                disp.append(i[j][dist]*pixelsize)
                if i[j][-1]-currframe==1:
                    speed.append((((i[j][trajx]**2)+(i[j][trajy]**2))**(0.5))*pixelsize/framerate)
                if (cell,currframe) not in sumspeed.keys():

sumspeed[(cell,currframe)]=[i[j][trajx]*pixelsize/framerate,abs(i[j][trajx])*pixelsize/framerate,i[j]
[trajy]*pixelsize/framerate,abs(i[j][trajy])*pixelsize/framerate,1]
                else:
                    #print currframe, sumspeed[(cell,currframe)], i[j][trajx], i[j][trajy]
                    sumspeed[(cell,currframe)][0]+=i[j][trajx]*pixelsize/framerate
                    sumspeed[(cell,currframe)][1]+=abs(i[j][trajx])*pixelsize/framerate
                    sumspeed[(cell,currframe)][2]+=i[j][trajy]*pixelsize/framerate
                    sumspeed[(cell,currframe)][3]+=abs(i[j][trajy])*pixelsize/framerate
                    sumspeed[(cell,currframe)][4]+=1
                    #print sumspeed[(cell,currframe)]
                currframe=i[j][-1]
            medspeed.append((scat,numpy.median(speed)))
            meddisp.append((scat,numpy.median(disp)))
```

```
            maxspeed.append((scat,numpy.max(speed)))
            maxdisp.append((scat,numpy.max(disp)))
        #print sumspeed
        for i in sumspeed.keys():
            scat=i[0]+random.uniform(-.25,.25)
            allsumspeedx.append((scat,sumspeed[i][0]*100/sumspeed[i][1]))
            allsumspeedy.append((scat,sumspeed[i][2]*100/sumspeed[i][3]))

allpercentsums.append((i[1],sumspeed[i][4],sumspeed[i][0]*100/sumspeed[i][1],sumspeed[i][2]*
100/sumspeed[i][3]))
        allpercentsums.sort()
        return medspeed,meddisp,maxspeed,maxdisp,allsumspeedx,allsumspeedy,allpercentsums

    def intintandintdistpercell(self,movielen,channelname,pixelsize):
        for heading in self[0]:
            if 'Intensity_IntegratedIntensity' in heading:
                if channelname in heading:
                    intint=self[0].index(heading)
            if 'TrajectoryX' in heading:
                trajx=self[0].index(heading)
            if 'TrajectoryY' in heading:
                trajy=self[0].index(heading)
        if pixelsize==False:
            pixelsize=1
        tracksbycell={}
        intensitybycell={}
        classifiedbycell={}

        for i in self[1:]:
            cell=((i[0][-1]-1)/movielen)+1
            if cell not in tracksbycell.keys():
                tracksbycell[cell]=[]
                intensitybycell[cell]=[]
            if i[0][-1]%movielen==1:
                disp=[0]
                intensity=[]
                frame=[0]
                for j in range(1,len(i)):
                    frame.append(i[j][-1]-i[0][-1])
                    intensity.append(i[j][intint])
                    pythag=((i[j][trajx]**2)+(i[j][trajy]**2))**(0.5)
                    disp.append(disp[-1]+pythag*pixelsize)
                intensitybycell[cell].append(numpy.median(intensity))
                tracksbycell[cell].append([numpy.median(intensity),frame,disp])

        for eachcell in intensitybycell.keys():
```

```
        if len(intensitybycell[eachcell])>3:
            classifiedbycell[eachcell]=[]
            per25=stats.scoreatpercentile(intensitybycell[eachcell],25)
            per50=stats.scoreatpercentile(intensitybycell[eachcell],50)
            per75=stats.scoreatpercentile(intensitybycell[eachcell],75)
            for eachtrack in tracksbycell[eachcell]:
                if eachtrack[0]<per25:
                    classifiedbycell[eachcell].append([0]+eachtrack[1:])
                elif eachtrack[0]<per50:
                    classifiedbycell[eachcell].append([1]+eachtrack[1:])
                elif eachtrack[0]<per75:
                    classifiedbycell[eachcell].append([2]+eachtrack[1:])
                else:
                    classifiedbycell[eachcell].append([3]+eachtrack[1:])
        else:
            classifiedbycell[eachcell]=[]
            for eachtrack in tracksbycell[eachcell]:
                classifiedbycell[eachcell].append([3]+eachtrack[1:])

    return classifiedbycell
```

def xyandtindiv(self,indiv):

```
    """Pull the x and y coordinates of a single spot.
    Index 0 is the frame number of the first appearance"""
    x=self[0].index('Location_Center_X')
    y=self[0].index('Location_Center_Y')
    a=[self[indiv][0][0]]
    for j in self[indiv]:
        a.append((j[x],j[y]))
    return a
```

def lengthhist(self):

```
    """Provide a rough length histogram for the spots"""
    pulllengths=[]
    for i in self[1:]:
        pulllengths.append(len(i)-1)
    lengthhist,bins=numpy.histogram(pulllengths,bins=20)
    lengthout=[]
    for j in range(len(lengthhist)):
        binstr=str(bins[j])+'-'+str(bins[j+1])
        lengthout.append((lengthhist[j],binstr))
    return lengthout
```

CSVCombinerPandas.py

‘’’’’Pulls CSVs and combines them into a single excel file per folder, along with allowing the user to add certain normalizations or extra analyses."""

```
import os
import csv
import math
import easygui as eg
import shelve
import numpy
import HandyXLModulesPruned as HXM
from openpyxl import Workbook
import pandas as pd
from openpyxl.cell import get_column_letter

def findcsv(directory): #Finds all files with a .csv extension in a given directory
    if not os.path.exists(directory): #Warn if directory doesn't exist
        print 'error: no such directory'
    else: #Create a list of the files
        t=[]
        for i in os.listdir(directory): #for all the files in the directory...
            if '.csv' in i: #if they have a .csv extension...
                j=os.path.join(directory,i) #create a full directory+filename string
                t.append([i,j]) #move to master list
            else: pass
    return t

def compare2dicts(dict1,dict2,dist=500):
    outdict={}
    for i in dict1.keys():
        if i in dict2.keys():
            dict2list=[]
            for j in dict2[i]:
                dict2list.append((j[1],j[2]))
            for k in dict1[i]:
                minval=dist*math.sqrt(2)
                for m in dict2list:
                    if abs(k[1]-m[0])<dist:
                        if abs(k[2]-m[1])<dist:
                            pythagval=HXM.pythag(k[1],m[0],k[2],m[1])
                            if pythagval<minval:
                                if pythagval!=0:
                                    minval=pythagval
```

```
            outdict[k[0]]=minval
        else:
            for j in dict1[i]:
                outdict[j[0]]=dist*math.sqrt(2)
    return outdict

def pullallvals(pandadf,values,parentvalue):
    #pulls a value or values, returns a dictionary where the keys are a tuple of image and nucleus
numbers and
    #the value is a list of lists of the row and the value(s) for each object
    nucdic={}
    for i in range(pandadf.shape[0]):
        b=[i]
        for j in values:
            b.append(pandadf.get_value(i,j))
        if (pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,parentvalue)) not in
nucdic.keys():
            nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,parentvalue))]=[b]
        else:
            nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,parentvalue))]+=[b]
    return nucdic

def makenormfactorsub(pandadf,value):
    nucdic={}
    for i in range(pandadf.shape[0]):
        b=pandadf.get_value(i,value)
        if (pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber')) not in
nucdic.keys():
            nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber'))]=[b]
        else:

nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber'))]+=[b]
    normdic={}
    for i in nucdic:
        normdic[i]=numpy.average(nucdic[i])
    return normdic

def makenormfactordiv(pandadf,value,relmean=False):
    nucdic={}
    normdic={}
    if relmean==False:
        #print 'Using already calculated mean'
        for i in range(pandadf.shape[0]):

normdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber'))]=pandadf.g
et_value(i,value)
```

```
    else:
        masterlist=[]
        for i in range(pandadf.shape[0]):
            b=pandadf.get_value(i,value)
            masterlist.append(b)
            if (pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber')) not in
nucdic.keys():

nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber'))]=[b]
            else:

nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'ObjectNumber'))]+=[b]

        mastermean=numpy.average(masterlist)
        for i in nucdic:
            normdic[i]=numpy.average(nucdic[i])/mastermean
    #print normdic
    return normdic

def maketotal(pandadf,value,parentvalue):
    nucdic={}
    for i in range(pandadf.shape[0]):
        b=pandadf.get_value(i,value)
        try:
            b=float(b)
            if (pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'Parent_'+parentvalue)) not
in nucdic.keys():

nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'Parent_'+parentvalue))]=b
            else:

nucdic[(pandadf.get_value(i,'ImageNumber'),pandadf.get_value(i,'Parent_'+parentvalue))]+=b
        except:
            pass
    return nucdic

def csvcombinermain(directory,csvoutname,default=None):
    pandadict={}
    w=Workbook() #Create new excel file
    filelist=findcsv(directory)
    for y in filelist: #for each csv file...
        c=len(y[0]) #see how long the filename is- for removing the .csv extension below
        while c>=35: #Sheet names can only be 31 characters long- prompt user if shorter sheet
name required
            y[0]=(eg.enterbox('Filename '+y[0]+' is too long- enter a shorter one')+'.csv')
            c=len(y[0])
```

```
        b=w.create_sheet(title=y[0][:-4]) #name each sheet according to the name of it's .csv file
        a=csv.reader(open(y[1])) #Read the file
        for rows in a:
            rowcopy=[]
            for eachcell in rows:
                try:
                    eachcell=float(eachcell)
                except:
                    pass
                rowcopy.append(eachcell)
            b.append(rowcopy) #Transfer CSV's into python lists for use below
        pandadict[y[0][:-4]]=pd.read_csv(y[1],header=0)
    #print pandadict.keys(),pandadict.values()

    sheetlist=pandadict.keys()
    headingdict={}
    numcoldict={}
    for i in sheetlist:
        headingdict[i]=pandadict[i].columns
        numcoldict[i]=len(pandadict[i].columns)+1


    if default==None:
        prevdef=shelve.open(os.path.join(os.curdir,'csvshelf'),writeback=True)
        usedef=eg.ynbox(msg='Do you want to use a default?')
        if usedef:
            default=eg.choicebox(msg='Pick the default you wish to use', choices=prevdef.keys())
            tocompinit,dist,kidanalysis,normanalysis,fractanalysis,waveratio=prevdef[default]
        else:
            if eg.ynbox(msg='Do you want to compare the locations of different objects?'):
                tocompinit=[]
                tocompinitchoices=eg.multchoicebox(msg='Pick the items whose locations you want
to compare',choices=w.get_sheet_names())
                allparentdic={}
                for eachchoice in tocompinitchoices:
                    for eachheading in headingdict[eachchoice]:
                        if 'Parent' in eachheading:
                            if eachheading not in allparentdic.keys():
                                allparentdic[eachheading]=[eachchoice]
                            else:
                                allparentdic[eachheading].append(eachchoice)
                for eachkey in allparentdic.keys():
                    if len(allparentdic[eachkey])==len(tocompinitchoices):
                        tocompinit=(tocompinitchoices,eachkey)
                if len(tocompinit)<=1:
                    tocompinit=False
```

```
            else:
                dist=float(eg.enterbox(msg='Enter the maximum distance (in pixels) to look for
nearby objects'))
            else:
                tocompinit=False
                dist=False

            if eg.ynbox(msg='Do you want to compare the number of 2 different children objects in
the same parent (ie in a given nucleus, what is the ratio of telomeres to centromeres)?'):
                have2kids=[]
                allkids=[]
                readablekids=[]
                for k in headingdict.keys():
                    numkids=HXM.partialcount(headingdict[k],'Children')
                    if numkids>=2:
                        have2kids.append(k)
                        for j in headingdict[k]:
                            if 'Children' in j:
                                allkids.append((k,j))
                                readablekids.append(HXM.stripheaderonce(k)+' -
'+HXM.middleofdelim(j))
                if len(have2kids)==0:
                    kidanalysis=False
                else:
                    numeratorkids=eg.multchoicebox(msg='Pick all the NUMERATOR children
objects', choices=readablekids)
                    if len(numeratorkids)==0:
                        kidanalysis=False
                    else:
                        kidanalysis=[]
                        for num in numeratorkids:
                            subkidlist=[]
                            for subnum in readablekids:
                                if num[:num.index('-')]==subnum[:num.index('-')]:
                                    subkidlist.append(subnum)
                            denomkids=eg.multchoicebox(msg='Pick all the DENOMINATOR children
objects for '+num,choices=subkidlist)
                            if len(denomkids)==0:
                                pass
                            else:
                                for subdenom in denomkids:

kidanalysis.append((allkids[readablekids.index(num)],allkids[readablekids.index(subdenom)]))

                    #print kidanalysis
                    if len(kidanalysis)==0:
```

```
                kidanalysis=False
        else:
            kidanalysis=False

        if eg.ynbox(msg='Do you want to normalize any area or intensity measurements to the
parental value?'):
            normanalysis=[]
            allaandi=[]
            readableaandi=[]
            for k in headingdict.keys():
                for heads in headingdict[k]:
                    if '_Area' in heads:
                        allaandi.append((k,heads))
                        readableaandi.append(HXM.stripheaderonce(k)+' -
'+HXM.stripheaderonce(heads))
                    elif 'Intensity' in heads:
                        allaandi.append((k,heads))
                        readableaandi.append(HXM.stripheaderonce(k)+' -
'+HXM.stripheaderonce(heads))
            tobenormed=eg.multchoicebox(msg='Pick all the objects TO BE normalized',
choices=readableaandi)
            if len(tobenormed)==0:
                normanalysis=False
            else:
                for sinnorm in tobenormed:
                    normers=eg.multchoicebox(msg='Pick all the objects to normalize '+sinnorm+'
BY',choices=readableaandi)
                    if len(normers)==0:
                        pass
                    else:
                        for eachnorm in normers:
                            subordiv=eg.buttonbox(msg='Should division or subtraction be used to
normalize '+sinnorm+' by '+eachnorm+'?',choices=('Division','Subtraction'))

normanalysis.append((allaandi[readableaandi.index(sinnorm)],allaandi[readableaandi.index(eachn
orm)],subordiv))
            #print normanalysis
            #print (type(normanalysis))
            if len(normanalysis)==0:
                normanalysis=False

        else:
            normanalysis=False

        if eg.ynbox(msg='Do you want to compare the fraction of any TOTAL child
measurements to their parent measurements (ie, % of a nucleus covered by telomeres)?'):
```

```
            fractanalysis=[]
            have2kids=[]
            allkids=[]
            readablekids=[]
            for k in headingdict.keys():
                for j in headingdict[k]:
                    if 'Children' in j:
                        allkids.append((k,j))
                        readablekids.append(HXM.stripheaderonce(k)+' - '+HXM.middleofdelim(j))
            numeratorkids=eg.multchoicebox(msg='Pick all the children objects',
choices=readablekids)
            if len(numeratorkids)==0:
                fractanalysis=False
            else:
                for m in numeratorkids:
                    parentaai=[]
                    parentsheet=allkids[readablekids.index(m)][0]
                    for n in headingdict[parentsheet]:
                        if '_Area' in n:
                            parentaai.append(n)
                        elif 'Intensity' in n:
                            parentaai.append(n)
                    for n in headingdict.keys():
                        if HXM.middleofdelim(allkids[readablekids.index(m)][1]) == n:
                            childsheet=n
                    sharedaai=[]
                    for n in headingdict[childsheet]:
                        if n in parentaai:
                            sharedaai.append(n)
                    else:
                        valtofract=eg.multchoicebox(msg='Pick all the values you wish to do fraction
analysis on for '+m,choices=sharedaai)
                        for r in valtofract:
                            fractanalysis.append(((parentsheet,r),(childsheet,r)))
            if len(fractanalysis)==0:
                fractanalysis=False

        else:
            fractanalysis=False

        if eg.ynbox(msg='Do you want to compare the ratio of two wavelengths in any
objects?'):
            waveratio=[]
            allwaves=[]
            readablewaves=[]
            for k in headingdict.keys():
```

```
                for heads in headingdict[k]:
                    if 'Intensity' in heads:
                        allwaves.append((k,heads))
                        readablewaves.append(HXM.stripheaderonce(k)+' -
'+HXM.stripheaderonce(heads))
            toberated=eg.multchoicebox(msg='Pick all the objects TO BE normalized',
choices=readablewaves)
            if len(toberated)==0:
                waveratio=False
            else:
                for sinwave in toberated:
                    subreadablewaves=[]
                    sinwavepage=allwaves[readablewaves.index(sinwave)][0]
                    for fullwave in range(len(allwaves)):
                        if allwaves[fullwave][0]==sinwavepage:
                            subreadablewaves.append(readablewaves[fullwave])
                    raters=eg.multchoicebox(msg='Pick all the objects to normalize '+sinwave+'
BY',choices=subreadablewaves)
                    if len(raters)==0:
                        pass
                    else:
                        for eachrate in raters:

waveratio.append((allwaves[readablewaves.index(sinwave)],allwaves[readablewaves.index(eachr
ate)]))

                if len(waveratio)==0:
                    waveratio=False

        else:
            waveratio=False

        if eg.ynbox(msg='Do you wish to save these settings as a default?'):
            defname=eg.enterbox('Enter a descriptive name for this default', strip=False)
            prevdef[defname]=tocompinit,dist,kidanalysis,normanalysis,fractanalysis,waveratio

        prevdef.close()
    else:
        tocompinit, dist, kidanalysis, normanalysis,fractanalysis,waveratio=default

    #run analysis of distances
    if tocompinit!=False:
        pairdata={}
        for obj in tocompinit[0]:
```

```
pairdata[obj]=pullallvals(pandadict[obj],["Location_Center_X","Location_Center_Y"],tocompini
t[1])
        #print pairdata
        pairslist=[]
        for ind1 in tocompinit[0]:
            templist=[]
            for ind2 in tocompinit[0]:
                templist.append((ind1,ind2))
            pairslist.append(templist)
        #print pairslist
        for startind in pairslist:
            #print startind
            activesheetcols=numcoldict[startind[0][0]]
            wrsheet=w.get_sheet_by_name(startind[0][0])
            for pair in startind:
                #print pair
                activesheetletter=get_column_letter(activesheetcols)
                wrsheet.cell('%s%s' % (activesheetletter, 1)).value=str(pair[0])+'To'+str(pair[1])
                distance=compare2dicts(pairdata[pair[0]],pairdata[pair[1]],dist)
                for i in distance.keys():
                    wrsheet.cell('%s%s' % (activesheetletter, i+2)).value=distance[i]
                numcoldict[startind[0][0]]+=1
                activesheetcols+=1

    if kidanalysis!=False:
        for i in kidanalysis:
            #print i, headingdict
            activesheetcols=numcoldict[i[0][0]]
            activesheetletter=get_column_letter(activesheetcols)
            wrsheet=w.get_sheet_by_name(i[0][0])
            readsheet=pandadict[i[0][0]]
            numerator=readsheet[i[0][1]]
            denominator=readsheet[i[1][1]]
            wrsheet.cell('%s%s' % (activesheetletter,
1)).value='%'+HXM.middleofdelim(i[0][1])+'/'+HXM.middleofdelim(i[1][1])
            #print len(numerator)
            for j in range(len(numerator)):
                if denominator[j]!=0:
                    #print
j,(100.0*numerator[j]/denominator[j]),type(100.0*numerator[j]/denominator[j])
                    wrsheet.cell('%s%s' % (activesheetletter,
j+2)).value=float(100.0*numerator[j]/denominator[j])
                else:
                    #print j,"div by 0"
                    wrsheet.cell('%s%s' % (activesheetletter, j+2)).value=0.0
```

```
            numcoldict[i[0][0]]+=1

    if normanalysis!=False:
        for i in normanalysis:
            activesheetcols=numcoldict[i[0][0]]
            activesheetletter=get_column_letter(activesheetcols)
            wrsheet=w.get_sheet_by_name(i[0][0])
            readsheet=pandadict[i[0][0]]
            parentsheet=pandadict[i[1][0]]
            images=readsheet['ImageNumber']
            nucs=readsheet['Parent_'+i[1][0]]
            values=readsheet[i[0][1]]
            #print images,nucs,values
            if i[2]=='Subtraction':
                subfact=makenormfactorsub(parentsheet,i[1][1])
                normname=i[1][0]+'-'+HXM.stripheaderonce(i[1][1])
                wrsheet.cell('%s%s' % (activesheetletter, 1)).value=HXM.stripheaderonce(i[0][1])+'
minus '+normname
                for j in range(len(values)):
                    wrsheet.cell('%s%s' % (activesheetletter, j+2)).value=values[j]-
subfact[(images[j],nucs[j])]
            else:
                #print parlist[strippedparlist.index(HXM.stripheaderonce(i[1][0]))]
                #print i[1][1]
                normname=i[1][0]+'-'+HXM.stripheaderonce(i[1][1])
                divfact=makenormfactordiv(parentsheet,i[1][1])
                wrsheet.cell('%s%s' % (activesheetletter, 1)).value=HXM.stripheaderonce(i[0][1])+'
divided by '+normname
                for j in range(len(values)):
                    wrsheet.cell('%s%s' % (activesheetletter,
j+2)).value=values[j]/divfact[(images[j],nucs[j])]
            numcoldict[i[0][0]]+=1

    if fractanalysis!=False:
        for i in fractanalysis:
            activesheetcols=numcoldict[i[0][0]]
            activesheetletter=get_column_letter(activesheetcols)
            wrsheet=w.get_sheet_by_name(i[0][0])
            readsheet=pandadict[i[0][0]]
            childsheet=pandadict[i[1][0]]
            denominator=readsheet[i[0][1]]
            images=readsheet['ImageNumber']
            nucs=readsheet['ObjectNumber']
            nomdic=maketotal(childsheet,i[1][1],i[0][0])
            wrsheet.cell('%s%s' % (activesheetletter, 1)).value='% of
'+HXM.stripheaderonce(i[0][1])+' of '+i[1][0]+'/'+i[0][0]
```

```
            for j in range(len(denominator)):
                if (images[j],nucs[j]) in nomdic.keys():
                    numerator=nomdic[(images[j],nucs[j])]
                else:
                    numerator=0
                if denominator[j]!=0:
                    wrsheet.cell('%s%s' % (activesheetletter,
j+2)).value=100.0*numerator/denominator[j]
                else:
                    wrsheet.cell('%s%s' % (activesheetletter, j+2)).value=0.0
            numcoldict[i[0][0]]+=1

    if waveratio!=False:
        for i in waveratio:
            activesheetcols=numcoldict[i[0][0]]
            activesheetletter=get_column_letter(activesheetcols)
            wrsheet=w.get_sheet_by_name(i[0][0])
            readsheet=pandadict[i[0][0]]
            wrsheet.cell('%s%s' % (activesheetletter, 1)).value='Ratio of
'+HXM.stripheaderonce(i[0][1])+' to '+HXM.stripheaderonce(i[1][1])
            for j in range(readsheet.shape[0]):
                #print (j,headingdict[HXM.stripheaderonce(i[0][0])].index(i[0][1]))
                try:
                    ratio=float(readsheet.get_value(j,i[0][1])/readsheet.get_value(j,i[1][1]))
                except ZeroDivisionError:
                    ratio=""
                wrsheet.cell('%s%s' % (activesheetletter, j+2)).value=ratio
            numcoldict[i[0][0]]+=1

    if default==None:
        r=os.path.join(directory,csvoutname+'.xlsx') #Save the excel file
    else:
        r=os.path.join(os.path.split(directory)[0],csvoutname+'.xlsx') #Save the excel file
    todel=w.get_sheet_by_name('Sheet')
    w.remove_sheet(todel)
    w.save(r)
    #w=load_workbook(r, use_iterators=True)

    #w.save(r)
```

def runcsvcombiner():

```
    if not eg.ynbox('Do you want to do the EXACT same operations on multiple data sets (ie
use batch mode)?'):
        direct=eg.diropenbox('Which folder has your output files in it?')
        csvout=eg.filesavebox('Where do you want to save your output file?',filetypes=["*.xlsx"])
```

```
        csvcombinermain(direct,csvout)
    else:
        masterdir=eg.diropenbox("Which is the main experiment folder that has all of your
subfolders with output files inside? (You'll choose which subfolders to use in a minute)")
        alldir=[]
        for i in os.listdir(masterdir):
            subfold=os.path.join(masterdir,i)
            if os.path.isdir(subfold):
                alldir.append(subfold)
        touse=eg.multchoicebox('Which of these do you want to analyze?',choices=alldir)
        prevdef=shelve.open(os.path.join(os.curdir,'csvshelf'),writeback=True)
        pickdefault=eg.choicebox(msg='Pick the default you wish to use', choices=prevdef.keys())
        default=prevdef[pickdefault]
        for folder in touse:
            csvcombinermain(folder,os.path.split(folder)[1],default)

if __name__=='__main__':
    runcsvcombiner()
```

OutputSigsXL.py

```
"""Does statistical analysis of user requested factors, and passes info to the graphing modules if

requested"""

from scipy import stats
import HandyXLModulesPruned as HXM
import easygui as eg
import shelve
import os
import OutputSigsXLGraphing as OG
from statsmodels.sandbox.stats.multicomp import multipletests
from openpyxl import Workbook
import pandas as pd
from openpyxl.cell import get_column_letter
from openpyxl.styles import PatternFill, Style, Alignment
from openpyxl.styles.borders import Border, Side
#import gc


lightbluefill = PatternFill(start_color='FFA6CEE3',end_color='FFA6CEE3',fill_type='solid')
darkbluefill = PatternFill(start_color='FF1F78B4',end_color='FF1F78B4',fill_type='solid')
lightredfill = PatternFill(start_color='FFFB9A99',end_color='FFFB9A99',fill_type='solid')
darkredfill = PatternFill(start_color='FFE31A1C',end_color='FFE31A1C',fill_type='solid')
lightgreenfill = PatternFill(start_color='FFB2DF8A',end_color='FFB2DF8A',fill_type='solid')
darkgreenfill = PatternFill(start_color='FF33A02C',end_color='FF33A02C',fill_type='solid')
thin_border = Border(left=Side(style='thin'), right=Side(style='thin'), top=Side(style='thin'),
bottom=Side(style='thin'))
wraptrue=Alignment(wrap_text=True)

othercells=Style(border=thin_border,alignment=wraptrue)
vsmall=Style(border=thin_border,alignment=wraptrue,fill=darkbluefill)
small=Style(border=thin_border,alignment=wraptrue,fill=lightbluefill)
big=Style(border=thin_border,alignment=wraptrue,fill=lightredfill)
vbig= Style(border=thin_border,alignment=wraptrue,fill=darkredfill)
onemedzero=Style(border=thin_border,alignment=wraptrue,fill=lightgreenfill)
twomedzero=Style(border=thin_border,alignment=wraptrue,fill=darkgreenfill)

def pickafilter(typeofthing,param,questiontoask,unfiltvalues,parentinfo=None):
    filterkeys=[]
    filtervals=[]
    intentered=False
    howtofilt=eg.indexbox('How do you want to filter '+questiontoask+'?',choices=('By a
numerical value', 'By a percentile'))
    if howtofilt==0: #if they say by numverical value
```

```
        while intentered==False:
            filt=eg.multenterbox(msg='How do you want to filter
'+questiontoask+'?',fields=('Operator- choose from ==, !=, <,>, <=,>=','Value'))#let the user
input the filter they want
            try:
                isint=int(filt[1])
                filterkeys.append(param+':'+filt[0]+filt[1])
                filtervals.append(unfiltvalues[:2]+[typeofthing,filt])
                intentered=True
            except:
                intentered=False
    else:
        toporbottom=eg.indexbox(msg='Do you want to look above or below a specified
percentile of' +param+'?',choices=('Above','Below','One set of each'))
        if toporbottom==0:
            while intentered==False:
                toppercent=eg.enterbox(msg='Enter the number for the percentile of' +param+' you
want to look above (ie above the _____ percentile')
                try:
                    isint=int(toppercent)
                    toappendtos=['Percentile>=',toppercent]
                    filterkeys.append(param+':'+toappendtos[0]+toappendtos[1])
                    filtervals.append(unfiltvalues[:2]+[typeofthing,toappendtos])
                    intentered=True
                except:
                    intentered=False
        elif toporbottom==1:
            while intentered==False:
                botpercent=eg.enterbox(msg='Enter the number for the percentile of ' +param+'
you want to look below (ie below the _____ percentile')
                try:
                    isint=int(botpercent)
                    toappendtos=['Percentile<=',botpercent]
                    filterkeys.append(param+':'+toappendtos[0]+toappendtos[1])
                    filtervals.append(unfiltvalues[:2]+[typeofthing,toappendtos])
                    intentered=True
                except:
                    intentered=False
        else:
            intenteredsecond=False
            while intentered==False:
                toppercent=eg.enterbox(msg='Enter the number for the percentile of ' +param+'
you want to look above (ie above the _____ percentile')
                try:
                    isint=int(toppercent)
                    toappendtos=['Percentile>=',toppercent]
```

```
                filterkeys.append(param+':'+toappendtos[0]+toappendtos[1])
                filtervals.append(unfiltvalues[:2]+[typeofthing,toappendtos])
                intentered=True
            except:
                intentered=False
        while intenteredsecond==False:
            botpercent=eg.enterbox(msg='Enter the number for the percentile of ' +param+'
you want to look below (ie below the _____ percentile')
            try:
                isint=int(botpercent)
                toappendtos=['Percentile<=',botpercent]
                filterkeys.append(param+':'+toappendtos[0]+toappendtos[1])
                filtervals.append(unfiltvalues[:2]+[typeofthing,toappendtos])
                intenteredsecond=True
            except:
                intenteredsecond=False
    #print filterkeys,filtervals
    return filterkeys,filtervals

def arrangedivsin(book,thingstopull):
    statstorun={}
    for x in thingstopull: #for x (or filtered and unfiltered if the user so chose)
        if x[2]==1: #if the user chose to filter based on a numerical value
            if 'Percentile' in x[3][0]:
                if float(x[3][1])>1:
                    numfloat=0.01*float(x[3][1])
                else:
                    numfloat=float(x[3][1])
                statstorun[x[0]+'-'+x[1]+'('+x[3][0]+x[3][1]+')']=(book[x[0]].query(x[1]+x[3][0][-
2:]+str(book[x[0]][x[1]].quantile(numfloat)))[x[1]],x[-1])
            else:
                statstorun[x[0]+'-
'+x[1]+'('+x[3][0]+x[3][1]+')']=(book[x[0]].query(x[1]+x[3][0]+x[3][1])[x[1]],x[-1])
        elif x[2]==2:
            for relativesort in x[3]:
                #print relativesort
                if len(relativesort)==3: #if we're filtering based on another measure of that
parameter (including # of children)
                    #[[[paramdict[param][0],eachfilt,filtvals[waystofilt][3]]]]
                    if 'Percentile' in relativesort[2][0]:
                        if float(relativesort[2][1])>1:
                            numfloat=0.01*float(relativesort[2][1])
                        else:
                            numfloat=float(relativesort[2][1])
```

```
                    statstorun[x[0]+'-'+x[1]+'(have'+relativesort[2][0]+relativesort[2][1]+'
'+relativesort[1]+')']=(book[x[0]].query(relativesort[1]+relativesort[2][0][-
2:]+str(book[x[0]][relativesort[1]].quantile(numfloat)))[x[1]],x[-1])
                else:
                    statstorun[x[0]+'-'+x[1]+'(have'+relativesort[2][0]+relativesort[2][1]+'
'+relativesort[1]+')']=(book[x[0]].query(relativesort[1]+relativesort[2][0]+relativesort[2][1])[x[
1]],x[-1])
            elif len(relativesort)==5: #if we're filtering based on a parental factor
                #[[paramdict[param][0],rels,rels[7:],relthings,filtvals[waystofilt][3]]]]

                #sixtytwo=pandabook['Nuclei'].query('Children_PNA_Count'+'=='+'62')
                #okaynuclei=zip(sixtytwo['ImageNumber'],sixtytwo['ObjectNumber'])

#PNAframe=pandabook['PNA'].loc[:,['ImageNumber','Parent_Nuclei','Location_Center_X']]
                #PNAframe['tuple']=zip(PNAframe['ImageNumber'],PNAframe['Parent_Nuclei'])
                #PNAmask=PNAframe.isin(okaynuclei).any(1)
                #subPNA=PNAframe[PNAmask]
                if 'Percentile' in relativesort[4][0]:
                    if float(relativesort[4][1])>1:
                        numfloat=0.01*float(relativesort[4][1])
                    else:
                        numfloat=float(relativesort[4][1])
                    parentframe=book[relativesort[2]].query(relativesort[3]+relativesort[4][0][-
2:]+str(book[relativesort[2]][relativesort[3]].quantile(numfloat)))
                else:

parentframe=book[relativesort[2]].query(relativesort[3]+relativesort[4][0]+relativesort[4][1])

parentimnucokay=zip(parentframe['ImageNumber'],parentframe['ObjectNumber'])
                measureframe=book[x[0]].loc[:,['ImageNumber',relativesort[1],x[1]]]

measureframe['famtuple']=zip(measureframe['ImageNumber'],measureframe[relativesort[1]])
                measuremask=measureframe.isin(parentimnucokay).any(1)
                statstorun[x[0]+'-'+x[1]+'(have parent '+relativesort[2]+' whose
'+relativesort[3]+' is '
+relativesort[4][0]+relativesort[4][1]+')']=(measureframe[measuremask][x[1]],x[-1])
            elif len(relativesort)==6: #if we're filtering based on having children that match a
certain measurement

#[[[paramdict[param][0],rels,rels[9:],myparentcolumn,relthings,filtvals[waystofilt][3]]]]
                if 'Percentile' in relativesort[5][0]:
                    if float(relativesort[5][1])>1:
                        numfloat=0.01*float(relativesort[5][1])
                    else:
                        numfloat=float(relativesort[5][1])
```

```
                childframe=book[relativesort[2]].query(relativesort[4]+relativesort[5][0][-
2:]+str(book[relativesort[2]][relativesort[4]].quantile(numfloat)))
            else:

childframe=book[relativesort[2]].query(relativesort[4]+relativesort[5][0]+relativesort[5][1])
            childimnucokay=zip(childframe['ImageNumber'],childframe[relativesort[3]])
            measureframe=book[x[0]].loc[:,['ImageNumber','ObjectNumber',x[1]]]

measureframe['famtuple']=zip(measureframe['ImageNumber'],measureframe['ObjectNumber'])
            measuremask=measureframe.isin(childimnucokay).any(1)
            statstorun[x[0]+'-'+x[1]+'(have child '+relativesort[2]+' whose '+relativesort[4]+'
is ' +relativesort[5][0]+relativesort[5][1]+')']=(measureframe[measuremask][x[1]],x[-1])

        else:
            statstorun[x[0]+'-'+x[1]]=(book[x[0]][x[1]],x[-1])
    return statstorun

def choosewhichstatssin(basebook,silent=False,whichdefaultext=False):
    pulldefs=shelve.open(os.path.join(os.curdir,'OutputSigsXLshelf'),writeback=True)
    if silent==True:
        xparams=pulldefs[whichdefaultext]
    else:
        if eg.ynbox(msg='Do you want to use the defaults?'):
            whichdefault=eg.choicebox('Which default set do you want to
use?',choices=pulldefs.keys())
            xparams=pulldefs[whichdefault]

        else:
            #Pull all the possible parameters
            total=[]
            totaldict={}
            for eachsheet in basebook.keys():
                colheads=basebook[eachsheet].columns
                for eachcol in colheads:
                    total.append(str(eachsheet)+'-'+str(eachcol))
                    totaldict[str(eachsheet)+'-'+str(eachcol)]=[eachsheet,eachcol]

            xparams=[]
            paramdict={}
            firstselected=eg.multchoicebox(msg='Select parameters',choices=total)
            addmore=True
            while addmore==True:
                eg.textbox(msg='So far you have selected the following parameters. On the next
screen you will be asked if you want to add any more.',text='\n'.join(firstselected))
                moretodo=eg.ynbox('Do you need to add any more parameters?')
                if moretodo==0:
```

```
                addmore=False
            else:
                gx=eg.multchoicebox(msg='Which of these previously unselected parameters do
you want to add?', choices=list(set(total)-set(firstselected)))
                firstselected+=gx
                addmore=True
        for param in firstselected:
            paramdict[param]=totaldict[param]+[None]
        donefiltering=False
        while donefiltering==False:
            filtf=eg.multchoicebox(msg='Do you want to filter any of
these?',choices=(paramdict.keys()))
            for param in paramdict.keys():
                if param in filtf:
                    heads=basebook[param[:param.index('-')]].columns
                    numorstr=eg.indexbox(('How do you want to filter '+param),choices=['By a
specific value or percentile','By another measure of that same object','By relationship to another
object'])
                    if numorstr==0: #if they say by numverical value
                        filtkeys,filtvals=pickafilter(1,param,param,paramdict[param])
                        for waystofilt in range(len(filtkeys)):
                            paramdict[filtkeys[waystofilt]]=filtvals[waystofilt]
                    elif numorstr==1:
                        whichfilt=eg.multchoicebox('Which other measure(s) do you want to sort
'+param+' by?',choices=heads)
                        for eachfilt in whichfilt:
                            filtkeys,filtvals=pickafilter(2,param+': by '+eachfilt,eachfilt+' in breaking
down '+param,paramdict[param])
                            #eachfiltsort=eg.multenterbox(msg='What filter do you want to put on
'+str(eachfilt)+'?',fields=('Operator- choose from ==, !=, <,>, <=,>=','Value'))
                            for waystofilt in range(len(filtkeys)):

paramdict[filtkeys[waystofilt]]=filtvals[waystofilt][:3]+[[[paramdict[param][0],eachfilt,filtvals[way
stofilt][3]]]]
                    elif numorstr==2:
                        relatives=[]
                        for eachhead in heads:
                            if 'Children' in eachhead:
                                relatives.append(eachhead)
                            elif 'Parent' in eachhead:
                                relatives.append(eachhead)
                        whichrels=eg.multchoicebox('Which related objects do you want to sort
'+param+' by?',choices=relatives)
                        for rels in whichrels:
                            if 'Children' in rels:
                                relname=rels[9:]
```

```
                        else:
                            relname=rels[7:]
                        relheads=basebook[rels].columns
                        if 'Children' in rels:
                            myparentcolumn='Parent_'+str(paramdict[param][0])
                        whichrelthings=eg.multchoicebox('Which parameter of '+relname+' do
you want to sort '+param+' by?',choices=relheads)
                        for relthings in whichrelthings:
                            filtkeys,filtvals=pickafilter(2,param+': by '+rels+ '-'+relthings,relthings+'
of '+relname+' as a way to break down'+param,paramdict[param])
                            if 'Parent' in rels:
                                for waystofilt in range(len(filtkeys)):

paramdict[filtkeys[waystofilt]]=filtvals[waystofilt][:3]+[[[paramdict[param][0],rels,rels[7:],relthin
gs,filtvals[waystofilt][3]]]]
                            else:
                                for waystofilt in range(len(filtkeys)):

paramdict[filtkeys[waystofilt]]=filtvals[waystofilt][:3]+[[[paramdict[param][0],rels,rels[9:],mypare
ntcolumn,relthings,filtvals[waystofilt][3]]]]

            allparams=paramdict.keys()
            allparams.sort()
            eg.textbox(msg='So far you have generated the following parameters. On the next
screen you will be asked if you want to add more filters.',text='\n'.join(allparams))
            if eg.ynbox('Do you want to add any other filters?'):
                donefiltering=False
            else:
                donefiltering=True
        removefromparams=eg.multchoicebox('Do you want to remove any parameters from
this list? This will be your last chance to do so.',choices=paramdict.keys())
        for toremove in removefromparams:
            paramdict.pop(toremove)
        graphaslog=eg.multchoicebox('Do you want to graph any of the following in log scale
(alone or in addition to linear scale)?',choices=paramdict.keys())
        if graphaslog!=[]:
            graphonlylog=eg.multchoicebox('Which of these do you want to graph in log scale
ONLY? (Unselected items will be graphed as linear and log)',choices=graphaslog )
        else:
            graphonlylog=[]
        for param in paramdict:
            if param in graphonlylog:
                paramdict[param].append('log')
            elif param in graphaslog:
                paramdict[param].append('loglin')
            else:
```

```
                paramdict[param].append('lin')
            xparams.append(paramdict[param])
        xparams.sort()
        if eg.ynbox('Do you want to save these settings as a new default?'):
            newdefname=eg.enterbox(msg='Give this default a descriptive identifier')
            pulldefs[newdefname]=xparams

    pulldefs.close()
    #print g
    return xparams

def findtheps(list1,baseindex):
    basemed=list1[baseindex].median()
    #print 'basemed',basemed
    changelist=[]
    mwplist=[]
    ksplist=[]
    for i in range(len(list1)):
        if i!=baseindex:
            if basemed!=0:
                changelist.append((100.0*(list1[i].median()-basemed))/basemed)
            else:
                changelist.append('n.d.-> base median=0')
            try:
                #print stats.mannwhitneyu(list1[baseindex],list1[i])
                mwplist.append(stats.mannwhitneyu(list1[baseindex],list1[i])[1])
            except ValueError:
                mwplist.append(1)
                print 'Raised Error'
            ksplist.append(stats.ks_2samp(list1[baseindex],list1[i])[1])

        else:
            changelist.append('Baseline')
            mwplist.append(1)
            ksplist.append(1)
    return changelist,mwplist,ksplist
    #print mwplist,ksplist


def makethesubsets(histdic,listofstats, readablelabellist,grouplist):
    subhistdic={}
    subreadablelist=[]
    for eachgroup in grouplist:
        subreadablelist.append(readablelabellist[eachgroup])
    for eachstat in listofstats:
        subhistdic[eachstat]=[]
```

```
        for eachgroup in grouplist:
            subhistdic[eachstat].append(histdic[eachstat][eachgroup])
    return subhistdic,subreadablelist

def
runeachone(writebook,names,paths,whichfiles,runcount,outdir,outfilename,silent=
False,whichisbase=False,whichdefault=False):
    histdic={}
    listofstats=[]
    graphlin=[]
    graphlog=[]
    if len(whichfiles)==1:
        metadataformatrix=False
        ident=outfilename[:-5]
        while len(ident)>31:
            if silent==False:
                ident=eg.enterbox(ident+' is too long to be a sheet name- enter a shorter name')
            else:
                ident=ident[:30]

basebook=pd.read_excel(paths[names.index(whichfiles[0])],sheetname=None,convert_float=Fal
se)
        if whichfiles[0][-8:-5]=='Out':
            readablelabellist=[whichfiles[0][:-8]]
        else:
            readablelabellist=[whichfiles[0][:-5]]
        whichstats=choosewhichstatssin(basebook,silent=silent,whichdefaultext=whichdefault)
        #print whichstats
        basevals=arrangedivsin(basebook,whichstats)
        for i in basevals:
            #print basevals[i][0]
            #print i, whichstats[0][i],basevals[i]
            if len(basevals[i][0])>1:
                histdic[i]=[basevals[i][0]]
                if basevals[i][1]=='log':
                    graphlog.append(i)
                elif basevals[i][1]=='loglin':
                    graphlin.append(i)
                    graphlog.append(i)
                else:
                    graphlin.append(i)
        listofstats=histdic.keys()
        listofstats.sort()
    else:
        if silent==False:
            baseline=eg.choicebox(msg='Which file is the baseline?', choices=whichfiles)
```

```
        else:
            baseline=whichisbase
        baseindex=whichfiles.index(baseline)
        ident=baseline[:-5]+' as baseline'
        while len(ident)>31:
            if silent==False:
                ident=eg.enterbox(ident+' is too long to be a sheet name- enter a shorter name')
            else:
                ident=ident[:30]
        readablelabellist=[]
        for i in whichfiles:
            if i[-8:-5]=='Out':
                readablelabellist.append(i[:-8])
            else:
                readablelabellist.append(i[:-5])

basebook=pd.read_excel(paths[names.index(whichfiles[baseindex])],sheetname=None,convert_
float=False)
        metadataformatrix=False
        if silent==False:
            if len(whichfiles)>15:
                usemetadata=eg.ynbox('You have a lot of conditions.  Do you want to use metadata
to split them into a matrix with 2 axes? This is also how your data will be broken down to be
graphed')
                if usemetadata:
                    whichmeta=[]
                    while len(whichmeta)!=2:
                        whichmeta=eg.multchoicebox(msg='Select exactly 2
parameters',choices=list(basebook['Image'].filter(regex='Metadata').columns))
                    metadataformatrix=True
                    metadata1={}
                    metadata2={}

        whichstats=choosewhichstatssin(basebook,silent=silent,whichdefaultext=whichdefault)
        basevals=arrangedivsin(basebook,whichstats)
        wrsheet=writebook.get_active_sheet()
        if wrsheet.title=='Sheet':
            wrsheet.title=ident
        else:
            wrsheet=writebook.create_sheet(title=ident)
        wrsheet.page_setup.orientation = wrsheet.ORIENTATION_LANDSCAPE
        wrsheet.cell('A1').value=ident
        wrsheet['A1'].style=othercells
        wrsheet.column_dimensions['A'].width = 35

        #print whichstats[0]
```

```
        #print len(basevals),len(whichstats[0])
        for i in basevals:
            #print basevals[i][0]
            #print i, whichstats[0][i],basevals[i]
            if len(basevals[i][0])>1:
                histdic[i]=[]
                if basevals[i][1]=='log':
                    graphlog.append(i)
                elif basevals[i][1]=='loglin':
                    graphlin.append(i)
                    graphlog.append(i)
                else:
                    graphlin.append(i)

        listofstats=histdic.keys()
        listofstats.sort()
            #else:
                #print basevals[i][0]," no objects met that filter"
        for i in range(len(whichfiles)):
            if metadataformatrix==False:
                wrsheet.cell(row=1,column=i+2,value=whichfiles[i])
                wrsheet['%s%s' % (get_column_letter(i+2),1)].style=othercells
                wrsheet.column_dimensions[get_column_letter(i+2)].width = 90.0/len(whichfiles)


compbook=pd.read_excel(paths[names.index(whichfiles[i])],sheetname=None,convert_float=Fa
lse)
            compvals=arrangedivsin(compbook,whichstats)
            for m in compvals:
                if m in histdic.keys():
                    histdic[m].append(compvals[m][0])

            if metadataformatrix==True:
                metadatapoint1=compbook['Image'][whichmeta[0]][0]
                metadatapoint2=compbook['Image'][whichmeta[1]][0]
                if metadatapoint1 not in metadata1.keys():
                    metadata1[metadatapoint1]=[i]
                else:
                    metadata1[metadatapoint1]+=[i]
                if metadatapoint2 not in metadata2.keys():
                    metadata2[metadatapoint2]=[i]
                else:
                    metadata2[metadatapoint2]+=[i]

        #print histdic
        if metadataformatrix==True:
```

```
            metadatalist1=metadata1.keys()
            metadatalist1.sort()
            metadatalist2=metadata2.keys()
            metadatalist2.sort()
            horizchoice=eg.choicebox('Select which of these sets of metadata you want as the
horizontal axis. The other will be the vertical axis.',
choices=[str(metadatalist1),str(metadatalist2)])
            if horizchoice==str(metadatalist1):
                horizdict=metadata1
                horizlist=metadatalist1
                vertdict=metadata2
                vertlist=metadatalist2
            else:
                vertdict=metadata1
                vertlist=metadatalist1
                horizdict=metadata2
                horizlist=metadatalist2
            matrixdict={}
            #print horizdict,vertdict
            for eachtreat in range(len(whichfiles)):
                for horizkey in horizdict.keys():
                    if horizkey!='Name':
                        if eachtreat in horizdict[horizkey]:
                            horizval=horizkey
                for vertkey in vertdict.keys():
                    if vertkey!='Name':
                        if eachtreat in vertdict[vertkey]:
                            vertval=vertkey

matrixdict[eachtreat]=[readablelabellist[eachtreat],horizlist.index(horizval),vertlist.index(vertval)
]

        #print histdic
        mannwhitforcorr=[]
        ksforcorr=[]
        percentchangesnested=[]
        for i in range(len(listofstats)):
            if metadataformatrix==False:
                wrsheet.cell(row=i+2,column=1,value=listofstats[i])
                wrsheet['%s%s' % ('A',i+2)].style=othercells
            percentchange,mannwhitney,ks=findtheps(histdic[listofstats[i]],baseindex)
            percentchangesnested.append(percentchange)
            mannwhitforcorr+=mannwhitney
            ksforcorr+=ks
        #print multipletests(mannwhitforcorr,method='h')
        try:
```

```
            mwbools,mwvals,z,y=multipletests(mannwhitforcorr,method='fdr_bh')
            #print len(mannwhitforcorr),len(mwbools), len(mwvals)
            #print mannwhitforcorr, mwbools, mwvals
        #except:
        except ValueError:
            mwbools=len(mannwhitforcorr)*[True]
            mwvals=len(mannwhitforcorr)*[1.000]
        try:
            ksbools,ksvals,z,y=multipletests(ksforcorr,method='fdr_bh')
        #except:
        except ValueError:
            ksbools=len(ksforcorr)*[True]
            ksvals=len(ksforcorr)*[1.000]
        itercount=0
        for parameter in range(len(percentchangesnested)):
            #print sigtest
            if metadataformatrix==False:
                for treatment in range(len(percentchangesnested[parameter])):
                    change=percentchangesnested[parameter][treatment]
                    if type(change)==str:
                        towrite=', '+change
                        changeneg='zero'
                    else:
                        if change<0:
                            changeneg=True
                        else:
                            changeneg=False
                        towrite=', '+"%0.2f" %change +'% change in median'
                    if mwbools[itercount]==True:
                        if ksbools[itercount]==True:
                            wrsheet.cell(row=parameter+2,column=treatment+2,value='M.W.
<'+"%0.3f" %mwvals[itercount]+', K.S. <'+"%0.3f" %ksvals[itercount]+towrite)
                            if changeneg==True:
                                wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=vsmall
                            elif changeneg==False:
                                wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=vbig
                            else:
                                wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=twomedzero
                        else:
                            wrsheet.cell(row=parameter+2,column=treatment+2,value='M.W.
<'+"%0.3f" %mwvals[itercount]+towrite)
                            if changeneg==True:
```

```
                        wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=small
                    elif changeneg==False:
                        wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=big
                    else:
                        wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=onemedzero
                else:
                    if ksbools[itercount]==True:
                        wrsheet.cell(row=parameter+2,column=treatment+2,value='K.S. <'+"%0.3f"
%ksvals[itercount]+towrite)
                    if changeneg==True:
                        wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=small
                    elif changeneg==False:
                        wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=big
                    else:
                        wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=onemedzero
                else:
                    wrsheet.cell(row=parameter+2,column=treatment+2,value='n.s.'+towrite)
                    wrsheet['%s%s' %
(get_column_letter(treatment+2),parameter+2)].style=othercells
                itercount+=1
        else:
            indexrow=(parameter*(len(vertlist)+2))+2
            wrsheet.cell(row=indexrow,column=1,value=listofstats[parameter])
            wrsheet['%s%s' % ('B',indexrow)].style=othercells
            wrsheet.column_dimensions['B'].width = 90.0/(len(horizlist)+1)
            for eachhoriz in range(len(horizlist)):
                wrsheet.cell(row=indexrow,column=eachhoriz+3,value=horizlist[eachhoriz])
                wrsheet['%s%s' % (get_column_letter(eachhoriz+3),indexrow)].style=othercells
                wrsheet.column_dimensions[get_column_letter(eachhoriz+3)].width =
90.0/(len(horizlist)+1)
            for eachvert in range(len(vertlist)):
                wrsheet.cell(row=indexrow+1+eachvert,column=2,value=vertlist[eachvert])
                wrsheet['%s%s' % ('B',indexrow+1+eachvert)].style=othercells
            for treatment in range(len(percentchangesnested[parameter])):
                change=percentchangesnested[parameter][treatment]
                if type(change)==str:
                    towrite=', '+change
                    changeneg='zero'
                else:
                    if change<0:
```

```
                    changeneg=True
                else:
                    changeneg=False
                towrite=', '+"%0.2f" %change +'% change in median'
            thistreatrow=indexrow+matrixdict[treatment][2]+1
            thistreatcol=matrixdict[treatment][1]+3
            if mwbools[itercount]==True:
                if ksbools[itercount]==True:
                    wrsheet.cell(row=thistreatrow,column=thistreatcol,value='M.W. <'+"%0.3f"
%mwvals[itercount]+', K.S. <'+"%0.3f" %ksvals[itercount]+towrite)
                    if changeneg==True:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=vsmall
                    elif changeneg==False:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=vbig
                    else:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=twomedzero
                else:
                    wrsheet.cell(row=thistreatrow,column=thistreatcol,value='M.W. <'+"%0.3f"
%mwvals[itercount]+towrite)
                    if changeneg==True:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=small
                    elif changeneg==False:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=big
                    else:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=onemedzero
            else:
                if ksbools[itercount]==True:
                    wrsheet.cell(row=thistreatrow,column=thistreatcol,value='K.S. <'+"%0.3f"
%ksvals[itercount]+towrite)
                    if changeneg==True:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=small
                    elif changeneg==False:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=big
                    else:
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=onemedzero
                else:
                    wrsheet.cell(row=thistreatrow,column=thistreatcol,value='n.s.'+towrite)
```

```
                        wrsheet['%s%s' %
(get_column_letter(thistreatcol),thistreatrow)].style=othercells
                    itercount+=1

    writebook.save(os.path.join(outdir,outfilename))
    #gc.collect()
    if runcount==0:
        if silent==True:
            OG.dothegraphing(histdic,outdir,outfilename,listofstats,readablelabellist,
graphlog,graphlin)
        else:
            if eg.ynbox('Do you want to run the histograms?'):
                if metadataformatrix==False:
                    OG.dothegraphing(histdic,outdir,outfilename,listofstats,readablelabellist,
graphlog,graphlin)

                else:
                    mastermeta=dict(horizdict.items()+vertdict.items())
                    for eachmetagroup in mastermeta.keys():

                        subhistdic,subreadablelist=makethesubsets(histdic,listofstats,
readablelabellist,mastermeta[eachmetagroup])

OG.dothegraphing(subhistdic,outdir,outfilename,listofstats,subreadablelist,graphlog,graphlin,nam
eappendix=eachmetagroup)
                        #gc.collect()
                #histsheet.flush_row_data()

            if eg.ynbox('Do you want to run any scatter plots on the parameters just analyzed?'):
                OG.dothescattergraphing(histdic,outdir,outfilename,listofstats,readablelabellist)



def dothestuff(direct=0):
    if direct==0:
        direct=eg.diropenbox()
    a=HXM.findexcel(direct)
    b=[]
    c=[]
    for i in a:
        b.append(i[0])
        c.append(i[1])
    whichfiles=eg.multchoicebox(msg='Which input files do you want to use?', choices=b)
    #print b,c,whichfiles
    w=eg.filesavebox(msg='What do you want to name the output
file?',filetypes=["*.xlsx"])+'.xlsx'
```

```
    writebook=Workbook() #make a new file
    outdir,filename=os.path.split(w)

    runcount=0
    another=True
    while another==True:
        runeachone(writebook,b,c,whichfiles,runcount,outdir,filename)
        runcount+=1
        if not eg.ynbox('Do you want to run the same input and output files but with a different
baseline?'):
            another=False
    writebook.save(w)


if __name__=='__main__':
    dothestuff()
```

def dothestuffsilently(b,c,whichfiles,w,whichbase,whichdef):

```
    writebook=Workbook() #make a new file
    outdir,filename=os.path.split(w)

runeachone(writebook,b,c,whichfiles,0,outdir,filename,silent=True,whichdefault=whichdef,which
isbase=whichbase)
    writebook.save(w)
```

OutputSigsXLGraphing.py

```
"""Graphing utilities for OutputSigsXL"""

import HandyXLModulesPruned as HXM
import easygui as eg
import shelve
import os
from matplotlib.backends.backend_pdf import PdfPages
import pandas as pd
import numpy as np


def dothegraphing(histdic,outdir,outfilename,listofstats,readablelabellist,
graphlog,graphlin,nameappendix=''):
    histPDF=PdfPages(os.path.join(outdir,outfilename[:-5]+nameappendix+'Histograms.pdf'))
    enrichdict={}
    enrichcount=0
    for stattouse in listofstats:
        #print histdic[stattouse], readablelabellist, len(histdic[stattouse])
        graphlinbool=False
        graphlogbool=False
        if stattouse in graphlin:
            graphlinbool=True
            if '%' in stattouse:
                if 'Area' or 'Intensity' not in stattouse:
                    HXM.graphmanyhists(histdic[stattouse],readablelabellist,stattouse,normed=True,
histtype='bar',PDF=histPDF,log=False,setbins=[0,5,10,15,20,25,30])


HXM.graphswarmpandas(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF,bargraph=T
rue,clipoutliers=True,bandwidth=0.25)

HXM.graphswarmpandas(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF,bargraph=F
alse,clipoutliers=True,bandwidth=0.25)
            else:

HXM.graphswarmpandas(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF,bargraph=F
alse,clipoutliers=True,bandwidth=0.25)

HXM.graphswarmpandas(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF,bargraph=T
rue,clipoutliers=True,bandwidth=0.25)
            HXM.graphscumhist(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF)
        if stattouse in graphlog:
            graphlogbool=True
```

```
HXM.graphswarmpandas(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF,log=True,ba
rgraph=False,clipoutliers=True,bandwidth=0.25)

HXM.graphscumhist(histdic[stattouse],readablelabellist,stattouse,PDF=histPDF,log=True)
        if '(' in stattouse:
            findunfiltered=stattouse[:stattouse.index(r'(')]
            namefilter=stattouse[stattouse.index(r'('):-1]
            banlist=[np.inf,-np.inf,np.nan]
            z=pd.DataFrame(columns=[findunfiltered,'Treatment',namefilter])
            for treatgroup in range(len(histdic[stattouse])):
                wholegroup=list(histdic[findunfiltered][treatgroup])
                for eachitem in wholegroup:
                    if eachitem not in banlist:

z=z.append({findunfiltered:eachitem,'Treatment':readablelabellist[treatgroup],namefilter:'All'},ign
ore_index=True)
                ingroup=list(histdic[stattouse][treatgroup])
                for eachitem in ingroup:
                    if eachitem not in banlist:

z=z.append({findunfiltered:eachitem,'Treatment':readablelabellist[treatgroup],namefilter:'True'},i
gnore_index=True)
            if graphlinbool==True:

enrichdict[enrichcount]=[z,stattouse,findunfiltered,namefilter,False,readablelabellist,histPDF]
                enrichcount+=1
            if graphlogbool==True:

enrichdict[enrichcount]=[z,stattouse,findunfiltered,namefilter,True,readablelabellist,histPDF]
                enrichcount+=1
    enrichkeys=enrichdict.keys()
    if len (enrichkeys)>0:
        enrichkeys.sort()
        for i in enrichkeys:

HXM.graphsubgrouppanda(enrichdict[i][0],enrichdict[i][1],enrichdict[i][2],enrichdict[i][3],enrich
dict[i][4],enrichdict[i][5],PDF=enrichdict[i][6])

            """if findunfiltered in listofstats:
                histdic[findunfiltered+'outgroup of'+stattouse]=[]
                for treatgroup in range(len(histdic[stattouse])):
                    wholegroup=list(histdic[findunfiltered][treatgroup])
                    ingroup=list(histdic[stattouse][treatgroup])
                    for ingroupval in ingroup:
                        if ingroupval in wholegroup:
```

```
                    wholegroup.remove(ingroupval)
                histdic[findunfiltered+'outgroup of'+stattouse].append(wholegroup)
            if graphlinbool==True:
                enrichlist.append((histdic[stattouse],histdic[findunfiltered+'outgroup
of'+stattouse],readablelabellist,'Relative value of '+stattouse,False))
            if graphlogbool==True:
                enrichlist.append((histdic[stattouse],histdic[findunfiltered+'outgroup
of'+stattouse],readablelabellist,'Relative value of '+stattouse,True))


    for i in range(len(enrichlist)):
        enr=enrichlist[i]
        HXM.graphsubgroupswarm(enr[0],enr[1],enr[2],enr[3],PDF=histPDF,log=enr[4])"""


    histPDF.close()
    #gc.collect()

def dothescattergraphing(histdic,outdir,outfilename,listofstats,readablelabellist):
    scattPDF=PdfPages(os.path.join(outdir,outfilename[:-5]+'Scatters.pdf'))
    scattdefs=shelve.open(os.path.join(os.curdir,'OSXLscatter'),writeback=True)
    if eg.ynbox('Do you want to use a default?'):
        whichdefault=eg.choicebox('Which default set do you want to
use?',choices=scattdefs.keys())
        scattstorun=scattdefs[whichdefault][0]
    else:
        scattstorun=[[],[],[]]
        readabletorun=[]
        finished=False
        while finished==False:
            itemforx=eg.choicebox('Which statistic do you want to have as the x-
axis',choices=listofstats)
            xlogorlin=eg.boolbox('Do you want the x-axis to be log or linear?',
choices=('log','linear'))
            subitems=[]
            dashindex=itemforx.index('-')
            for i in listofstats:
                if itemforx[:dashindex]==i[:dashindex]:
                    if itemforx[dashindex:]!=i[dashindex:]:
                        subitems.append(i)
            itemsfory=eg.multchoicebox('Which statistic(s) do you want to have on the y axis
against '+itemforx+'?',choices=subitems)
            ylogorlin=eg.multchoicebox('Which of these do you want graphed on a log scale?
Unselected items will be graphed on a linear scale.',choices=itemsfory)
            for i in range(len(itemsfory)):
                scattstorun[0].append(itemforx)
```

```
            scattstorun[1].append(itemsfory[i])
            if xlogorlin==True:
                if itemsfory[i] in ylogorlin:
                    scattstorun[2].append('xy')
                else:
                    scattstorun[2].append('x')
            else:
                if itemsfory[i] in ylogorlin:
                    scattstorun[2].append('y')
                else:
                    scattstorun[2].append(False)
            readabletorun.append(itemforx+' vs '+itemsfory[i])
        eg.textbox(msg='So far you have selected',text='\n'.join(readabletorun))
        if eg.ynbox(msg='Are there any others you would like to add?'):
            finished=False
        else:
            if eg.ynbox('Do you want to save these settings as a new default?'):
                newdefname=eg.enterbox(msg='Give this default a descriptive identifier')
                scattdefs[newdefname]=[scattstorun]
            finished=True
    scattdefs.close()

    for i in range(len(scattstorun[0])):
        #print scattstorun[0][i],scattstorun[1][i]

HXM.graphscatterpandas(histdic[scattstorun[0][i]],histdic[scattstorun[1][i]],readablelabellist,sca
ttstorun[0][i],scattstorun[1][i],PDF=scattPDF,log=scattstorun[2][i])
    scattPDF.close()

    for k in range(len(readablelabellist)):
        scattPDF=PdfPages(os.path.join(outdir,outfilename[:-5]+'-
'+readablelabellist[k]+'Scatters.pdf'))
        for i in range(len(scattstorun[0])):
            #print scattstorun[0][i],scattstorun[1][i]

HXM.graphscatterpandas([histdic[scattstorun[0][i]][k]],[histdic[scattstorun[1][i]][k]],[readablela
bellist[k]],scattstorun[0][i],scattstorun[1][i],PDF=scattPDF,log=scattstorun[2][i])

        scattPDF.close()
```

SimpleTrackingCleanup.py

```
"""Analyzes timelapse movies and makes graphs of them"""

import easygui as eg
import HandyXLModulesPruned as HXM
import Spots
import numpy
from scipy import stats
import os
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt
#from mpl_toolkits.mplot3d import Axes3D
#from matplotlib._png import read_png
#from pylab import ogrid
#import PIL
from openpyxl import Workbook, load_workbook
import seaborn as sns


def lastunderscoresuffix(str):
    undind=0
    for i in range(len(str)):
        if str[i]=='_':
            undind=i
    return str[undind:]

def imagefilenames(filename,whichfiletodraw):
    for i in os.listdir(os.path.split(filename)[0]):
            if '_Image.csv' in i:

array3=numpy.genfromtxt(os.path.join(os.path.split(filename)[0],i),delimiter=',',dtype=None)

    diffiles=[]
    diffilenames=[]
    difpaths=[]
    imageheadings=list(array3[0,:])
    #print imageheadings
    firstrow=list(array3[1,:])
    for i in range(len(imageheadings)):
        if 'FileName' in imageheadings[i]:
            diffiles.append(i)
            diffilenames.append(imageheadings[i])
        if 'Path' in imageheadings[i]:
            difpaths.append(i)
        if 'ImageNumber'==imageheadings[i]:
```

```
            filenum=i
    filenumcol=list(array3[:,filenum])
    if len(diffiles)>1:
        checkfirsts=[]
        for i in diffiles:
            if firstrow[i] not in checkfirsts:
                checkfirsts.append(firstrow[i])
        if len(checkfirsts)>1:
            if whichfiletodraw==False:
                filenameuse=eg.choicebox('Which of these is the image you want to draw
on?',choices=diffilenames)
            else:
                filenameuse=diffilenames[whichfiletodraw]
            files=list(array3[:,imageheadings.index(filenameuse)])
            path=list(array3[:,imageheadings.index('Path'+filenameuse[4:])])
        else:
            files=list(array3[:,diffiles[0]])
            path=list(array3[:,imageheadings.index('Path'+imageheadings[i][4:])])
    else:
        files=list(array3[:,diffiles[0]])
        path=list(array3[:,difpaths[0]])
    fullfilename={}
    firstfilename={}
    count=0
    imcount=1
    for i in range(1,len(path)):
        chindex=files[i].index('ch')
        if files[i][:chindex]!=files[i-1][:chindex]:
            count=0
            firstfilename[imcount]=os.path.join(path[i],files[i])
            imcount+=1
        else:
            count+=1
        fullfilename[int(filenumcol[i])]=(os.path.join(path[i],files[i]),count)

    #print fullfilename
    return fullfilename,firstfilename
```

def runtrackingcleanup(filein,minlen,maxlenmovie,maxlentrack,identifier,addonlist,fileout,makemovies,todrawsuffix,movielen,framerate,pixelsize,whichfiletodraw=False,nuclei=False):

```
    wrbook=load_workbook(fileout)
    if 'Sheet' in wrbook.get_sheet_names():
        todel=wrbook.get_sheet_by_name('Sheet')
        wrbook.remove_sheet(todel)
```

```
    if len(identifier)>31:
        try:
            isdate=int(identifier[:8])
            sheetidentifier=identifier[8:]
            if len(sheetidentifier)>31:
                sheetidentifier=sheetidentifier[:31]
        except:
            sheetidentifier=identifier[:30]
    else:
        sheetidentifier=identifier
    wrsheet=wrbook.create_sheet(title=sheetidentifier) #add the new sheet

    e=Spots.spots(filein,movielen,minlen,maxlenmovie) #initialize the spot instances
    #print e
    print filein, 'made spots'
    f=e.xyandrealt()
    #print 'xyandrealt'
    spotheadings=e[0]

    #print names

    if makemovies==True:
        #print 'movies'
        sns.reset_orig()

colorlist=['crimson','darkorange','darkorchid','darkcyan','burlywood','deeppink','lawngreen','indig
o','seashell','orangered','olive','powderblue','yellow']
        #colorlist=['r','c','m','y','k','b','g']
        names,percellnames=imagefilenames(filein,whichfiletodraw)
        #print names,percellnames
        tracks,tracksfor3d=e.trackstonow(movielen)
        """for i in tracksfor3d.keys():
            #print percellnames[i]
            plt.ioff()
            fig = plt.figure()

imagein=os.path.join(os.path.split(filein)[0],os.path.split(percellnames[i])[1][:os.path.split(percelln
ames[i])[1].index('.')]+'_0'+todrawsuffix)
            #print percellnames[i],imagein
            rawim=plt.imread(imagein)
            #print rawim
            x,y= ogrid[0:rawim.shape[0],0:rawim.shape[1]]
            ax = fig.gca(projection='3d')
            #ax.plot_surface(x,y,0.0,rstride=1,cstride=1,facecolors=rawim,color='w')
            for j in tracksfor3d[i]:
                ax.plot(j[1], j[0], j[2])
```

```
            ax.view_init(elev=50, azim=12)
            istring='%02d' %i
            plt.savefig(os.path.join(os.path.split(fileout)[0],identifier+'Cell'+istring+'_3DTracks.png'))
            plt.close() """

        for i in tracks.keys():
            #print names[i]
            subfolder=os.path.split(names[i][0])[1]
            subfoldername=subfolder[:subfolder.index('ch')]+'tracks'
            newdir=os.path.join(os.path.split(filein)[0],subfoldername)
            if not os.path.isdir(newdir):
                os.mkdir(newdir)
                #c=names[i][:names[i].index('.')]+'_'+str(i)+'_raw.bmp'
            dotindex=0
            for eachchar in range(len(os.path.split(names[i][0])[1])):
                if os.path.split(names[i][0])[1][eachchar]=='.':
                    dotindex=eachchar
            #print i,names[i],percellnames[i]
            #print names[i]

imagein=os.path.join(os.path.split(filein)[0],os.path.split(names[i][0])[1][:dotindex]+'%.2d'
%names[i][1]+todrawsuffix)
            plt.ioff()
            fig = plt.figure()
            rawim=plt.imread(imagein)
            implot=plt.imshow(rawim,cmap='gray')
            plt.axis('image')
            plt.axis('off')

plt.savefig(os.path.join(newdir,subfolder[:subfolder.index('.')]+'_'+str(names[i][1])+'_rawmovies.
png'),bbox_inches='tight',pad_inches=0)
            for eachtrack in tracks[i]:
                #plt.plot(eachtrack[0],eachtrack[1])

plt.plot(eachtrack[0],eachtrack[1],marker='o',markersize=7,fillstyle='none',color=colorlist[eacht
rack[2]])
            plt.axis('image')
            plt.axis('off')

plt.savefig(os.path.join(newdir,subfolder[:subfolder.index('.')]+'_'+str(names[i][1])+'movies.png'),
bbox_inches='tight',pad_inches=0)
            plt.close()

    col=7
    stufftowrite=[]
    meanchannels=[]
```

```
    for i in spotheadings:
        if 'MeanIntensity' in i:
            meanchannels.append(i[(i.index('MeanIntensity')+14):])
        if 'IntegratedIntensity' in i:
            channelname=i[(i.index('IntegratedIntensity')+20):]
            intoutput,medint=e.intint(channelname)
            meanoutput,medmean=e.intmean(channelname)
            tracksbycell=e.intintandintdistpercell(movielen,channelname,pixelsize)
            addonlist[7][identifier]=tracksbycell
            if identifier in addonlist[0].keys():
                addonlist[0][identifier].append(medmean+[channelname])
            else:
                addonlist[0][identifier]=[medmean+[channelname]]
            stufftowrite.append(intoutput)
            wrsheet.cell(row=1,column=col,value=i)
            col+=1
        if 'AreaShape_Area' in i:
            sizeoutput,medsize=e.size(pixelsize)
            addonlist[1][identifier]=medsize
            stufftowrite.append(sizeoutput)
            if pixelsize!=False:
                wrsheet.cell(row=1,column=col,value='Area (um^2)')
            else:
                wrsheet.cell(row=1,column=col,value='Area (pixels^2)')
            col+=1
        #print 'area'
        if 'DistanceTraveled' in i:

speedoutput,dispoutput,integdistout,medspeed,maxdisp=e.speedanddisp(pixelsize,framerate)
            #print 'first'

medspeedcell,meddispcell,maxspeedcell,maxdispcell,sumspeedx,sumspeedy,allmovementsums=e
.speedanddisppercell(movielen,pixelsize,framerate)
            #print
medspeedcell,meddispcell,maxspeedcell,maxdispcell,sumspeedx,sumspeedy,allmovementsums
            #print 'second'
            addonlist[2][identifier]=medspeed
            addonlist[3][identifier]=maxdisp

addonlist[6][identifier]=[HXM.unziprezip(medspeedcell),HXM.unziprezip(meddispcell),HXM.unz
iprezip(maxspeedcell),HXM.unziprezip(maxdispcell),HXM.unziprezip(sumspeedx),HXM.unziprez
ip(sumspeedy)]
            #print addonlist[6][identifier]
            stufftowrite.append(speedoutput)
            stufftowrite.append(dispoutput)
            stufftowrite.append(integdistout)
```

```
            if pixelsize!=False:
                if framerate!=False:
                    wrsheet.cell(row=1,column=col,value='Speed (um/second)')
                    wrsheet.cell(row=1,column=col+1,value='Displacement from 0 (um)')
                    wrsheet.cell(row=1,column=col+2,value='Integrated Distance (um)')
                else:
                    wrsheet.cell(row=1,column=col,value='Speed (um/frame)')
                    wrsheet.cell(row=1,column=col+1,value='Displacement from 0 (um)')
                    wrsheet.cell(row=1,column=col+2,value='Integrated Distance (um)')
            else:
                if framerate!=False:
                    wrsheet.cell(row=1,column=col,value='Speed (pixels/second)')
                    wrsheet.cell(row=1,column=col+1,value='Displacement from 0 (pixels)')
                    wrsheet.cell(row=1,column=col+2,value='Integrated Distance (pixels)')
                else:
                    wrsheet.cell(row=1,column=col,value='Speed (pixels/frame)')
                    wrsheet.cell(row=1,column=col+1,value='Displacement from 0 (pixels)')
                    wrsheet.cell(row=1,column=col+2,value='Integrated Distance (pixels)')
            col+=4
        #print "intensity, area, distance"

        if pixelsize!=False:
            if framerate!=False:
                #col+=5
                msdeach,msdall=e.realmsd(pixelsize,framerate,identifier,maxlentrack)
                #print msdeach
                wrsheet.cell(row=1,column=col,value='Time(sec)')
                wrsheet.cell(row=1,column=col+1,value='MSD')
                wrsheet.cell(row=1,column=col+3,value='n of MSD')
                wrsheet.cell(row=1,column=col+2,value='SEM of MSD')
                for eachrow in range(len(msdall)):
                    for eachcol in range(len(msdall[eachrow])):

wrsheet.cell(row=2+eachrow,column=col+eachcol,value=msdall[eachrow][eachcol])
                spotnum,diffco=e.diffco(pixelsize,framerate)
                addonlist[4][identifier]=[msdeach,HXM.unziprezip(msdall)]
                wrsheet.cell(row=1,column=col+4,value='Spot #')
                wrsheet.cell(row=1,column=col+5,value='Diffusion Coefficient (um^2/sec)')
                for eachrow in range(len(spotnum)):
                    wrsheet.cell(row=2+eachrow,column=col+4,value=spotnum[eachrow])
                for eachrow in range(len(diffco)):
                    wrsheet.cell(row=2+eachrow,column=col+5,value=diffco[eachrow])
                col+=7
            else:
                pass
        else:
```

```
        #col+=5
        msdeach,msdall=e.realmsd(1,1,identifier)
        wrsheet.cell(row=1,column=col,value='Time(frames)')
        wrsheet.cell(row=1,column=col+1,value='MSD')
        wrsheet.cell(row=1,column=col+3,value='n of MSD')
        wrsheet.cell(row=1,column=col+2,value='SEM of MSD')
        for eachrow in range(len(msdall)):
            for eachcol in range(len(msdall[eachrow])):
                wrsheet.cell(row=2+eachrow,column=col+eachcol,value=msdall[eachrow][eachcol])
        addonlist[4][identifier]=[msdeach,HXM.unziprezip(msdall)]
        col+=5
    #print msdeach
    #print "MSD"
    addonlist[5][identifier]=[]
    if len(meanchannels)!=0:
        firstcolor=True
        wrsheet.cell(row=1,column=col,value='Frame#')
        wrsheet.cell(row=1,column=col+1,value='# particles')
        for entry in range(len(meanchannels)):
            wrsheet.cell(row=1,column=col+2*entry+2,value='Mean per frame intensity-
'+meanchannels[entry])
            wrsheet.cell(row=1,column=+2*entry+3,value='St.dev. per frame intensity-
'+meanchannels[entry])
            frames,count,means,meandev=e.intmeanframe(meanchannels[entry])
            zipped=[meanchannels[entry]]+zip(frames,means)
            addonlist[5][identifier].append(zipped)
            if firstcolor==True:
                for eachrow in range(len(frames)):
                    wrsheet.cell(row=2+eachrow,column=col,value=frames[eachrow])
                for eachrow in range(len(count)):
                    wrsheet.cell(row=2+eachrow,column=col+1,value=count[eachrow])
                firstcolor=False
            for eachrow in range(len(means)):
                wrsheet.cell(row=2+eachrow,column=col+2*entry+2,value=means[eachrow])
                wrsheet.cell(row=2+eachrow,column=col+2*entry+3,value=meandev[eachrow])
        col+=2*len(meanchannels)+3
    if len(meanchannels)==2:
        try:
            frames,norm,normdev=e.intnormframe(meanchannels[0],meanchannels[1])
            zipped=[meanchannels[0]+'/'+meanchannels[1]]+zip(frames,norm)
            addonlist[5][identifier].append(zipped)
            wrsheet.cell(row=1,column=col,value='Normalized mean intensity-
'+meanchannels[0]+'/'+meanchannels[1])
            wrsheet.cell(row=1,column=col+1,value='Normalized st.dev. intensity-
'+meanchannels[0]+'/'+meanchannels[1])
            for eachrow in range(len(norm)):
```

```
                wrsheet.cell(row=2+eachrow,column=col,value=norm[eachrow])
                wrsheet.cell(row=2+eachrow,column=col+1,value=normdev[eachrow])
            col+=3
        except:
            frames,norm,normdev=e.intnormframe(meanchannels[1],meanchannels[0])
            zipped=[meanchannels[0]+'/'+meanchannels[1]]+zip(frames,norm)
            addonlist[5][identifier].append(zipped)
            wrsheet.cell(row=1,column=col,value='Normalized mean intensity-
'+meanchannels[1]+'/'+meanchannels[0])
            wrsheet.cell(row=1,column=col+1,value='Normalized st.dev. intensity-
'+meanchannels[1]+'/'+meanchannels[0])
            for eachrow in range(len(norm)):
                wrsheet.cell(row=2+eachrow,column=col,value=norm[eachrow])
                wrsheet.cell(row=2+eachrow,column=col+1,value=normdev[eachrow])
            col+=3

    else:
        col+=1
    #print "graph intensities"

    hist=e.lengthhist()
    wrsheet.cell(row=1,column=col+1,value='Length Histogram')
    wrsheet.cell(row=2,column=col+1,value='Length Bin')
    wrsheet.cell(row=2,column=col+2,value= '# of tracks')
    for ent in range(len(hist)):
        wrsheet.cell(row=3+ent,column=col+1,value=hist[ent][1])
        wrsheet.cell(row=3+ent,column=col+2,value=hist[ent][0])

    wrsheet.cell(row=1,column=col+4,value='Frame #')
    wrsheet.cell(row=1,column=col+5,value='# of spots')
    wrsheet.cell(row=1,column=col+6,value='% Uncorrected X')
    wrsheet.cell(row=1,column=col+7,value='% Uncorrected Y')
    for eachrow in range(len(allmovementsums)):
        for eachcol in range(len(allmovementsums[eachrow])):

wrsheet.cell(row=2+eachrow,column=col+4+eachcol,value=allmovementsums[eachrow][eachc
ol])

    writeout=HXM.deepunziprezip(stufftowrite)
    #print writeout
    writesheetrows=3 #starting row value, increments each loop below
    wrsheet.cell('A1').value='Spot Index'
    wrsheet.cell('B1').value='Frame #'
    wrsheet.cell('C1').value='X'
    wrsheet.cell('D1').value='Y'
```

```
    for i in range(len(f)):
        wrsheet.cell(row=writesheetrows-1,column=1,value=str(i+1))
        wrsheet.cell(row=writesheetrows-1,column=2,value=f[i][0])
        for eachrow in range(len(f[i][1:])):
            for eachcol in range(len(f[i][1:][eachrow])):

wrsheet.cell(row=writesheetrows+eachrow,column=2+eachcol,value=f[i][1:][eachrow][eachcol
])
        for eachrow in range(len(writeout[i][1:])):
            for eachcol in range(len(writeout[i][1:][eachrow])):

wrsheet.cell(row=writesheetrows+eachrow,column=5+eachcol,value=writeout[i][1:][eachrow]
[eachcol])
        writesheetrows+=len(f[i])

    wrbook.save(fileout)
    return addonlist


def dothestuff(defaultvals=False):
    """Asks the user for a .csv file, makes a copy of it into an excel file
    if that hasn't already been done, then creates an excel file that contains
    the x and y coordinates of all spots that pass the user's gate"""

    filefolder=eg.diropenbox(msg='Choose the parent folder that has all the subfolders
containing tracking files')
    subfolds=[]
    for i in os.listdir(filefolder):
        if os.path.isdir(os.path.join(filefolder,i)):
            subfolds.append(i)
    foldstouse=eg.multchoicebox(msg='Which of these folders contain tracking
files?',choices=subfolds)
    idents=[]
    for i in foldstouse:
        if 'out' in i:
            idents.append(i[:i.index('out')])
        elif 'Out' in i:
            idents.append(i[:i.index('Out')])
        else:
            idents.append(i)
    if len(idents)>1:
        baselineident=eg.choicebox(msg='Which of these is the baseline?', choices=idents)
    else:
        baselineident=None
    foldercsvs=[]
    nuclei=False
```

```
    for i in os.listdir(os.path.join(filefolder,foldstouse[0])):
        if '.csv' in i:
            foldercsvs.append(i)
            if 'Nuclei' in i:
                nuclei=i
    csvstouse=eg.multchoicebox(msg='Which of these files contain tracked
objects?',choices=foldercsvs)
    addondicts=[{},{},{},{},{},{},{},{},{},{}]

    if defaultvals==False:
        fulllen=int(eg.enterbox(msg='How many frames are in each original movie?'))
        gate=eg.boolbox(msg='Do you want to discard tracks shorter than a given length?',
choices=('Yes','No'))
        if gate:
            hasint=False
            setgate=eg.enterbox(msg='What is the minimum track length to consider?')
            while hasint==False: #Loop to make sure the user input is an integer
                try:
                    setgate=int(setgate)
                    hasint=True
                except:
                    setgate=eg.enterbox(msg='What is the minimum track length to consider?')
        else:
            setgate=0
        setmax=eg.ynbox(msg='Do you want to set a maximum frame number of the movie to
stop analyzing after?')
        if setmax==1:
            maxint=False
            while maxint==False:
                maxgate=eg.enterbox(msg='What is the maximum frame number to consider?')
                try:
                    maxgate=int(maxgate)
                    maxint=True
                except:
                    maxint=False
        else:
            maxgate=fulllen

        setmaxtrackpersist=eg.ynbox(msg='Do you want to change the maximum track
persistance used for MSD from the default (movielen/2)?')
        if setmaxtrackpersist==1:
            maxtrackint=False
            while maxtrackint==False:
                maxtrackgate=eg.enterbox(msg='What is the maximum track persistance to use for
MSD?')
                try:
```

```
                maxtrackgate=int(maxtrackgate)
                maxtrackint=True
            except:
                maxtrackint=False
        else:
            maxtrackgate=(fulllen/2)

        pixelsize=eg.ynbox(msg='Do you want to convert from pixels to microns?')
        if pixelsize==1:
            pixfloat=False
            while pixfloat==False:
                pixelsize=eg.enterbox(msg='Enter the size of each pixel SIDE in microns')
                try:
                    pixelsize=float(pixelsize)
                    pixfloat=True
                except:
                    pixfloat=False
        else:
            pixelsize=False
        framerate=eg.ynbox(msg='Do you want to convert from frames to seconds?')
        if framerate==1:
            framefloat=False
            while framefloat==False:
                framerate=eg.enterbox(msg='Enter the time between frames in seconds')
                try:
                    framerate=float(framerate)
                    framefloat=True
                except:
                    framefloat=False
        else:
            framerate=False

        movies=eg.ynbox(msg='Do you want to make movies of the tracks?')
        suffixi=[]
        for i in os.listdir(os.path.join(filefolder,foldstouse[0])):
            if lastunderscoresuffix(i) not in suffixi:
                if 'PRJ' not in lastunderscoresuffix(i):
                    suffixi.append(lastunderscoresuffix(i))
        if movies==True:
            drawsuffix=eg.choicebox(msg='Which of these is the suffix of the files you want to
draw the tracks on?', choices=suffixi)
        else:
            drawsuffix=False
    else:
        setgate,maxgate,maxtrackgate,movies,drawsuffix,fulllen,framerate,pixelsize=defaultvals
```

```
    fileoutname=eg.filesavebox(msg='What do you want to name the file?',filetypes=["*.xlsx"])
    if fileoutname[-5:]!='.xlsx':
        fileoutname+='.xlsx'
    writebook=Workbook() #make a new file
    writebook.save(fileoutname)

    for i in foldstouse:
        for j in csvstouse:
            #print i,j
            if nuclei!=False:
                fullnuclei=os.path.join(filefolder,i,nuclei)
            else:
                fullnuclei=False

runtrackingcleanup(os.path.join(filefolder,i,j),setgate,maxgate,maxtrackgate,idents[foldstouse.ind
ex(i)]+j[:-
4],addondicts,fileoutname,movies,drawsuffix,fulllen,framerate,pixelsize,whichfiletodraw=2,nuclei
=fullnuclei)

    if framerate!=False:
        timeunits='seconds'
    else:
        timeunits='frames'
    #Make graphs
    if pixelsize!=False:
        distunits='(um)'
        if framerate!=False:
            speedunits='(um/sec)'
        else:
            speedunits='(um/frame)'
    else:
        distunits='(pixels)'
        if framerate!=False:
            speedunits='(pixels/sec)'
        else:
            speedunits='(pixels/frame)'

    mastergraphlist=[]
    masterbubblelist=[]
    speedhistdic={}
    disphistdic={}

    for i in foldstouse:
        graphstorun={}

graphdict={'MSD':[],'IndScat':[],'AreaSpeedDist':[],'IntSpeedDist':[],'FrameVsInt':[],'Other':[]}
```

```
        bubblestorun={}
        bubblelist=[]

        for j in csvstouse:
            displist=None
            speedlist=None
            arealist=None
            chanlist=None
            comboident=idents[foldstouse.index(i)]+j[:-4]
            whichcsv=j[:-4]

            if comboident in addondicts[4].keys():
                graphstorun[(comboident,'Time ('+timeunits+')','MSD ('+distunits[1:-
1]+'^2)')]=(addondicts[4][comboident][1][0],addondicts[4][comboident][1][1],addondicts[4][co
mboident][1][2])
                graphdict['MSD'].append((whichcsv,'Time ('+timeunits+')','MSD ('+distunits[1:-
1]+'^2)'))
            if comboident in addondicts[6].keys():
                #print addondicts[6].keys()
                #print addondicts[6][comboident]
                graphstorun[(comboident,'Cell #', 'Median
Speed'+speedunits)]=addondicts[6][comboident][0]
                graphdict['IndScat'].append((whichcsv,'Cell #', 'Median Speed'+speedunits))
                graphstorun[(comboident,'Cell #', 'Median
Displacement'+distunits)]=addondicts[6][comboident][1]
                graphdict['IndScat'].append((whichcsv,'Cell #', 'Median Displacement'+distunits))
                graphstorun[(comboident,'Cell #', 'Maximum
Speed'+speedunits)]=addondicts[6][comboident][2]
                graphdict['IndScat'].append((whichcsv,'Cell #', 'Maximum Speed'+speedunits))
                graphstorun[(comboident,'Cell #', 'Maximum
Displacement'+distunits)]=addondicts[6][comboident][3]
                graphdict['IndScat'].append((whichcsv,'Cell #', 'Maximum Displacement'+distunits))
                graphstorun[(comboident,'Cell #', '% Uncorrected Movement in
X')]=addondicts[6][comboident][4]
                graphdict['IndScat'].append((whichcsv,'Cell #', '% Uncorrected Movement in X'))
                graphstorun[(comboident,'Cell #', '% Uncorrected Movement in
Y')]=addondicts[6][comboident][5]
                graphdict['IndScat'].append((whichcsv,'Cell #', '% Uncorrected Movement in Y'))
            if comboident in addondicts[5].keys():
                for color in addondicts[5][comboident]:
                    graphstorun[(comboident,'Frame #','Mean intensity('+color[0]+')')]=zip(*color[1:])
                    graphdict['FrameVsInt'].append((whichcsv,'Frame #','Mean
intensity('+color[0]+')'))
                #graphs about mean intensities at each frame
                pass
            if comboident in addondicts[3].keys():
```

```
                displist=addondicts[3][comboident]
                if j[:-4] not in disphistdic.keys():
                    disphistdic[j[:-4]]=[[idents[foldstouse.index(i)]]+displist]
                else:
                    disphistdic[j[:-4]].append([idents[foldstouse.index(i)]]+displist)
            if comboident in addondicts[2].keys():
                speedlist=addondicts[2][comboident]
                if j[:-4] not in speedhistdic.keys():
                    speedhistdic[j[:-4]]=[[idents[foldstouse.index(i)]]+speedlist]
                else:
                    speedhistdic[j[:-4]].append([idents[foldstouse.index(i)]]+speedlist)
            if comboident in addondicts[1].keys():
                arealist=addondicts[1][comboident]
            if arealist!=None:
                if speedlist!=None:
                    graphstorun[(comboident,'Median Area','Median
Speed'+speedunits,'x')]=(arealist,speedlist)
                    graphdict['AreaSpeedDist'].append((whichcsv,'Median Area','Median
Speed'+speedunits,'x'))
                if displist!=None:
                    graphstorun[(comboident,'Median Area','Maximum
Displacement'+distunits,'x')]=(arealist,displist)
                    graphdict['AreaSpeedDist'].append((whichcsv,'Median Area','Maximum
Displacement'+distunits,'x'))
            if comboident in addondicts[0].keys():
                chanidlist=[]
                chanlist=[]
                for k in range(len(addondicts[0][comboident])):
                    chanid=addondicts[0][comboident][k].pop(-1)
                    chanidlist.append(chanid)
                    chanlist.append(addondicts[0][comboident][k])
            if chanlist!=None:
                for k in range(len(chanidlist)):
                    if speedlist!=None:
                        graphstorun[(comboident,'Median Mean Intensity-'+chanidlist[k],'Median
Speed'+speedunits,'x')]=(chanlist[k],speedlist)
                        graphdict['IntSpeedDist'].append((whichcsv,'Median Mean Intensity-
'+chanidlist[k],'Median Speed'+speedunits,'x'))
                    if displist!=None:
                        graphstorun[(comboident,'Median Mean Intensity-'+chanidlist[k],'Maximum
Displacement'+distunits,'x')]=(chanlist[k],displist)
                        graphdict['IntSpeedDist'].append((whichcsv,'Median Mean Intensity-
'+chanidlist[k],'Maximum Displacement'+distunits,'x'))
                    if arealist!=None:
                        graphstorun[(comboident,'Median Area','Median Mean Intensity-
'+chanidlist[k],'xy')]=(arealist,chanlist[k])
```

```
                    graphdict['Other'].append((whichcsv,'Median Area','Median Mean Intensity-
'+chanidlist[k],'xy'))
            if len(chanlist)>=2:
                done=[]
                for k in range(len(chanlist)):
                    for l in range(len(chanlist)):
                        if k!=l:
                            m=[k,l]
                            m.sort()
                            if m not in done:
                                done.append(m)
                                graphstorun[(comboident,'Median Mean Intensity-
'+chanidlist[m[0]],'Median Mean Intensity-'+chanidlist[m[1]],'xy')]=(chanlist[m[0]],chanlist[m[1]])
                                graphdict['Other'].append((whichcsv,'Median Mean Intensity-
'+chanidlist[m[0]],'Median Mean Intensity-'+chanidlist[m[1]],'xy'))

graphlist=graphdict['MSD']+graphdict['IndScat']+graphdict['AreaSpeedDist']+graphdict['IntSpeed
Dist']+graphdict['FrameVsInt']+graphdict['Other']
        #print graphlist

graphlistformaster=graphdict['MSD']+graphdict['AreaSpeedDist']+graphdict['IntSpeedDist']+gra
phdict['FrameVsInt']+graphdict['Other']

        outpdfs=PdfPages(fileoutname[:-5]+idents[foldstouse.index(i)]+'Graphs.pdf')
        for m in graphlist:
            for r in graphstorun:
                if m[0]==r[0][len(idents[foldstouse.index(i)]):]:
                    if m[1]==r[1]:
                        if m[2]==r[2]:
                            #print m, r
                            if 'MSD' in r[2]:
                                HXM.graphmsds([graphstorun[r][0]],[graphstorun[r][1]],mantitle=r[0]+'-
MSD',xunits=timeunits,yunits=distunits,PDF=outpdfs,havestdev=[graphstorun[r][2]])
                            else:
                                #print graphstorun[r]
                                if len(r)>3:

HXM.graphscatterpandas([graphstorun[r][0]],[graphstorun[r][1]],[r[0]],r[1],r[2],mantitle=r[0]+'
- '+r[1]+' vs. '+r[2],PDF=outpdfs,log=r[3])
                                else:

HXM.graphscatterpandas([graphstorun[r][0]],[graphstorun[r][1]],[r[0]],r[1],r[2],mantitle=r[0]+'
- '+r[1]+' vs. '+r[2],PDF=outpdfs)
                            #change these to be hexbins
        for j in csvstouse:
            comboident=idents[foldstouse.index(i)]+j[:-4]
```

```
            displist=addondicts[3][comboident]
            speedlist=addondicts[2][comboident]
            HXM.graphmanyhists([speedlist],[comboident],comboident+'- Median
Speed'+speedunits,PDF=outpdfs)

            HXM.graphmanyhists([displist],[comboident],comboident+'- Maximum
Displacement'+distunits,PDF=outpdfs)

        for j in csvstouse:
            comboident=idents[foldstouse.index(i)]+j[:-4]
            if comboident in addondicts[4].keys():
                #print addondicts[4][comboident][0]

HXM.graphmsds(addondicts[4][comboident][0],addondicts[4][comboident][0],PDF=outpdfs,xu
nits=timeunits,yunits=distunits,each=True,mantitle='Per-Track MSDs'+comboident)
        for j in csvstouse:
            comboident=idents[foldstouse.index(i)]+j[:-4]
            if comboident in addondicts[7].keys():
                for m in addondicts[7][comboident].keys():
                    HXM.graphtracksinacell(addondicts[7][comboident][m],'Integrated Distance Of
Tracks'+distunits+'-'+comboident+' Cell '+str(m),PDF=outpdfs)

        mastergraphlist.append(graphstorun)
        masterbubblelist.append(bubblestorun)
        outpdfs.close()

    outpdfs=PdfPages(fileoutname[:-5]+'Graphs.pdf')
    for q in graphlistformaster:
        xlist=[]
        ylist=[]
        stdlist=[]
        for z in range(len(foldstouse)):
            for m in mastergraphlist[z]:
                if q[0]==m[0][len(idents[z]):]:
                    #print 'q[0]', q[0], m[0]
                    if q[1]==m[1]:
                        #print 'q[1]', q[1], m[0]
                        if q[2]==m[2]:
                            xlist.append(mastergraphlist[z][m][0])
                            ylist.append(mastergraphlist[z][m][1])
                            if 'MSD' in q[2]:
                                stdlist.append(mastergraphlist[z][m][2])


#Insert something to make a new sheet here, to handle all the final summary statistics
```

```
        #print xlist, ylist
        if 'MSD' in q[2]:
            HXM.graphmsds(xlist,ylist,labels=idents,mantitle=q[0]+'-
MSD',xunits=timeunits,yunits=distunits,PDF=outpdfs,havestdev=stdlist)
        else:
            if len(q)>3:
                HXM.graphscatterpandas(xlist,ylist,idents,q[1],q[2],mantitle=q[0]+' - '+q[1]+' vs.
'+q[2],PDF=outpdfs,log=q[3])
            else:
                HXM.graphscatterpandas(xlist,ylist,idents,q[1],q[2],mantitle=q[0]+' - '+q[1]+' vs.
'+q[2],PDF=outpdfs)


    for r in speedhistdic.keys():
        speedhistdicidents=[]
        speedhistvals=[]
        for q in speedhistdic[r]:
            #print q
            speedhistdicidents.append(q[0])
            speedhistvals.append(q[1:])
        if baselineident!=None:
            for k in range(len(speedhistdicidents)):
                if speedhistdicidents[k]==baselineident:
                    baseline=k
            #print speedhistdicidents
            for m in range(len(speedhistdicidents)):
                if m!=baseline:
                    ksvalue,pvalue=stats.ks_2samp(speedhistvals[baseline],speedhistvals[m])
                    speedhistdicidents[m]=speedhistdicidents[m]+'\n p='+'%.2f' %pvalue

        HXM.graphhistsaslines(speedhistvals,speedhistdicidents,r+'-Median
Speed'+speedunits,PDF=outpdfs,nbins=20)
        HXM.graphscumhist(speedhistvals,speedhistdicidents,r+'Median
Speed'+speedunits,PDF=outpdfs)


    for r in disphistdic.keys():
        disphistdicidents=[]
        disphistvals=[]
        for q in disphistdic[r]:
            disphistdicidents.append(q[0])
            disphistvals.append(q[1:])
        if baselineident!=None:
            for k in range(len(disphistdicidents)):
                if disphistdicidents[k]==baselineident:
                    baseline=k
```

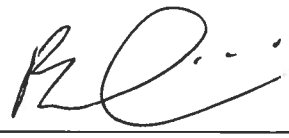
```
            for m in range(len(disphistdicidents)):
                if m!=baseline:
                    ksvalue,pvalue=stats.ks_2samp(disphistvals[baseline],disphistvals[m])
                    disphistdicidents[m]=disphistdicidents[m]+'\n p='+'%.2f' %pvalue

        HXM.graphhistsaslines(disphistvals,disphistdicidents,r+'- Maximum
Displacement'+distunits,PDF=outpdfs,nbins=20)
        HXM.graphscumhist(disphistvals,disphistdicidents,r+'- Maximum
Displacement'+distunits,PDF=outpdfs)

    outpdfs.close()


if __name__=='__main__':
    defaultvals=[4,25,12,False,'_Threecolorraw.bmp',25,4,0.065]
    #setgate,maxgate,maxtrackgate,movies,drawsuffix,fulllen,framerate,pixelsize=defaultvals
    dothestuff(defaultvals)
    """again=True
    while again==True:
        dothestuff()
        a=eg.boolbox('Do you want to do another?',choices=['Yes','No'])
        if not a:
            again=False"""
```

Publishing Agreement

It is the policy of the University to encourage the distribution of all theses, dissertations, and manuscripts. Copies of all UCSF theses, dissertations, and manuscripts will be routed to the library via the Graduate Division. The library will make all theses, dissertations, and manuscripts accessible to the public and will preserve these to the best of their abilities, in perpetuity.

Please sign the following statement:

I hereby grant permission to the Graduate Division of the University of California, San Francisco to release copies of my thesis, dissertation, or manuscript to the Campus Library to provide access and preservation, in whole or in part, in perpetuity.



Author Signature

5/25/16

Date