2009

Evolutionary Analysis of the Glutamic Acid Decarboxylase 67 Immediate Upstream Region in African Clawed Frogs

Jonathan Lomax Boyd

College of William & Mary - Arts & Sciences

Follow this and additional works at: https://scholarworks.wm.edu/etd

Part of the Developmental Biology Commons, and the Evolution Commons

Recommended Citation

https://dx.doi.org/doi:10.21220/s2-jp8a-vm44

This Thesis is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.
Evolutionary Analysis of the Glutamic Acid Decarboxylase 67 Immediate Upstream Region in African Clawed Frogs

Jonathan Lomax Boyd
Virginia Beach, Virginia

Bachelor of Science, University of Richmond, 2006

A Thesis presented to the Graduate Faculty of the College of William and Mary in Candidacy for the Degree of Master of Science

Department of Biology

The College of William and Mary
August, 2009
This Thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science

Approved by the Committee, July, 2009

Committee Chair
Chancellor Professor Margaret Saha, Biology
The College of William and Mary

Hamilton Professor Lizabeth Allison, Biology
The College of William and Mary

Marjorie S. Curtis Term Distinguished Associate Professor George Gilchrist, Biology
The College of William and Mary
COMPLIANCE PAGE

Research approved by

Institutional Animal Care and Use Committee

Institutional Biosafety Committee

Protocol number(s): 1ACUC-2007-11-13-5001

1BC-2007-02-15-4594

Date(s) of approval: 11/04/2008

02/17/2009
Changes in the expression pattern of genes during development has been shown to underlie dramatic evolutionary changes. The relationship, however, between sequence variation within non-coding regulatory regions of DNA and transcriptional output remains poorly understood in the context of molecular evolution. We report an evolutionary analysis of the immediate upstream region of xGAD67 in Xenopus laevis laevis. xGAD67 is the rate limiting enzyme primarily responsible for the acquisition of the inhibitory GABAergic neurotransmitter phenotype in the developing vertebrate central nervous system. Expression of GAD67 is part of the terminal differentiation of neural cells that use GABA (gamma-aminobutyric acid) as their neurotransmitter. The expression pattern of GAD67 during embryonic development has been shown to be highly similar among vertebrates where such studies are available, suggesting that the regulatory elements coordinating expression may be under selective pressure. Using a natural population of X. I. laevis frogs and several closely related species we attempted to detect the action of selection with a variety of test statistics on the immediate upstream region of xGAD67. Analysis of the polymorphism frequency spectrum suggests the action of purifying selection around position -200 regardless of the Xenopus outgroup used in the analysis. The action of positive selection was also detected around position -900 and -1200 for X. I. sudanensis and X. amieti, respectively. Inferences of selection based upon patterns of polymorphism may provide an alternative approach to identifying uncharacterized regulatory elements.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

## INTRODUCTION

| NEURAL DEVELOPMENT OVERVIEW | 1 |
| GLUTAMIC ACID DECARBOXYLASE EXPRESSION | 4 |
| EVOLUTION OF CIS-REGULATORY ELEMENTS | 6 |
| XENOPUS – A POLYPLOID MODEL SYSTEM FOR GENETICS | 13 |
| PROJECT OVERVIEW | 15 |

## EXPERIMENTAL AND NUMERICAL METHODS

| TISSUE COLLECTION | 17 |
| PCR, CLONING, AND SEQUENCING | 17 |
| SEQUENCE STATISTICS | 20 |
| SELECTION ANALYSIS | 20 |
| ESTIMATING $\rho$ | 27 |
| COALESCENT SIMULATION | 28 |

## RESULTS

| ALIGNMENT OF THE XGAD67 UPSTREAM REGION, INTRON 15, AND INTRON 17 | 29 |
| FIGURE 1 – MANUALLY ADJUSTED ALIGNMENT OF THE UPSTREAM REGION OF XGAD67 | 32 |
| FIGURE 2 – MANUALLY ADJUSTED ALIGNMENT OF XGAD67 INTRON 15 | 42 |
| FIGURE 3 – CLUSTALX 2.0.22 ALIGNMENT OF XGAD67 INTRON 17 | 54 |
| NUCLEOTIDE POLYMORPHISM | 59 |
| INDEL POLYMORPHISM | 59 |
| TABLE 1 – STATISTICS FOR THE UPSTREAM REGION OF XGAD67 | 61 |
| TABLE 2 – STATISTICS FOR XGAD67 INTRON 15 | 62 |
| TABLE 3 – STATISTICS FOR THE XGAD67 INTRON 17 | 63 |
| ESTIMATING HUDSON’S $\rho$ AND TEST STATISTIC DISTRIBUTIONS OF A STABLE PANMICTIC POPULATION | 64 |
| FIGURE 4 – DISTRIBUTION OF TEST STATISTICS FOR A NEUTRALY EVOLVING POPULATION BY COALESCENT SIMULATION | 65 |
| SELECTION ANALYSIS | 66 |
| FIGURE 5 – POLYMORPHISM FREQUENCY SPECTRUM FOR THE UPSTREAM REGION OF XGAD67 | 68 |
| FIGURE 6 – FAY AND WU’S H SITE-FREQUENCY SPECTRUM OF THE UPSTREAM REGION OF XGAD67 | 69 |
| TABLE 4 – COMPARISON OF NUCLEOTIDE VARIATION IN THE UPSTREAM REGION AND INTRON 15 OF XGAD67 | 71 |
| TABLE 5 – HK A TEST OF NEUTRALITY FOR THE UPSTREAM REGION OF XGAD67 | 72 |
Many people are responsible, some in small and subtle ways while others in more significant and obvious ways, for making the preparation of this manuscript feasible. On the familial side, my brothers, Joe and Justin, and parents unquestionably supported my eggheaded and financially irrational decision to return to science. Without that support none of the following pages could have been written and my future in science would have been remarkably different. On the professional side, Margaret Saha and Ralph Haygood were equal, yet disparate, sources of encouragement and never hesitant or frugal with scientific advise. I am indebted to my enablers, listed above and others, for the unpredictability of future results.
INTRODUCTION

Development of the vertebrate central nervous system is driven by conserved genetic networks from tissue specification to cell differentiation. The intricate structure of neural networks and the behavioral repertoire they generate have led to intense interest in uncovering the underlying genetics of neural development and the mechanisms by which these networks have evolved. Driven by differential gene expression, a comprehensive understanding of development will require knowledge of how genes important to neural differentiation are regulated at all organizational levels. Our focus is on the evolutionary pressures acting on the immediate upstream region of glutamic acid decarboxylase 67 (xGAD67), an enzyme that confers the GABAergic phenotype in the developing central nervous system (CNS) of vertebrates. We investigated the amount and pattern of variation in the upstream region of xGAD67 in a natural population of X. laevis laevis individuals and several closely related species. The noncoding 5' flanking region of many genes has been shown to harbor functional, experimentally verified cis-regulatory elements (Wray et al., 2003). Moreover, these 5' proximal promoter elements have been proposed to constitute a major mechanism of vertebrate brain evolution (Britten and Davidson, 1971; Haygood et al., 2007; King and Wilson, 1975).

Neural Development Overview

Neural development in vertebrates invokes a succession of cell fate restrictions, which controls the process by which undifferentiated progenitor cells become fully functioning neurons. Neuronal precursor cells can be identified in the blastula where
pluripotent cells become segregated from presumptive epidermal precursors. Induction of neural tissue is mediated primarily through vertical signaling from the dorsal meso-endoderm to the ectoderm. The inductive potential of the dorsal mesoderm to direct neurulation and axis formation was demonstrated by the well known Spemann and Mangold organizer experiments (Bouwmeester, 2001), and later shown to result from β-catenin and Vg1/VegtT coexpression and decreased bone morphogenetic protein (BMP) inhibition. Early neurulation is characterized by the formation and shaping of the neural plate. The plate then thickens during mid-neurula and bends to shape the neural groove, which will invaginate to form the neural tube. By late neurulation, localized expression of N-cadherin and N-CAM has initiated separation of the neural tube from the overlaying ectoderm. Complete closure requires, at least in mammals, Pax3, Sonic hedgehog, and openbrain expression (Milunsky et al., 1989).

Following neural tube closure, the neural tube becomes subdivided into major regions of the CNS along the anterior-posterior (A/P) axis. Expression of Hox genes defines gross anterior and posterior regions while the ectoderm differentiates into epidermal and neural fated tissue based on the specific activation or inhibition of BMP and Wnt proteins. Full inhibition of both BMP and Wnt results in the most anterior structures of the CNS, while inhibition of BMP and activation of Wnt gives rise to the spinal cord. The anterior region develops three characteristic vesicles that will become the forebrain, midbrain, and hindbrain. The spinal cord becomes morphologically distinct from the anterior CNS by an occlusion of cerebrospinal fluid constricting the neural tube at the base of the hindbrain (Desmond, 1982; Desmond and Levitan, 2002).
The dorsal-ventral (D/V) axis of the neural tube is polarized concomitantly with the A/P axis. The features of the ventral neural tube are induced by a gradient of Sonic hedgehog secretion emanating from the notochord. The most ventral region of the neural tube, where the concentration of Sonic hedgehog is highest, becomes the floor plate. The dorsal region of the neural tube is induced by secretion of TGF-β proteins (e.g. BMP4, BMP7, activin) from the overlying epidermis. The roof plate forms where the concentration of TGF-β proteins are highest and secretes secondary signaling molecules. Different layers of interneuron precursors are specified along the D/V axis by exposure to varying concentrations of Sonic hedgehog, TGF-β, and induced downstream regulators (e.g. Pax7, Nkx6.1, Nkx2.2, and Pax6) (Ericson et al., 1996).

As master regulatory genes define the A/P and D/V axis, a signaling cascade of neurogenic transcription factors, such as SoxD, Neurogenin, and NeuroD coordinate the downstream expression of terminal differentiation genes characteristic of neurons. For example, the expression of neural specific cytoskeletal structures, voltage-gated ion-channels, and biosynthetic enzymes required for production of some neurotransmitters are required for a fully differentiated neuron. In general, neurons acquire either inhibitory or excitatory neurotransmitter phenotypes whereby they communicate with neighboring cells through the release of small molecules (i.e. neurotransmitters) that cause inhibitory or excitatory post-synaptic responses.
Glutamic Acid Decarboxylase Expression

γ-Amino butyric acid (GABA) is the predominant inhibitory neurotransmitter utilized in both the vertebrate and invertebrate CNS (Watanabe et al., 2002). Acquisition of the GABAergic phenotype generally requires expression of the biosynthetic enzyme glutamic acid decarboxylase (GAD), which catalyzes the decarboxylation of glutamate to form GABA. Two predominant GAD isoforms, GAD67 and GAD65, have been identified in many vertebrates (Arata et al., 2008; Bu et al., 1992; Erlander et al., 1991; Martyniuk et al., 2007; Trabucchi et al., 2008), and are expressed during early development and adulthood, respectively (Pinal and Tobin, 1998; Popp et al., 2009).

GAD65 is primarily found in the terminal bulbs of mature GABAergic neurons and is used for neurotransmitter mediated synaptic communication in adults (Martin and Rimvall, 1993). On the other hand, GAD67 is located in both the cytoplasm of the soma and terminal bulbs of presumptive GABAergic neurons (Kaufman et al., 1991). While GAD67 produced GABA may be used in neurotransmission, its roles as a trophic factor in synaptogenesis and providing protection during oxidative stress has been demonstrated (Allain et al., 2006; Lamigeon et al., 2001; Pinal and Tobin, 1998; Popp et al., 2009; Waagepetersen et al., 1999). GAD67 is also known to promote cell proliferation, migration, differentiation, and cell death (Owens and Kriegstein, 2002). Both isoforms of GAD are differentially regulated (Popp et al., 2009; Somogyi et al., 1995); however, two variants of GAD67, GAD44 and GAD25, have been identified using Western blot analysis (Behar et al., 1993). Both GAD67 and GAD65 are also known to undergo post-translational modification (Buddhala et al., 2009; Wei and Wu, 2008).
The gross morphological features of the vertebrate CNS are remarkably similar between mammalian and non-mammalian animals. Master regulatory genes that specify and differentiate the neural tube, such as Pax6, Dlx1/2, Emx1/2, and Tbr1 are similarly expressed. Regional expression of neurogenic and proneural genes in pallial and subpallial regions are comparable despite topological differences in telencephalon development, namely evagination in mammals and eversion is teleosts (Mueller et al., 2006; Wullimann and Mueller, 2004). The location of GABAergic neurons is similarly concentrated in the spinal cord, hindbrain, midbrain, and forebrain (telencephalon) of vertebrates (Guirado et al., 1999; Katarova et al., 2000; Marin et al., 1998; Medina and Reiner, 1995; Reiner et al., 1998).

The similarity of cellular phenotypes between deeply diverged vertebrates suggests the deployment of an equally conserved regulatory network. Changes within regulatory networks, such as mutations that alter the regulation or structure of homeotic genes, reveal the delicacy of signaling systems that define major morphological structures during development. We suggest that the upstream cis-regulatory elements (CREs) directing GAD67 expression are under measurable functional constraint. Although many of the cis-regulatory elements coordinating embryonic GAD67 expression have not been identified, the most likely location of putative regulatory elements is immediately upstream of the translational start site. The conserved expression of GAD67 across distantly related vertebrate species establishes an a priori expectation that the underlying regulatory elements should be under purifying selection.
Evolution of Cis-regulatory Elements

Changes in noncoding, regulatory regions of DNA have long been suspected as a contributor to evolutionary innovation (Britten and Davidson, 1971; Jacob and Monod, 1961; King and Wilson, 1975). Molecular evolutionary analysis of gene expression centers on understanding the significance of sequence variation in trans acting factors and cis acting elements. The former is confined to the DNA sequence of transcription factor (TF) proteins that act in trans to regulate transcriptional activation. Cis acting regulatory elements are short sequences of DNA, referred to as transcription factor binding sites (TFBS), which are located adjacent to the protein-coding region being regulated. More broadly, however, there is contention whether the predominant mechanism of evolutionary innovation emerges from mutations in protein coding or non-coding, presumably regulatory, regions of DNA.

The rapid expansion of complete genome sequences for many species, including the human genome and many other closely related primates, has provided significant data to the cis versus trans debate. For example, the hypothesis by King and Wilson that cis regulatory mutations were the major mechanism behind human specific evolution has been supplemented by complete genome sequences. While they were correct that chimpanzees and humans are highly similar at the DNA level, at least 80% of the protein coding genes contain at least one amino acid change (Glazko et al., 2005). Moreover, several studies have implicated changes in protein coding regions, such as transcription factors and sensory perception and immune system genes, that have undergone rapid evolution via natural selection along the human lineage (Bustamante et al., 2005; Clark et al., 2003; Dorus et al., 2004; Nielsen et al., 2005).
In addition to the genome-level surveys, which attempt to summarize the pattern of sequence divergence between species, investigation at individual genes has provided insight into the cis versus trans debate. Changes in the expression pattern of genes is often attributed to changes in cis-regulatory elements, however, mutations within protein coding regions, often referred to as structural mutations, can also have a dramatic impact on gene expression. Synonymous substitutions in!Drosophila melanogaster!‘s alcohol dehydrogenase are known to decrease enzymes leaves by impacting mRNA stability (Carlini and Stephan, 2003).

The genetic code facilitates direct inferences to be made about the phenotypic consequences of nucleotide changes in protein coding regions, such as mutations that result in synonymous or nonsynonymous amino acid changes. More recently, the field of structural genomics has developed algorithms that make 3-dimensional structural predictions of proteins directly from DNA sequence based on similarity to other sequences for which crystallographic or nuclear magnetic resonance structural data are available (Montelione et al., 2009; Nair et al., 2009). However, the identification and phenotypic impact of variation within cis-regulatory elements is less tractable than their protein counterparts.

Furthermore, progress in understanding the “genetic toolkit” necessary for normal development has expanded the protein-centric view of molecular evolution to include non-coding regions of the genome that harbor regulatory elements. Briefly, the “genetic toolkit” is a collection of genes that determine the overall animal body plan and the number, identity, and pattern of morphological features (Carroll, 2001). Important caveats have been noted regarding the role of protein coding sequences of developmental genes in the evolution of form and function (Carroll, 2008).
Numerous TFs have been identified that have disparate master regulatory roles during development, such as the role of sonic hedgehog in ventral differentiation in the neural tube and later in limb bud formation (Carroll, 2001). Many other master regulatory genes have also been recruited for other roles later in development and therefore, regulate the development of several spatially and temporally disparate structures. A study by Stark et al. (2007) estimated that 67 Drosophila transcription factors had, on average, 124 target genes (Carroll, 2008; Stark et al., 2007). The Drosophila TF Twist has been shown to have approximately 500 target genes that mediate diverse cellular processes including cell proliferation, cell migration, and morphogenesis (Sandmann et al., 2007). More specific to development, sonic hedgehog regulates floor plate development in the early vertebrate CNS, and is later recruited to determine digit number and polarity, cerebellum development, and feather bud formation in chickens (McMahon et al., 2003). Therefore, while a newly arisen variant in a TF may provide some evolutionary innovation in one structure during development, that alteration could have dramatic pleiotropic effects in the other pathways that the gene regulates.

While animals have been diversifying for more than 500 million years, the protein components of many signaling systems remain highly similar. For example, 12 Wnt genes have been identified in vertebrates, 11 of which are also found in cnidarians (Kusserow et al., 2005). The similarity of orthologs at the sequence level, despite deep species divergence, is continued at the functional level. Functional equivalence of deeply diverged orthologs was demonstrated when Drosophila Pax-6 knockouts were rescued with mouse-Pax6 homologs to induce ectopic ommatidia (Halder et al., 1995). Similarly, the cnidarian Achaete-Scute homolog
was able to induce sensory organ formation in *Drosophila* and recognize the endogenous 
protein binding partner *Daughterless* (Grens et al., 1995).

In order for CREs to have a role in evolutionary innovation or constraint they must 
constitute a major mechanism of gene regulation and present phenotypic consequences 
when new variants arise. Genes can be regulated by a variety of mechanisms, including 
chromatin re-modeling, DNA methylation, transcriptional initiation, alternative splicing of 
RNA, mRNA stability (e.g. 5’ capping and 3’ polyadenylation), translational controls, 
covalent post-translational modifications, intracellular trafficking, and protein degradation. 
Although most genes are regulated at several of these levels, most empirical evidence 
suggests that transcriptional initiation is the most common and rate-limiting point in 
regulating gene expression (Carey et al., 2009).

In regards to phenotypic consequences of CRE variation, early screens in *Drosophila* 
for homeotic mutations demonstrated dramatic changes in abdominal identity through 
mutations in *cis*-regulatory regions of *Adh-B* (Celniker et al., 1990). Moreover, the range of 
morphological consequences due to mutations in the promoter region of *Ubx* in *Drosophila* 
were originally attributed to entirely different genes (Lewis, 1978).

Numerous cases exist where induced mutations have paralleled the phenotypic 
differences between species. For example, mutations have been induced in *Drosophila* that 
mimic the identity and number of appendages found in other insects (Carroll, 1995; Raff, 
1983), floral anatomy similar to other angiosperms in *Arabidopsis thaliana* (Lawton-Rauth, 
2000), and tail anatomy of other nematodes in *Caenorhabditis elegans* (Fitch, 1997). However, 
as pointed out by Wray et al. (2003), if mutations that alter transcriptional regulation are to 
be recognized as a major mechanism of evolution then empirical evidence for the genetic
basis of phenotypic differences in natural populations must be demonstrated (Wray et al., 2003).

Definitive cases where changes in transcriptional regulation have driven lineage-specific evolutionary change are limited, but any evolutionary transition germinates from variation within populations. Widespread use of model systems to understand development circumvents many technical obstacles, but can inadvertently project a sense of invariability. Substantial evidence suggests that natural populations from many species harbor significant variation in the spatial pattern of gene expression. In natural populations of rainbow trout, a variant of phosphoglucomutase is expressed in the liver and is associated with increased glycolysis flux in the embryo, developmental buffering, body size advantage, earlier sexual maturity, and overall accelerated prehatching growth (Allendorf et al., 1983; Allendorf et al., 1982). The expression pattern of amylase varies in the midgut of *D. melanogaster* and *D. pseudoobscura*, while the spatial pattern of *Distal-less* varies in correlation with patterns of wing coloration in the butterfly *Bicyclus anynana* (Belade et al., 2002). In the later case, the expression pattern of *Distal-less* correlated with artificial selection for patterns of wing coloration.

In addition to changes in the spatial pattern of expression, the amount and inducibility of gene expression can be altered at the transcriptional level. Polymorphisms located in *cis* have been associated with changes in the expression level of several gene in natural populations; the aforementioned amylase expression in *D. melanogaster* varies in response to starch diet (Matsuo and Yamazaki, 1984). The level of β-glucuronidase expression varies in natural populations of *Mus domesticus* and is known to differentially respond to endogenous androgen levels (Bush and Paigen, 1992). Changes in prolactin
expression have been documented in the teleost *Oreochromis niloticus* (Streelman and Kocher, 2002), and levels of the cytochrome P450 gene *Cyp6g1* in *D. melanogaster* have been linked to insertion of an upstream transposable element that confers significant insecticide resistance (Daborn et al., 2002). Furthermore, a study of 140 experimentally validated functional cis-regulatory polymorphisms for 107 genes in *Homo sapiens* were found to have a two-fold increase in the rate of transcription for >63% of surveyed genes (Rockman and Wray, 2002).

Although intraspecific changes in the spatial pattern, amount, and inducibility of gene expression have been demonstrated, to what extent is this variation heritable? Quantitative genetic surveys based on protein and mRNA expression have been conducted in a variety of species. A strong genetic component was discovered for variation in protein expression, measured electrophoretically, in *Zea mays* (Maize) (Damerval et al., 1994; De Vienne, 2001), *Pinus pinaster* (Maritime Pine) (Costa, 1999), and *Glycine max* (soybean) (Gerber, 2000). Comparison of gene expression, measured as a quantitative trait with microarrays, between natural and lab strains of *Saccharomyces cerevisiae* found that 32% of 570 differentially expressed transcripts mapped to regions within 10 kb of the expressed gene (Brem et al., 2002). A similar study in mice used the correlation between transcript abundance and quantitative trait loci (QTL) to conclude an even higher portion of cis located QTLs (Schadt et al., 2003). More broadly, however, statistical techniques have been developed to investigate the regulation of entire genetic networks and their heritability through expression QTL (eQTL) analysis (Kliebenstein, 2009).

Proponents of cis-regulatory mediated evolution have propounded that evolution can act more efficiently through changes in cis regulatory elements than protein coding regions (Stern, 2000; Wray et al., 2003). Many mutations or alleles at protein coding loci are recessive
and therefore not immediately exposed to selective pressure, particularly if heterosis, the increased fitness of heterozygotes over homozygotes, is at work. However, functional variation in promoter regions is likely to be exposed immediately to selection because of codominance; measurements of allele-specific transcript abundance indicate independent regulation (Pastinen et al., 2004; Ronald et al., 2005; Wittkopp et al., 2004). For example, temperature variation in the environment created a latitudinal cline in the proximal promoter region of lactate dehydrogenase-B (ldh-b) in populations of *Fundulus heteroclitus* (Mummichog). By comparing functional and non-functional regions of the *ldh-b* promoter, investigators conclusively identified directional selection within promoter elements (Crawford et al., 1999) that were driving population differentiation, specifically, through alteration in Sp1 binding sites (Segal et al., 1999).

Natural variation in expression of the stress-inducible chaperone *Hsp70Ba* gene was ascribed to transcriptional dysregulation induced by multiple transposable element (TE) insertions. Promoter regions polymorphic for two different insertion elements had measurable influence on *hsp70Ba* transcript abundance as revealed by RNase protection assays. Moreover, alleles polymorphic for the TE insertion had phenotypic consequences in inducible thermotolerance and female reproductive success, and therefore immediately subject to purifying selection (Lerman et al., 2003; Walser et al., 2006).

In assessing the evolutionary significance of *cis*-regulatory mutations it remains quite challenging to define casual relationships. Correlational studies between gene expression and phenotypic divergence are valuable for generating hypotheses about the process of evolutionary innovation but transitioning from pattern to process remains dubious (Hoekstra and Coyne, 2007).
Xenopus — A polyploid model system for genetics

The African clawed frogs *Xenopus* and *Silurana*, sister taxa within the subfamily Xenopodinae, family Pipidae (de Sa and Hillis, 1990), are prevalent model organisms (Cannatella and Sa, 1993) and arguably the most well characterized amphibians at the molecular level. However, the distinction between the *Xenopus* and *Silurana* genera are often not fully appreciated in studies that utilize these model systems. The tendency to treat these genera as biological equivalents underscores the continued propagation of a historical artifact despite significant molecular, cytogenetic, and morphological evidence to the contrary. Many experimental studies and major bioinformatic resources (e.g. NCBI) refer to *Xenopus tropicalis* and *Silurana tropicalis* interchangeably. Some of the notable distinctions include, for example, that species along the *Xenopus* lineage have multiples of 18 chromosomes, whereas the *Silurana* lineage includes species with multiples of 20 chromosomes (Cannatella and Trueb, 1988; Evans et al., 2004). Moreover, *Xenopus* and *Silurana* are estimated to have been evolving independently for the last 53-21 million years (Chain and Evans, 2006; Evans et al., 2004), roughly equivalent to the divergence between *Strepsirrhini* (e.g. extant lemurs) and *Haplorrhini* (e.g. lineage leading to humans) primates.

*X. laevis* have occupied a central role in early vertebrate embryology because of their large embryos, clutch sizes, and easy husbandry in the laboratory. However, more recently *Xenopus* and *Silurana* have gained prominence as an ideal vertebrate model system in evolutionary and developmental biology through extensive EST DNA sequencing, completed sequencing of the *S. tropicalis* genome, advances in transgene efficiency, and prefabricated expression microarrays. While many *Xenopus* species are known to be the
result of bifurcating speciation others are the product of reticulated lineages (i.e. allopolyplidization) (Evans et al., 2004).

The *Xenopus* clade is known to include 10 tetraploid (2N = 36), 5 octoploid (2N = 72), and 2 dodecaploid species that arose through allooloploidy (2N = 108) (Schmid and Steinlein, 1991; Tymowska, 1991). Based on the karyotypes of extant *Xenopus* frogs, their evolutionary history is suggested to have originated from an 18 chromosome diploid ancestor. However, WGDs are not unique to clawed frogs, and have occurred in teleost fish (Taylor et al., 2003), salmonid fish (Allendorf, 1984), and the ancestor of all jawed vertebrates (Dehal and Boore, 2005).

Any study that incorporates allele frequency and/or gene expression data in a polyploid model system must exercise interpretive caution. Implementing standard population genetic models based on segregation and independent assortment of alleles may be confounded by whole genome duplication (WGD) events in clawed frogs (Kobel, 1996) that have the potential to produce patterns of polysomic inheritance. Concern stems from the possible formation of multivalents or random bivalents during meiosis that would disrupt the standard diploid model of Mendalian segregation and independent assortment. In autopolyploids, polysomic inheritance is a legitimate concern because each homolog is an exact duplicated copy of its ancestor and unequal chromosome segregation during meiosis is probable. However, allopolyploid genomes arise from the hybridization of two diverged species. Therefore, a disomic pattern of inheritance, consistent with standard diploid population genetic models, may evolve rapidly or immediately following polyploidization. Laboratory generated hybrids of *Xenopus* have suggested disomic inheritance as the prevalent state (Muller, 1977). Moreover, the sequence identity of duplicated paralogs in *X. l. laevis* are
likely to be very diverged as evidenced by the divergence observed between *Xenopus* and *Silurana* over ~50 million years of evolution (Evans et al., 2004). Our analysis is therefore based on a conservative assumption that our loci displays a diploid pattern of inheritance and subsequently amenable to standard population genetic models.

Although cytogenetic evidence suggest disomic inheritance in alloployploids, the potential exists for homologs from hybridized, yet diverged, species to recombine and obscure the evolutionary history of any given gene or region of the genome. Detecting recombination can be challenging depending on the degree of divergence and extent or rate of heterogeneity of recombination (Posada, 2002). However, investigation at the RAG-1 locus in *Xenopus* and *Silurana* suggest that recombination is infrequent, at least at this locus, along both lineages (Evans et al., 2005). Similar results were found regardless of the method or parameters specified (e.g. Informative Sites Test, Recombination Detection Program, Geneconv). Therefore, inferences made from the pattern of polymorphism at most loci that assume no recombination may be considered conservative.

**Project Overview**

We investigated whether the immediate upstream region of *xGAD67* in *Xenopus laevis* has been subject to selective forces. *xGAD67* is a terminal differentiation gene with a tightly regulated spatiotemporal expression pattern during critical periods of neural development, which appears conserved across deeply diverged vertebrates. There is abundant evidence that *cis*-regulatory elements constitute a large source of variation in gene expression in natural populations. The structure and significance of variation in *cis* elements is not well understood, largely due to the difficulty in identifying and determining their
functional effect on overall transcription. However, where there is a conserved biological function, such as the pattern of expression between different species, there is an a priori expectation of constraint and marked reduction in variation. We investigated the amount and pattern of nucleotide variation in the 5' flanking region of xGAD67 in a natural population of X. l. laevis' individuals using a diverse array of tests for selection.

1 Most laboratory strains of X. laevis are, if fact, inbred lines of the species Xenopus laevis laevis from South Africa. Existing phylogenic evidence has identified several Xenopus laevis subspecies, including X. l. victorianus and X. l. sudanensis.
EXPERIMENTAL AND NUMERICAL METHODS

Tissue collection

Genomic DNA samples from 15 *X. laevis laevis* individuals were kindly provided as a gift by B. Evans (McMaster University, Hamilton, Ontario). All samples were collected on the same night in the vicinity of the Lewis Gay Dam, north of Cape Point Nature Reserve in Cape Province, South Africa. Genomic DNA was provided in 1X Tris-EDTA buffer and stored at -20°C. One female and one male from the four closely related outgroups *X. amieti*, *X. clivii*, *X. l. sudanensis*, and *X. muelleri* were obtained commercially (Xenopus Express). Genomic DNA from one individual per outgroup was extracted from ~20 mg of femoral muscle tissue with DNeasy tissue kit (Qiagen) and stored at -80°C.

PCR, cloning, and sequencing

The following primers were used to amplify three regions of the *xGAD67* locus: 5'-ACACCAGCAGCTCATT-3' / 5'-TTGGCCAGGTTAGCTCTTTC-3' (immediate upstream region including a portion of the first exon), 5'-GCCAGAGGTTGTATTTTTCACA-3' / 5'-TGATTTTGAAGCGAAAATCC-3' (intron 15, partial exon 15 and 16), 5'-GCCAAGGTTGTTATTTTCACA-3' / 5'-TGATTTTGAAGCGAAAATCC-3' (intron 17, partial exon 17 and 18). PCR was conducted in a 50 μl reaction volume using 1.25 μl of each [10μM] primer, 0.2 μl of High Fidelity Platinum Taq Polymerase (Invitrogen), 2.5 μl of 10x buffer, 1 μl of 50 mM MgCl₂, 1 μl of 10 mM dNTPs, ~225ng of genomic DNA, and brought up to volume with nuclease-free water (Promega). The amplification reaction was performed in a GeneAmp PCR System 9700.
under the following conditions: 1 cycle of 1 min at 94°C; 35 cycles of 20 sec at 94°C for denaturation, 20 sec at 55/53.5/57°C for annealing, and 1-1.5 min at 68°C for extension. PCR products were separated on an ethidium bromide agarose gel and visualized on a FluorChem HD2 analyzer (Alpha Innotech). Amplified products were directly cloned from the PCR reaction mixture into pSC-A-amp/kan vector using the StrataClone PCR Cloning Kit (StrataClone). The ligation reaction consisted of 3 µl of cloning buffer, 2 µl of PCR product, and 1 µl of vector mix (StrataClone), incubated at room temperature for 5 min and immediately transformed or stored at -20°C. Ligation reaction products were transformed into StrataClone SoloPack competent cells according to the manufacturers protocol. Plasmid DNA was recovered from the bacteria with the Wizard Plus SV Miniprep Kit (Promega) and eluted in 30 µl nuclease-free water and stored at 4°C.

DNA samples were prepared for Sanger automated capillary sequencing using BigDye Terminator v3.1 cycle sequencing kit (Applied Biosystems). The terminal chain PCR labeling was assembled with ~200 ng template DNA, 3.2 µM of M13 forward and M13 reverse primers, 2 µl of BigDye v3.1 Ready Reaction mix, and brought up to 10 µl with nuclease-free water (Promega). Labeled PCR products were combined with 2.5 µl of 125 mM EDTA, 30 µl of 100% ethanol, and incubated at room temperature for 15 min. Samples were centrifuged for 20 min at 4300 rpm at 4°C in a Mikro 200R. Samples were washed with 200 µl of 70% ethanol and centrifuged an additional 5 min at 4°C. Samples were dried ~5 min on a Savant Speed Vac Plus SC110A and resuspended with 20 µl Hi-Di formamide (Applied Biosystems). Samples were either used immediately for sequencing or stored overnight at 4°C.
DNA samples were loaded onto a 3130 Avant Genetic Analyzer equipped with a 3130 Capillary Array 50 cm column and high performance polymer Pop7 (Applied Biosystems). Internal sequencing primers were designed as necessary in order to ensure complete bi-directional sequencing. Sequence fragments were assembled with Sequencer 4.9 (Gene Codes) using default parameters. Assemblies were exported from Sequencer in FASTA format and stored for general viewing and organization on Vector NTI (Invitrogen) software. The identity of each allele was confirmed using sequence data from multiple clones (2-8) aligned with Clustal running locally in Vector NTI.

Population and outgroup sequences for introns 15 and 17 were aligned using ClustalX 2.0.11 (http://www.clustal.org/) under default settings (Larkin et al., 2007) and manually adjusted using Mesquite (http://mesquiteproject.org) in order to minimize the number of unnecessary gaps and maximize local regions of identity (Maddison and Maddison, 2009). The upstream region of xGAD67 from our X. laevis population and outgroup samples was aligned with Threaded Blockset Aligner (TBA) using default parameters (Blanchette et al., 2004) with a binary tree of our five Xenopus species described in a modified Newick format (details in Appendix) and manually adjusted in Mesquite. Multiple sequence alignments for each locus were generated in PHYLP and FASTA formats for downstream applications.
**Sequence Statistics**

Software kindly provided by R. Haygood (Duke University, Durham, NC) was used to generate our summary statistics and selection results. R.H. programs were executed in MacPython 2.5 within the Biopython framework (http://www.biopython.org; build 2.5.2); source code is provided in the Appendix.

**Selection Analysis**

The conserved expression pattern of *GAD67* across distantly related vertebrates suggests that purifying selection should be acting in noncoding regions that harbor regulatory elements. We have therefore employed several test statistics of selection in order to determine if the immediate 5' upstream region of *xGAD67* has experienced selective pressure. All of our models assume an ideal Wright-Fisher population (Fisher, 1930; Wright, 1931) and are equitably applied to our allopolyploid system for reasons mentioned earlier. Thus, populations have discrete and non-overlapping generations, with $2N$ number of genes and $N$ number of diploid individuals, constant population size where all individuals have equal fitness, no geographic or social structure, and no recombination except where indicated (Hein et al., 2005).

The signature of directional selection, either positive or negative, can be inferred as a significant deviation from a neutral expectation based on predicted patterns of polymorphism and substitution. Two test statistics, Tajima’s *D* and Fay and Wu’s *H*, attempt to detect selection by using two different estimators of the scale mutation rate (*θ*), also referred to as a population summary statistics of variation. Watterson introduced the first measure of a general class of *θ* estimators where $\theta = 4N_e\mu$; $N_e$ is the effective population
size and \( \mu \) is the neutral mutation rate of the population per generation, as an estimate of genetic diversity based on a scaled mutation rate (Watterson, 1975). For a population at equilibrium, Watterson derived the expectation for the number of segregating sites (\( S \))

\[
E(S) = a_1 \theta S,
\]

and variance

\[
V(S) = a_1 \theta S + a_2 \theta^2
\]

where

\[
a_1 = \frac{\sum_{S=1}^{S-1} \frac{1}{S}}{
\sum_{S=1}^{S-1} \frac{1}{S^2}}
\]

of \( \theta \) based on the number of segregating sites (\( S \)) in a Wright-Fisher population model of \( n \) randomly selected haploid individuals (also referred to as gametes or haplotypes). When the sample size \( n << N_e \) then \( S \) becomes the sum of \( n - 1 \) independent, geometrically distributed random variables (Watterson, 1975). An infinite sites model of mutation is assumed where each locus contains a large number of independent sites, mutations are rare events, and back mutations do not occur. In order to detect the action of selection in the \( xGAD67 \) promoter, we have obtained sequence data from a population of \( X. l. laevis \) individuals and one representative sequence from each closely related outgroup species: \( X. amieti, X. clivii, X. l. sudanensis, \) and \( X. muelleri \). We used software written by R.H. to compute Tajima's \( D \), Fu and Li's \( D \), and Fay and Wu's \( H \), and Hudson, Kreitman, Aguade (HKA) test statistics. These summary statistics with the exception of HKA utilize the difference between various estimators of \( \theta \) and are detailed below. Our analyses assume an ideal Wright-Fisher population at stationarity.
We estimated Tajima’s $D$ test statistic at the xGAD67 locus with 26 X. l. laevis samples (Tajima, 1989). Tajima proposed the $\theta_\pi$ estimator based on the average number of pairwise differences, $\pi$, expected in a random sample drawn from a population (Tajima, 1983) and determined the expected number of pairwise differences

$$E(\pi) = \theta_\pi = \frac{\sum \sum \pi_{ij}}{\binom{n}{2}},$$

and variance,

$$V(\pi) = b_1 \theta_\pi + b_2 \theta_\pi^2,$$

where

$$b_1 = \frac{n + 1}{3(n - 1)}$$
$$b_2 = \frac{2(n^2 + n + 3)}{9n(n - 1)}$$

for a neutrally evolving locus. Therefore, $\theta_\delta$ and $\theta_\pi$ are estimators of low frequency (i.e. rare variants) and intermediate frequency (i.e. common variants) alleles, respectively. The biological significance of these two different estimators of $\theta$ is their relationship in the presence of selection. Under neutrality there should be no difference between $\theta_\delta$ and $\theta_\pi$ ($0 = \theta_\pi - \theta_\delta$), however purifying selection will act to keep new variants at low frequency whereby generating an excess of extreme low frequency alleles in a population $\theta_\delta > \theta_\pi$. The presence of positive selection, conversely, drives new advantageous alleles to higher frequency and $\theta_\pi > \theta_\delta$. Tajima proposed the test statistic
which has a β-distribution under neutrality. While Tajima's $D$ can be an informative statistic for interpreting polymorphism data there are several limitations. Demographic perturbations, such as population bottlenecks, recent expansions, hitchhiking, and mutational rate heterogeneity can produce a pattern of polymorphism similar to selection. For example, population bottlenecks will greatly increase $\theta_S$ while reducing $\theta$ and result in negative $D$ values that mimic purifying selection. The difficulty in distinguishing demographic influences from selection can be partly mitigated by using several different measures of selection and investigating multiple loci. For example, a dramatic drop in population size with display an excess of low frequency alleles across all loci while an excess of low frequency allele generated by a strong selective sweep will only effect tightly link loci (i.e. genetic hitchhiking). The statistical significance of our results was determined by coalescent simulation detailed in a subsequent section (see Coalescent Simulation).

We computed Fu and Li's $D$ test statistic of selection on our population data with each outgroup species (Fu and Li, 1993). Fu and Li proposed to examine the distribution of mutations, also referred to as the site-frequency spectrum, within a genealogical framework. Similar to Tajima's $D$, Fu and Li posited that deviations from neutrality can be detected by the differences between the number of mutations expected in internal ($\eta_i$) and external ($\eta_e$) branches of a genealogy, and derived the relationship of these expectances to $\theta$ estimators. The distribution of mutations along a genealogy can be categorized as internal if they occur
in the older regions (i.e. outgroup branches) of the tree, or external if they occur in newer parts of the tree (ingroup population branches). \( \theta \) estimates of internal, \( i \), and external, \( e \), mutations are

\[
\theta_i = \frac{\eta_i}{(a_n - 1)}, \\
\theta_e = \eta_e
\]

respectively, where

\[
a_n = \sum_{k=1}^{n-1} \frac{1}{k}.
\]

In a sample of \( n \) sequences with \( i = 1, 2, \ldots, m \) nucleotide sites. The total numbers of mutations in the sample and the number of mutations in the external branches of the genealogy are

\[
\eta = \sum_i \eta_i \text{ and } \eta_e = \sum_i \eta_i,
\]

respectively; \( s_i \) are the number of segregating sites in the population-outgroup alignment minus one, \( e_i \) are the number of singleton sites between the ingroup and outgroup sequences (ingroup segregating sites ignored). The null hypothesis, under the neutral model, states that there should be no difference between \( \theta_i \) and \( \theta_e \) estimates, therefore the normalized test statistic becomes

\[
D = \frac{\eta - a_n \eta_e}{\sqrt{u_D \eta + v_D \eta^2}}
\]

where

\[
v_D = 1 + \frac{a_n^2}{a_n^2 + b_n} \left( c_n - \frac{n + 1}{n - 1} \right) \\
u_D = a_n - 1 - v_D
\]
and
\[ b_n = \sum_{k=1}^{n-1} \frac{1}{k^2}. \]

Consider a sample of sequences from a population and an outgroup organized into a genealogy, where internal branches represent mutations in “older” regions and external branches consist of “new” mutations. Under this model, the action of adaptive or purifying selection would generate an excess of mutations in the external branches of the genealogy. We conducted the test with \( n = 26 \) samples from the \( X. l. laevis \) population, which served as external branches. Related species \( X. amieti, X. clivii, X. l. sudanensis, \) and \( X. muelleri \) were treated as \( n_e \) outgroups to define internal older branches of the genealogy. The statistical significance of our results was determined by coalescent simulation detailed in a subsequent section (see Coalescent Simulation).

We also estimated Fay and Wu’s \( H \) statistic as the third independent survey of the site-frequency spectrum (Fay and Wu, 2000) at the \( xGAD67 \) locus. Analogous to the preceding tests, the \( H \) statistic is based on the difference between two estimators of \( \theta \), specifically \( \theta_\pi \) and \( \theta_H \) (Fu, 1995), where

\[ \theta_H = \sum_{i=1}^{n-1} \frac{2S_i^2}{n(n-1)}, \]

and

\[ \theta_\pi = \sum_{i=1}^{n-1} \frac{2S_i(n-i)}{n(n-1)} \]

where \( S_i \) is the number of segregating sites found in a sample size of \( n \) with a frequency \( i \) times. Noteworthy is that as \( i \) approaches \( n/2 \), alleles of intermediate frequency preferentially influence \( \theta_\pi \), while as \( i \) approaches \( n \) (i.e. numerous derived alleles at high frequency) the
effect has a greater contribution on \( \theta_H \). Originally proposed to overcome the limitations of Tajima’s \( D \) in detecting hitchhiking events, the \( \theta_H \) estimator is influenced by an excess of high-frequency alleles characteristic of strong selection. Fay and Wu’s \( H \) statistic is the difference between \( \theta_\pi \) and \( \theta_H \), divided by the variance of these estimators, and expected to be zero at a neutrally evolving locus. Negative values of \( H \) indicate that there is an excess of high frequency alleles, the signature that tightly linked low frequency alleles have been driven to high frequency by genetic hitchhiking. As stated previously, derived and ancestral sequences were defined by an ingroup population of \( X. l. laevis \) individuals and several closely related outgroups (e.g. \( X. amieti, X. clivii, X. l. sudanensis, \) and \( X. muelleri \)).

Finally, we used software by R.H. to conduct the Hudson, Kreitman, and Aguade (HKA) test on the 5’ upstream region of \( xGAD67 \) using intron 17 as a neutral proxy (Hudson et al., 1987). The HKA test of a neutral model predicts that intraspecies polymorphism and interspecies divergence are the products of the neutral mutation rate. A Fisher-Wright ideal population and infinite-sites model of mutation are assumed. For each locus \( i \) (\( i = 2 \)) the number of mutations per sample per generation is Poisson distributed. There is no recombination within loci, but free recombination is allowed to occur between loci. The ingroup and outgroup populations are assumed at equilibrium with population sizes \( 2N \) and \( 2N_f \), respectively. The population and outgroup species diverged \( T \) generations in the past from a single ancestral population. Our analysis included 24 \( X. l. laevis \) sequences representing species A, also referred to as the ingroup, while a single sequence from species B will represent the outgroup species as done in Hudson et al. (1987) example data set. Several outgroup species will be considered independently. We examined \( i = 2 \) loci, the
promoter region of $xGAD67$ and intron 15, for $n = 24$ random samples. Hudson et al. (1987) derived the expectation and variance at a neutral locus for a population of samples as

$$E(S_i^A) \approx \theta C(n_A),$$

$$\text{Var}(S_i^A) \approx E(S_i^A) + \theta^2 \sum_{j=1}^{n_A-1} \frac{1}{j^2},$$

and for an outgroup as

$$E(D_i) = \theta \left( T + \frac{(1 + f)}{2} \right),$$

$$\text{Var}(D_i) = E(D_i) + \left( \theta \cdot \frac{(1 + f)}{2} \right)^2,$$

where

$$C(n) = \sum_{j=1}^{n-1} \frac{1}{j}$$

$S_i^A$ is the number of nucleotide sites in species A (i.e. $X. laevis$) that are polymorphic at loci $i$, $D_i^A$ is the number of substitution between the ingroup population and an outgroup species, where each outgroup ($X. amieti$, $X. clivii$, $X. l. sudanensis$, and $X. muelleri$) is examined separately, $\theta_i = 4N \mu_i$, $T = T' / 2N$. The test statistic

$$\chi^2 = \sum_{i=1}^{l} \left( \frac{S_i^A - E(S_i^A)^2}{\text{Var}(S_i^A)} \right) + \sum_{i=1}^{l} \left( \frac{D_i - E(D_i)^2}{\text{Var}(D_i)} \right),$$

was proposed as a goodness-of-fit of the empirical results to the neutral model.

**Estimating $\rho$**
Assumptions about the rate of recombination can have important implications for interpreting polymorphism. The distribution of test statistics extracted from simulated data sets can be highly influenced by linkage. Likelihood methods have been developed to analyze samples with multiple linked polymorphic sites (Kuhner et al., 2000; Nielsen, 2000). We used Hudson’s maxhap program (http://home.uchicago.edu/~rhudson1) to determine two-site configurations for a sample of X. *laevis* sequences with an ancestral state specified by several closely related species. Our approach is to compare empirically determined two-site sample configurations with two-site sample distributions under a neutral model of evolution (Hudson, 2001). The objective was to estimate \( \rho \), where \( \rho = 2 r N_e \); where \( N_e \) is the effective population size and \( r \) is the rate of recombination (Hill, 1975). The estimator \( \rho \), which is a scaled population estimate of recombination, was calculated at each locus for each outgroup species (see Appendix for details). While the most conservative estimate of recombination is \( \rho = 0 \), it may cause the neutral or null hypothesis of a given test statistic not to be rejected when, in fact, selection may be present. The program exhap was used to convert a set of haplotype data into an input file with the appropriately formatted pairs data. The program maxhap then estimates \( \rho \) from our phased haplotype polymorphism data.

**Coalescent Simulation**

Our ability to distinguish variation generated by selection and variation generated by random processes was assessed by coalescent simulation of a stable, neutrally evolving panmictic population (Hudson, 2002). Source code and documentation are available online (http://home.uchicago.edu/~rhudson1) while the details of our command-line instructions executed in a UNIX shell for each locus are provided in the Appendix. For the upstream
region and intron 17, we conducted 100,000 independent, replicate coalescent simulations for a Wright-Fisher ideal population of \( N \) haploid individuals at a locus of specified length and fixed number of segregating sites (Depaulis et al., 2001; Rosenberg and Nordborg, 2002; Wall and Hudson, 2001). Standard small sample approximations of the coalescent are assumed, where the sample size \( n \ll N_c \). An infinite sites model of mutation is assumed where multiple hits and back mutations do not occur.

A random genealogical history is generated for each replicate sample at zero and estimated rates of recombination. Mutations, which are parameterized as \( 4N_e \mu \), where \( N_e \) is the effective diploid population size, and \( \mu \) is the neutral mutation rate, are Poisson distributed along a genealogy with mean equal to the product of the mutation rate and branch length. For each independent replicate simulation \( \pi \), Tajima’s \( D \), Fay and Wu’s \( H \), and Fu and Li’s \( D \) values were extracted with Hudson’s sample_stats program. The frequency distributions of these test statistics were plotted as histograms using SPSS Statistics 12.0 and compared to our empirically derived results.

**RESULTS**

*Alignment of the \( \times GAD67 \) upstream region, intron 15, and intron 17*

The amount and significance of sequence variation in noncoding, potentially regulatory, DNA was investigated in the upstream region of \( \times GAD67 \) in *Xenopus*. Approximately 1.2 kbps upstream of the \( \times GAD67 \) initiating methionine codon was cloned and sequenced for 27 alleles derived from 30 individuals of a natural population of South
African *X. laevis* clawed frogs. Three alleles were excluded from our analysis based on poor quality sequence data or incomplete contigs assemblies. These alleles will be included once the data become available. Additionally, one allele was obtained from several closely related *Xenopus* species, including *X. l. sudanensis, X. amieti, X. clivii, and X. muelleri* at the homologous *xGAD67* locus. An alignment of all 31 sequences was originally attempted with ClustalX 2.0.11 under default parameters (see Appendix). The resulting alignment presented with numerous gaps of variable length. The pattern and variable length of gaps suggested that many could be spurious, resulting from the inability of ClustalX to aligned closely related sequences with potentially large indels (insertion or deletion). A poor alignment could artificially inflate the number and length of indel polymorphisms or obscure informative segregating sites. Attempts were made to find a more parsimonious alignment that would optimize local regions of identity and minimize the number of reported gaps. Various combinations of gap opening and extension penalties were explored, in addition to manual adjustments in Mesquite, but all failed to give an alignment with an appreciable reduction in gaps.

The Clustal-class aligners, such as T-Coffee (Notredame et al., 2000), MAFFT (Katoh et al., 2002), and MUSCLE (Edgar, 2004) implement global, progressive-programming algorithms, and were originally designed to analyze protein sequences. These global aligners may be inappropriate for aligning non-coding regions where the rates of nucleotide and indel length polymorphisms are higher than in protein coding regions (Haddrill et al., 2008; Halligan et al., 2004; Liang et al., 2008). Therefore, a dynamic-programming alignment was attempted on our sample set with Threaded Blockset Aligner (TBA), originally designed for genome assembly, using default parameters (see Appendix.).
Inspection of the TBA alignment suggests that a more parsimonious alignment was obtained. The TBA output alignment was manually adjusted with Mesquite and used for downstream analysis (Fig 1.). Inspection of the alignment demonstrates, as expected, that one large indel event was obscured in the previous two alignments by the introduction of numerous unnecessary, and biologically meaningless, gaps. An alignment of 1,308 bps was obtained for the upstream region of xGAD67 with 973 informative sites used to compute the several summary statistics and our selection analysis, which regard single nucleotide polymorphisms as informative. Detailed in a following subsection, we attempted to recapture the potential significance of indels with several measures of simple indel diversity and selection, including Tajima's $D$ and Fu and Li's $D$ test statistics modified for simple indels.
Figure 1. Manually adjusted TBA alignment of the upstream region of XGAD67. Manual adjustments were performed in Mesquite in order to minimize the number of unnecessary gaps. Local regions of conservation not aligned by TBA were identified and adjusted as necessary, often eliminating gaps. The transcriptional start site is indicated by +1 and the initiating methionine is underlined. Sequences proceed from 5' to 3' and include a portion of the first exon. Conserved sites are indicated by an asterisk (*).
Introns 15 and 17 of xGAD67 were amplified and sequenced for 24 and 27 alleles, respectively, from our population sample. There were no PCR products recovered for intron 15 and 17 from several individuals. The intronic alleles were aligned with ClustalX 2.0.11 using default parameters and manually adjusted with Mesquite to minimize the number of gaps, as described earlier for the upstream region (Fig 2 and Fig 3). Evidence from mammals suggests that introns are amongst the least constrained regions in the genome and therefore provide the best empirical measure of neutrality (2005; Hellmann et al., 2003; Keightley et al., 2005). The ClustalX alignment of intron 15 was slightly different with the inclusion of each Xenopus outgroup, therefore the results for each outgroup are reported (Table 2.). Intron 15 was determined to be 1,888 bps in length, however the number of informative sites ranged from 1,190 to 1,322 bps due to indel presence/absence in the outgroup. Alignment of intron 17 was uniform across all four Xenopus species and yielded 692 bps in total length with 654 considered as informative for analyzing the site-frequency spectrum.
TGGCAGGGCTAAAATGGGCCTAGTTACACTTGTAGAGAATAAGTAAAAGCAGCATACTAATGTTAGTAGAATAATATTAA
CCTGCTTTTCAGCTCTCTTGGTTTACACTGACTGGTTACCCTGGCTACCAGGCAGTAACCAATCAGAGACTTGAGGGGGA
CCCATTAGTTATGTTCTCTGTTATGAGTACAATTACTTGCAGGACTAAAATGTAGATGATAAGTAAAAGCAGAAATTCTC
TTAAATGATGAATACTTTTTTAAAATATTACAAATGAATAGTAAAAGGCTCACAAA-
GCCACATGGGTCATATCTGTTGCTTTTGAATCTGAGCTGAATGCTGAGGATCAATTGCAAACTCACTGAACAGAAATGTC
ATTTTATTATTATTGTTTTATTATTATTATTATXATATAAATTAAATCATTAATATATTTAATAATGATTATACAATTCA
45
Figure 2. Manually adjusted alignment of xGAD67 intron 15. A ClustalX 2.0.11 alignment, performed on 24 X. laevis alleles (XLL1 – XLL24) and four closely related outgroup species (XA, X. amieti; XC, X. olivii; XLS, X. l. sudanensis; XM, X. muelleri) under defaults parameters, was modified using Mesquite in order to minimize the number of reported gaps. The regions corresponding to exons 15 and 16 are underlined and labeled. Sequences proceed from 5’ to 3’ and conserved sites are indicated by an asterisk (*).
Figure 3. ClustalX 2.0.11 alignment of *xGAD67* intron 17. Sequences from 27 *X. laevis* alleles (XLL1 – XLL27) and four closely related outgroup species (XA, *X. amieti*; XC, *X. clivii*; XLS, *X. l. sudanensis*; XM, *X. muelleri*) were aligned under defaults parameters. The regions corresponding to exons 17 and 18 are underlined and labeled. Sequences proceed from 5' to 3' and conserved sites are indicated by an asterisk (*).
Nucleotide Polymorphism

Alignment of the xGA67 upstream region in 27 X. laevis individuals revealed 46 segregating sites and 28 singletons, which are segregating sites that are only represented once in a sample (Table 1). Intron 15 contained only two segregating sites both of which were singletons (Table 2). Nineteen segregating sites were identified in intron 17 with 15 singleton sites (Table 3). Average nucleotide diversity ($\pi$) was at least three fold higher in the upstream region ($\pi = 8.84$) in comparison to intron 15 and 17, respectively, $\pi = 0.17$ and 2.10. Watterson's estimator $\Theta_W$ for the upstream region was 0.0123 per site while $\Theta_H$ ranged from 2.78 to 9.62. Intron 15 showed a marked reduction in both $\Theta$ estimators, except for $\Theta_H$ along the X. clivii lineage. $\Theta_W$ at the intron 17 locus was equally reduced compared to the upstream region, yet $\Theta_H$ demonstrated intermediate values between the upstream region and intron 15. Comparison of $\Theta_H$ across all three loci reveals that the upstream region contains comparatively more variants at higher frequency that two intronic loci, an indicator of genetic hitchhiking.

Indel Polymorphism

Traditional estimators of population diversity only account for the presence and frequency of single nucleotide variants. Indels are common structural changes within the genome and are generated by DNA replication errors, transposon activity, and unequal crossing over during meiosis. Estimators of indels, analogous to the aforementioned nucleotide estimators, have been developed to capture data missed by single nucleotide surveys. The overall pattern of indel polymorphism between the upstream region of
xGAD67 and both introns was similar to that of nucleotide polymorphism; the upstream, presumptive promoter region, is measurably more variable than the corresponding neural proxies (Table 1). The upstream region contains four segregating simple indels with an \( \pi_{\text{indel}} = 0.65 \) and Watterson’s \( \theta_{\text{indel}} = 1.04 \). Simultaneously accounting for both intron frequency and length, Balhoff and Wray’s (BW) measure of average indel length polymorphism was estimated to be \( \pi_{\text{BW}} = 5.1 \). Segregating simple indels for intron 15 locus ranged from 1-3. Average pairwise indel diversity, \( \pi_{\text{indel}} \), varied from 0.08 to 1.12 while \( \pi_{\text{BW}} = 1.53 \) across all Xenopus lineages. Intron 17 had reduced intron diversity across all measures with only one segregating simple indel\(^2\), \( \pi_{\text{indel}} = 0.0741 \) and \( \pi_{\text{BW}} = 0.78 \) which would propose that the intron itself may be under purifying selection or tightly linked to a locus, possibly within the exon, that has undergone a selective sweep(s).

\(^2\) Measurements of “simple indels” count the number of indels and do not take into account the length of any given indel.
<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nucleotide Polymorphism</strong></td>
<td></td>
</tr>
<tr>
<td>Population Sample Size</td>
<td>27</td>
</tr>
<tr>
<td>Sequence Length</td>
<td>1308</td>
</tr>
<tr>
<td>Informative Length</td>
<td>973</td>
</tr>
<tr>
<td>k</td>
<td>46</td>
</tr>
<tr>
<td>Singletons</td>
<td>28</td>
</tr>
<tr>
<td>Watterson's $\theta_W$ per site</td>
<td>0.0123</td>
</tr>
<tr>
<td>$\pi$</td>
<td>8.84 ($p &lt; 0.15$)</td>
</tr>
<tr>
<td>Tajima's $D_T$</td>
<td>-0.98 ($p &lt; 0.165$)</td>
</tr>
<tr>
<td><strong>Simple Indel Polymorphism</strong></td>
<td></td>
</tr>
<tr>
<td>$K_{\text{indels}}$</td>
<td>4</td>
</tr>
<tr>
<td>Watterson's $\theta_{\text{indel}}$</td>
<td>1.04</td>
</tr>
<tr>
<td>$\rho_{\text{indels}}$</td>
<td>0.65</td>
</tr>
<tr>
<td>Tajima's $D_{\text{indels}}$</td>
<td>-0.99</td>
</tr>
<tr>
<td>Fu and Li's $D_{\text{indels}}$</td>
<td>-1.01</td>
</tr>
<tr>
<td>$p_{\text{indels}}$</td>
<td>5.10</td>
</tr>
<tr>
<td>$p_{\text{indels}}$ per site</td>
<td>0.0039</td>
</tr>
<tr>
<td><strong>Other Statistics</strong></td>
<td></td>
</tr>
<tr>
<td>Substituted Sites (L)</td>
<td></td>
</tr>
<tr>
<td>X. l. sudanensis</td>
<td>12</td>
</tr>
<tr>
<td>X. amicti</td>
<td>1</td>
</tr>
<tr>
<td>X. clivii</td>
<td>3</td>
</tr>
<tr>
<td>X. muelleri</td>
<td>3</td>
</tr>
<tr>
<td>Hudson's $\rho$ per site</td>
<td></td>
</tr>
<tr>
<td>X. l. sudanensis</td>
<td>0.00227</td>
</tr>
<tr>
<td>X. amicti</td>
<td>0.00274</td>
</tr>
<tr>
<td>X. clivii</td>
<td>0.00264</td>
</tr>
<tr>
<td>X. muelleri</td>
<td>0.00274</td>
</tr>
<tr>
<td>$\theta_H$</td>
<td>9.62 ($p &lt; 0.47$)</td>
</tr>
<tr>
<td>Fu and Li's $D_{\text{FL}}$</td>
<td>-2.80 ($p &lt; 0.007$)</td>
</tr>
<tr>
<td>Fay and Wu's $H$</td>
<td>-0.79 ($p &lt; 0.290$)</td>
</tr>
</tbody>
</table>

Table 1. Statistics for the upstream region of *xGAD67*. The amount and pattern of variation was analyzed with various summary statistics. The statistical significance of our results was assessed by coalescent simulation of a 973 bp locus, $\rho=0$ and $k=46$, evolving neutrally in panmictic population at stationary. Substituted sites (L) is the number of differences between the respective outgroup and the X. l. *laevis* population sequences minus polymorphic sites.
<table>
<thead>
<tr>
<th>Statistic</th>
<th>(X. \text{sudanensis})</th>
<th>(X. \text{amieti})</th>
<th>(X. \text{clivii})</th>
<th>(X. \text{muelleri})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Sample Size</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Sequence Length</td>
<td>1888</td>
<td>1888</td>
<td>1888</td>
<td>1888</td>
</tr>
<tr>
<td>Informative Length</td>
<td>1322</td>
<td>1281</td>
<td>1293</td>
<td>1190</td>
</tr>
<tr>
<td>(k)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Singletons</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Watterson’s (\theta_w) per site</td>
<td>0.000405</td>
<td>0.000418</td>
<td>0.000414</td>
<td>0.000450</td>
</tr>
<tr>
<td>(\pi)</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>Tajima’s (D_T)</td>
<td>-1.51</td>
<td>-1.51</td>
<td>-1.51</td>
<td>-1.51</td>
</tr>
</tbody>
</table>
| Simple Indel Polymorphism
| \(K_{\text{in}}\) | 3 | 3 | 2 | 1 |
| Watterson’s \(\theta_{\text{in}}\) | 0.80 | 0.80 | 0.54 | 0.27 |
| \(\pi_{\text{in}}\) | 1.12 | 1.12 | 0.60 | 0.08 |
| Tajima’s \(D_{\text{in}}\) | 1.00 | 1.00 | 0.27 | -1.16 |
| Fu and Li’s \(D_{\text{in}}\) | -0.24 | -0.24 | -0.72 | -1.65 |
| \(\pi_{\text{W}}\) | 1.53 | 1.53 | 1.53 | 1.53 |
| \(\pi_{\text{W}}\) per site | 0.000808 | 0.000808 | 0.000808 | 0.000808 |
| Other Statistics
| Substituted Sites (L) | 50 | 81 | 286 | 379 |
| Hudson’s \(\rho\) per site | 0.00436 | 0.00436 | 0.00582 | 0.00291 |
| \(\theta_{\text{H}}\) per site | 0.00725 | 0.00725 | 1.92 | 0.00362 |
| Fu and Li’s \(D_{\text{FL}}\) | -2.27 | -2.27 | 0.60 | -1.65 |
| Fay and Wu’s \(H\) | 0.16 | 0.16 | -1.83 | 0.08 |

Table 2. Statistics for the \(xGAD67\) intron 15. Statistical summary of the amount and pattern of variation in intron 15, a neutral proxy for the \(xGAD67\) locus. Twenty-four alleles of \(X. \text{l. laevis}\) were aligned with four \(Xenopus\) outgroups. The inclusion of each outgroup with the population sequences changed the alignment and the number of informative sites. The results from each alignment are therefore provided for each outgroup. The paucity of segregating sites did not permit statistical inferences to be made from modeling the locus by coalescent simulation.
### Table 3. Statistics for the xGAD67 intron 17

Statistical summary of the amount and pattern of variation in intron 17, a neutral proxy for the xGAD67 locus. Twenty-seven alleles of *X. laevis* were aligned with four *Xenopus* outgroups. The statistical significance of our results was assessed by coalescent simulation of a 654 bp locus, $\rho=0$ and $k=19$, evolving neutrally in panmictic population at stationary.
Estimating Hudson’s $\rho$ and test statistic distributions of a stable panmictic population

Recombination, or more accurately the rate of recombination, is frequently used in population and evolutionary genetics to interpret polymorphism data. The scaled recombination rate $\rho = N\mu$ was estimated using Hudson’s maxhap program, as described in the Methods section and Appendix, for each outgroup species. Estimates of $\rho$ across all four outgroup species at all three loci suggest that recombination is not pervasive (Tables 1-3).

Coalescence theory has become indispensable for interpreting polymorphism data and the likelihood of demographic or selective forces in shaping patterns of variation (Rosenberg and Nordborg, 2002). Coalescent simulations were conducted with Hudson’s ms program to generate 100,000 replicate samples of a stable, neutrally evolving panmictic population with no recombination. The distribution of $\pi$, Tajima’s $D$, Fu and Li’s $D$, Fay and Wu’s $H$ test statistics were extracted with Hudson’s sample_stats program in order to determine the likelihood of our results given a neutral model (Fig. 4). The sampling distributions of the all test statistics were not appreciably altered even at the highest estimated rate of recombination, $\rho = 0.00582$, therefore, null distributions with and without estimated rates of recombination are equally conservative.
Figure 4. Distribution of test statistics for a neutrally evolving population modeled by coalescent simulation. A genealogical history with a specified, Poisson distributed number of mutations was generated for a locus equal to the size of the upstream region of xGAD67. Hudson's ms generated 100,000 replicate sample populations and the values of four summary statistics (A-D) were extracted for each replicate sample with Hudson's sample_stats program. The observed statistic for the upstream region was calculated for each distribution to be (A) 8.84, (B) -0.98, (C) 2.77, and (D) 2.80. Summary statistics for C-D are averaged across all four Xenopus outgroups.
Selection Analysis

The significance of sequence variation at the xGAD67 locus was analyzed to determine if the immediate upstream region has been subject to selection. Four test statistics were employed on a X. I. laevis population sample of 27 alleles and four closely related species in order to detect the presence of selection. Tajima’s $D_T$ and Fay and Wu’s $H$ utilize different $\theta$ estimators, while Fu and Li’s $D_{FL}$ assesses the genealogical distribution of polymorphisms against a neutral expectation. In contrast, the HKA test relies on a neutral proxy. We chose to use intron 15 as the neutral proxy for the HKA test based on the observed number of within species differences and between species substitutions. Interestingly, no substituted sites were identified in the outgroup for intron 17. These statistics, except for the HKA test, were developed to detect selection from the frequency spectrum of polymorphic sites in a population sample and therefore indels are excluded from the analysis. Subsequently, $D_T$ and $D_{FL}$ tests were modified and reported as $D_{INDEL}$ and $D_{INDEL}$ in order to accommodate the possibility that indels could be under selection. In order to investigate the likely variation contained in summary statistics of nucleotide variation, we determined the polymorphism frequency spectrum for $D_T$ and $D_{FL}$ was also determined along the axis of transcription in a sliding window frame of 101 bases with a 10 base slide. Essentially, each summary statistic is calculated for a small region (e.g. 101 bps.). The 101 bp window or region of interest is then shifted 10 bps along the sequence and the summary statistics are again calculated.
Fu and Li’s $D$ test statistic suggests that the promoter region of $\times$GAD67 contains an excess of rare alleles that is significantly greater than the level expected in a neutrally evolving population (Table 1). The distribution of extreme low frequency alleles, summarized as negative Fu an Li values, is particularly concentrated in the most proximal region immediately upstream of the $\times$GAD67 coding region, approximately 175 bps upstream from the transcriptional start site (Fig 5). In this region, the Fu and Li statistic takes on the most negative values for the upstream region, indicating an abundance of rare alleles (i.e. singletons) immediately upstream of the transcriptional start site. Using Fu and Li’s nomenclature, there is an abundance of new mutations in the external branches of the *Xenopus* genealogy (i.e. *X. l. laevis* clade) compared to the older internal branches (i.e. *Xenopus* outgroups). An excess of low frequency alleles can be generated by rapid population expansion/contraction, positive selection, or purifying selection. Positive Fay and Wu $H$ values found in three out of four outgroup species for all three loci suggest that positive selection is unlikely as the signature of genetic hitchhiking$^3$ would preferentially increase $\theta_H$ to produce negative $H$ values (Table 1). Our inference is strengthened by the very similar, and positive, distribution of $H$ values immediately upstream of the transcriptional start for all four *Xenopus* outgroups (Figure 6). Moreover, dramatic perturbations in population size would also generate negative Tajima’s $D$ values, which were shown to be statistically consistent with a stable panmictic population in the upstream region. Therefore, a model of purifying selection in the region immediately upstream of $\times$GAD67 is consistent with our results.

$^3$ Genetic hitchhiking occurs with a locus has recently undergone a strong selective sweep and risen to high frequency within the population because of positive, favorable selection for a new variant. Nearby, neutral variation, also rises or hitchhikes to high frequency because of linkage.
Figure 5. Polymorphism frequency spectrum for the upstream region of $xGAD67$. Tajima’s $D$ ($D_T$) and Fu and Li’s $D$ ($D_{FL}$) test statistics are plotted along the axis of transcription from 5’ to 3’ in 101 bp sliding window frame with a slide of 10 bps. The red and blue horizontal lines indicate the average $D_T$ and $D_{FL}$ values, respectively. The +1 indicates the transcriptional start site.
Figure 6. Fay and Wu's $H$ site-frequency spectrum of the upstream region of \textit{xGAD67}. Fay and Wu's $H$ statistic was calculated per site in 101 bp sliding window frame (10 bp slide) and plotted against the axis of transcription from 5' to 3'. The black horizontal line indicates the average $H$ value per site for all four \textit{Xenopus} species. The +1 indicates the transcriptional start site.
Incongruously, intron 17 also displays a strong signature of purifying selection (Table 3). Fu and Li’s $D$ values are substantially higher than the expectation of a similarly sized locus evolving neutrally in a stable panmictic population ($p < 0.0005$). Tajima’s $D$ are in concordance with directional, potentially purifying, selection ($p < 0.006$), but do not discount the possibility of population expansions as a confounding scenario. While not statistically significant, Fay and Wu’s $H$ of 1.9 suggest that positive selection is unlikely.

Statistical inference of population parameters for intron 15 alone is problematic due to the deficiency of segregating sites within the $X. l. laevis$ population. Modeling the distribution of any given test statistic by coalescence assumes that number of mutations Poisson distributed along a genealogy can be approximated by the number of observed mutations at a locus. However, when the number of polymorphic sites is very low then mutations cannot be randomly distributed and any statistical inference based on our test statistics are inconclusive (Depaulis et al., 2001). On a qualitative level, there is a positive correlation between the number of substitutions ($I$) and the evolutionary distance of $Xenopus$ outgroups (Table 2). The linear accumulation of substitutions in each of the outgroups combined with a dearth of polymorphic sites may suggest balancing selection (Fu and Li, 1993), however, the paucity of informative sites makes such a conclusion speculative.

Results from the HKA test also support a substantial deviation from a neutral model of evolution (Table 5). Intron 15 was chosen as a neutral proxy to investigate the upstream region of $xGAD67$ because it contained both within species polymorphism and between species divergence. The ratio of within species to between species variation was considerably greater for the upstream region (3.83) compared with intron 15 (0.04). Moreover, the degree of variation within the $X. l. laevis$ population was 31.25x higher for the promoter than intron
15 while the level of divergence was only 0.32 (Table 4). These relationships of within species variation to between species divergence for the upstream region and intron 15 are consistent across all the outgroup species.

<table>
<thead>
<tr>
<th>Species</th>
<th>Upstream region</th>
<th>Intron 15</th>
<th>Ratio (upstream/intron 15)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X. t. sudanensis</td>
<td>0.047</td>
<td>0.002</td>
<td>31.250</td>
</tr>
<tr>
<td></td>
<td>0.012</td>
<td>0.038</td>
<td>0.326</td>
</tr>
<tr>
<td></td>
<td>3.833</td>
<td>0.040</td>
<td></td>
</tr>
<tr>
<td>X. amieti</td>
<td>0.047</td>
<td>0.002</td>
<td>31.250</td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td>0.061</td>
<td>0.017</td>
</tr>
<tr>
<td></td>
<td>46.000</td>
<td>0.025</td>
<td></td>
</tr>
<tr>
<td>X. clivii</td>
<td>0.047</td>
<td>0.002</td>
<td>31.250</td>
</tr>
<tr>
<td></td>
<td>0.003</td>
<td>0.216</td>
<td>0.014</td>
</tr>
<tr>
<td></td>
<td>15.333</td>
<td>0.007</td>
<td></td>
</tr>
<tr>
<td>X. muelleri</td>
<td>0.047</td>
<td>0.002</td>
<td>31.250</td>
</tr>
<tr>
<td></td>
<td>0.003</td>
<td>0.287</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>15.333</td>
<td>0.005</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Comparison of nucleotide variation in the upstream region and intron 15 of xGAD67. The amount of polymorphism and divergence in the 973 bp upstream region and 1,322 bp intron 15 is reported within a population of 24 X. laevis alleles and between four closely related Xenopus species.
Table 5. HKA test of neutrality for the upstream region of xGAD67. Intron 15 was chosen as a neutral proxy for the nearby upstream region of xGAD67. The HKA test of a neutral model assumes that the level of polymorphism and divergence are products of the neutral mutation rate and therefore equivalent. The statistical significance of the amount of variation in 24 X l. laevis alleles compared with four closely related Xenopus species is reported.

<table>
<thead>
<tr>
<th>Region</th>
<th>Polymorphic Sites</th>
<th>X. l. sudanensis</th>
<th>X. amieti</th>
<th>X. clivii</th>
<th>X. muelleri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Promoter</td>
<td>46</td>
<td>12</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Intron 15</td>
<td>2</td>
<td>50</td>
<td>81</td>
<td>286</td>
<td>379</td>
</tr>
</tbody>
</table>

(p < 3.49E-06) (p < 1.52E-11) (p < 1.88E-38) (p < 0)

DISCUSSION AND FUTURE DIRECTIONS

We have provided evidence that the proximal promoter region, immediately upstream of xGAD67, may be subject to purifying selection. More distal elements, approximately 800 bps upstream, may be under lineage specific selection. Changes within noncoding regulatory elements can have significant impacts on phenotypic diversity. Noncoding elements regulating the expression of developmental genes have been shown to drive major morphological transitions. However, whether the logic of regulatory evolution for large scale morphological features extends to the cellular physiological level remains unclear. Moreover, the intractability of identifying cis-regulatory elements has slowed progress in understanding regulatory evolution in comparison to protein coding regions. Significant progress has been made in elucidating the relationships between transcript abundance and patterns of noncoding variation in many model systems (Johnson et al., 2009;
Kim et al., 2009; Lawniczak et al., 2008; Tung et al., 2009; Zhang and Borevitz, 2009). Detailed studies in sea urchins have been able to superimpose selection analysis with maps of experimentally identified cis-regulatory elements at loci of several terminal differentiation genes (Balhoff and Wray, 2005; Walters et al., 2008). However, such analyses are difficult with most vertebrates and not feasible in higher primates. The relevance of cis-regulatory elements identified in cell assays to biological processes in the organism remains questionable.

*Xenopus* has continued to be a major model system for early vertebrate embryology with advances in genome sequencing and transgene technology (Ogino and Ochi, 2009). The ability to assay for cis-regulatory driven gene expression *in vivo* for a vertebrate system is significant. Progress in understanding the speciation genetics of African clawed frogs (Evans, 2008) coupled with experimental tractability make *Xenopus* a powerful model for investigating regulatory evolution.

We have provided an evolutionary genetic analysis of the upstream region of \(xGAD67\) using a South African population of *X. l. laevis* individuals and four closely related *Xenopus* species: *X. amieti*, *X. clivii*, *X. l. sudanensis*, and *X. muelleri*. We have provided evidence that \(~1.3\) kbps upstream of the translational start site may have been subject to purifying selection, and therefore, contain functional regulatory elements. The pattern of extremely low frequency alleles suggest that the intensity of purifying selection is particularly concentrated in the \(-200\) to \(+150\) region were putative cis-regulatory elements may be recognizing retina specific transcription factors necessary for GABAergic differentiation. This region is capable of driving retina-specific \(xGAD67\) expression when cloned into a green fluorescent protein (GFP) reporter and monitored in a developing transgenic *X. l.*
laevis embryo (data unpublished). The pattern of endogenous retina-specific xGAD67 expression is widely conserved in vertebrates and the action of purifying selection is not unexpected. However, similar studies in sea urchins have shown that widespread sequence conservation is not a necessary prerequisite driving conserved patterns of expression (Oda-Ishii et al., 2005). The features of cis-regulatory regions, such as modularity, are theoretically capable of emerging from nonadaptive processes (Lynch, 2007a; Lynch, 2007b) and redundancy in regulatory elements has been shown for promoters with shadow enhancers4 (Wray and Babbitt, 2008). However, the suggestion that gene expression is maintained by TF binding site turnover, where sites are lost and gained concomitantly, has found little empirical support (Doniger and Fay, 2007).

Our study suggests that if regulatory elements are present in the region 1.3 kbps upstream of xGAD67 then developmental expression of this locus is intolerant of mid to high frequency variation. Alternatively, the excess of rarest alleles may be due to sequencing or PCR amplification errors, however we attempted to mitigate those sources of error by using multiple clones and bi-directional sequencing to determine the sequence of each allele. Moreover, the abundance of low frequency alleles was not observed at other intronic loci. Another possibility is that our primers may have been allele specific, or allele degenerate whereby allopolyploid paralogs may have been included in the analysis. Both of these situations seem unlikely based on our initial electrophoretic results of PCR products where only 1 or 2 bands were observed (data not shown). Also, there exist the possibility where xGAD67 locus may be subject to an unusual pattern of inheritance for allopolyploids where multivalents form during meiosis (Sammut et al., 2002).

4 Cis-regulatory elements that recognize a similar complement of transcription factors acting on the main regulatory elements.
Based on the polymorphism frequency spectrum, we propose that the region -200 to +150 of \( xGAD67 \) likely contains \( \text{cis} \)-regulatory elements. In order to determine if this region or others nearby contain regulatory elements, a variety of transgene constructs will need to be generated. The significance of the polymorphism frequency spectrum of \( xGAD67 \) requires an \textit{in vivo} assay in order functionally characterize the regulatory elements. Transgenic \( X.\ laevis \) embryos can be generated by sperm nuclear injection of upstream regions fused to a GFP reporter (Smith et al., 2006). A variety of transgenic methods have been developed for high transgene efficiency (e.g. REMI, \( I\HindIII \) meganuclease, or transposon methods) (Ogino and Ochi, 2009). Constructs should be generated for both the longest and shortest allele in the population, 1,252 and 982 bps in length, respectively. Insertional elements on the order of 200 bps may have arisen from transposon activity and could contain entire regulatory elements with enhancer or silencer activity. The -200 to +150 bp region will need to be assayed with a basal promoter-driven GFP reporter plasmid that contains \( \text{cis} \) elements necessary for recruiting the basal transcriptional machinery needed for activity in all cells.

Experimental identification of \( \text{cis} \)-regulatory elements can be further resolved using a variety of techniques to investigate specific TF binding targets. For example, \( x\text{Distal-less-4} \) (\( x\text{Dl-4} \)) and \( xGAD67 \) expression has been shown to overlap in the forebrain (Brox et al., 2003). \( x\text{Dl-4} \) binding sites could be identified \textit{in silico} and independently, or coordinately, disrupted through site directed mutagenesis. High resolution regulatory maps could be superimposed on the site-frequency spectrum to further resolve which sites are evolving under selection. Lastly, the endogenous pattern of \( xGAD67 \) expression should also be assessed in each of the outgroups. While distant vertebrates are known to express \( xGAD67 \),
similarly during development, in situ hybridization should be performed to confirm that lineage specific changes in xGAD67 expression have not occurred.

We find the amount of polymorphic and substituted sites within introns 15 and 17 an unexpected result for a region presumed to be neutral. Intron 15 is an intermediate sized intron at the xGAD67 locus with an expected pattern of substitutions (Table 2) sustained along lineages of increasing phylogenetic distance (Evans et al., 2004). Curiously, only two segregating sites, both singletons, were identified among 24 alleles. A possibility exists that our original cloning attempts were allele specific whereby a considerable amount of the variation was excluded from our analysis. Amplifying intron 15 with an alternative set of primers would likely resolve whether our initial set was indeed allele specific. However, inspection of the alignment at two indel sites, one and six bps in length, respectively, indicate the presence of at least two different alleles of intermediate frequency isolated by PCR from our population sample.

Conversely, intron 17 contains 19 segregating sites but appears to accumulate no differences along any of our Xenopus lineages. Our study relies on one sequence from each outgroup, similar to Balhoff and Wray (2005) and Walters et al (2008), and may not be representative. The inverse relationship in variation between introns 15 and 17 is perplexing. Attempts to generalize the relationship between intron length and divergence have produced contradictory results, possibly attributed to lineage specific genome evolution (Gazave et al., 2007; Haddrill et al., 2005).

In summary, the upstream region of xGAD67 may contain cis-regulatory elements that are under selective constraint compared with both the neutral expectation and a nearby neutral region. Since no consensus sequence(s) exists for identifying promoter elements, we
propose that the polymorphism frequency spectrum can provide a detailed map of prospective regulatory elements for genes with conserved expression patterns and functions.

The sea urchin literature has several detailed functional descriptions of evolutionary processes acting within promoter regions (Balhoff and Wray, 2005; Walters et al., 2008), while few cases exist in vertebrates. We hope that our analysis will complement the functional utility of *Xenopus* as a vertebrate model system for understanding gene regulation as a major component of the evolutionary process.
Source code for summary statistics and sliding window analysis

The summary statistics and sliding window analyses were carried out with software written and provided by R.H.. The source code for all of our analyses are provided below. Separate module scripts have been prepared that make performing individual analysis easier; an example is provided later in the Appendix. The following software is known to run with Python 2.5 and Biopython 1.42. The HKA test was implemented using chi-squared distributions, which are implemented in C with a Python wrapper. That code is not provided below.

```python
import Bio.Alphabet # "
import Bio.Alphabet.IUPAC # "
import copy # "
import math # "
import operator # "
import re # "
import sys # "
import chiSquared # Custom.

# MyAlignment is a facade to hide the ugliness of Biopython’s Alignment.
    def _init_(self):
    def addSequence(self, sequence):
    def getNucleotides(self, site):
        return list(Bio.Align.Generic.Alignment.get_column(self, site))
    def getSequence(self, taxon):
    def getSiteCount(self):
    def getTaxonCount(self):
        return len(Bio.Align.Generic.Alignment.get_all_seqs(self))
    def concatenate(self, other):
        result = MyAlignment()
```
for taxon in range(self.getTaxonCount()):
    result.addSequence(self.getSequence(taxon)+other.getSequence(taxon))
return result

@classmethod
def concatenation(dummy, first, second):
    result = MyAlignment()
    for taxon in range(first.getTaxonCount()):
        result.addSequence(first.getSequence(taxon)+second.getSequence(taxon))
    return result

# FASTA_FileToMyAlignment is clumsy. A record parser and an iterator would be graceful.
def FASTA_FileToMyAlignment( ilename):
    handle = open( ilename, 'r')
    entries = handle.read().split(">")
    handle. close()
    alignment = MyAlignment()
    for entry in entries[1:]:
        alignment.addSequence(entry[entry.index("\n")+1:].replace("\n", ""))
    return alignment

# PHYLIP_FileToMyAlignment is clumsy. A record parser and an iterator would be graceful.
def PHYLIP_FileToMyAlignment( ilename):
    handle = open( ilename, 'r')
    lines = handle.readlines()
    handle.close()
    alignment = MyAlignment()
    for line in lines[1:]:
        matchData = re.search("A \S+\s+\(\S+\)\$", line)
        if matchData != None:
            alignment.addSequence(matchData.group(1))
        else:
            alignment.addSequence(line[10:-2])
    return alignment

# A SND is a single nucleotide difference, which might be a polymorphism within a population or a
# substitution between populations or both.
# SND objects usually aren't directly useful to users; they're meant to be created within Sample objects.
class Snd:
    def __init__(self, site, nucleotides, populations, sndClasses):
        self.site = site
        self.sndClasses = sndClasses
        self.nucleotides = {}
        self.nucleotideFrequencies = {}
        for population in populations.keys():
            self.nucleotides[population] = [nucleotides[taxon] for taxon in populations[population]]
            self.nucleotideFrequencies[population] = {}
        for nucleotide in set(self.nucleotides[population]):
            self.nucleotideFrequencies[population][nucleotide] = self.nucleotides[population].count(nucleotide)
    def __str__(self):
        return "(%s, %s, %s, %s) % (str(self.site), str(self.nucleotides), str(self.nucleotideFrequencies), str(self.sndClasses))"

# A simple indel has two alleles, one nothing but gaps and the other free of gaps. Two or more such
# features immediately adjacent to each other are a complex indel.
# SimpleIndel objects usually aren’t directly useful to users; they’re meant to be created within Sample objects.

Class SimpleIndel:

def __init__(self, site, size, alleles, populations, indelClasses):
    self.site = site
    self.size = size
    self.indelClasses = indelClasses
    self.alleles = {}
    self.alleleFrequencies = {
        for population in populations.keys():
            self.alleles[population] = [alleles[taxon] for taxon in populations[population]]
            self.alleleFrequencies[population] = {
                for allele in set(self.alleles[population]):
                    self.alleleFrequencies[population][allele] =
                        self.alleles[population].count(allele)
        def __str__(self):
            return "(%s, %s, %s, %s, %s)" % (str(self.site), str(self.size), str(self.alleles), str(self.alleleFrequencies), str(self.indelClasses))

# A complex indel is any indel that isn’t simple.
# ComplexIndel objects usually aren’t directly useful to users; they’re meant to be created within Sample objects.

Class ComplexIndel:

def __init__(self, site, size, alleles, populations, indelClasses):
    self.site = site
    self.size = size
    self.indelClasses = indelClasses
    self.alleles = {}
    self.alleleFrequencies = {
        for population in populations.keys():
            self.alleles[population] = [alleles[taxon] for taxon in populations[population]]
            self.alleleFrequencies[population] = {
                for allele in set(self.alleles[population]):
                    self.alleleFrequencies[population][allele] =
                        self.alleles[population].count(allele)
        def __str__(self):
            return "(%s, %s, %s, %s, %s)" % (str(self.site), str(self.size), str(self.alleles), str(self.alleleFrequencies), str(self.indelClasses))

# Sites containing ambiguous nucleotides are ignored apart from optional diplotype expansion.
ambiguousNucleotideSet = set((
    'R', # A, G
    'Y', # C, T
    'W', # A, T
    'S', # C, G
    'M', # A, C
    'K', # G, T
    'B', # C, G, T
    'D', # A, G, T
    'H', # A, C, T
    'V', # A, C, G
    'N' # A, C, G, T
))

# Synonymy in coding sequences is assessed using the standard genetic code.
aminoAcids = {

<table>
<thead>
<tr>
<th>Codon</th>
<th>Amino Acid</th>
<th>Codon</th>
<th>Amino Acid</th>
<th>Codon</th>
<th>Amino Acid</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTT</td>
<td>F</td>
<td>TTC</td>
<td>F</td>
<td>TTA</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCA</td>
<td>S</td>
<td>TCT</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCC</td>
<td>S</td>
<td>TCG</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAT</td>
<td>Y</td>
<td>TAC</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAA</td>
<td></td>
<td>TAG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAT</td>
<td>Y</td>
<td>TAC</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAA</td>
<td></td>
<td>TAG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCT</td>
<td>L</td>
<td>CTT</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTC</td>
<td>L</td>
<td>CTA</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTG</td>
<td>L</td>
<td>CCT</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCC</td>
<td>P</td>
<td>CCA</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCG</td>
<td>P</td>
<td>CAT</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CAC</td>
<td>H</td>
<td>CAA</td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CAG</td>
<td>Q</td>
<td>CGT</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGC</td>
<td>R</td>
<td>CGA</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CGG</td>
<td>R</td>
<td>ATT</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATC</td>
<td>T</td>
<td>ATA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATG</td>
<td>M</td>
<td>ACT</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACC</td>
<td>T</td>
<td>ACA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACG</td>
<td>T</td>
<td>AAT</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAC</td>
<td>N</td>
<td>AAA</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAG</td>
<td>K</td>
<td>AGT</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AGC</td>
<td>S</td>
<td>AGA</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AGG</td>
<td>R</td>
<td>GTT</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GTC</td>
<td>V</td>
<td>GTA</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GTG</td>
<td>V</td>
<td>GCT</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GCC</td>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def Translate(sequence):
    translation = ""
    site = 0
    while site < len(sequence)-3:
        if sequence[site:site+3] or sequence[site] in ambiguousNucleotideSet or sequence[site+1] in ambiguousNucleotideSet or sequence[site+2] in ambiguousNucleotideSet:
            translation += "-"
        else:
            translation += aminoAcids[sequence[site:site+3]]
            site += 3
    return translation

# Sample objects usually aren't directly useful to users; they're meant to be created within Statistics objects.

Class Sample:

def __init__(self, alignment, populations = {}, siteClasses = {}, options = {}):
    self.populations = copy.deepcopy(populations)
    self.siteClasses = copy.deepcopy(siteClasses)
    self.options = copy.deepcopy(options)
    if populations == {}:
        self.populations['_all'] = range(alignment.getTaxonCount())
    else:
        self.populations['_all'] = list(set(reduce(operator.concat, populations.values())))
    if siteClasses == {}:
        self.siteClasses['_all'] = range(alignment.getSiteCount())
    else:
        self.siteClasses['_all'] = list(set(reduce(operator.concat, siteClasses.values())))
    if options.has_key('expand_diplotypes'): alignment = self.expandDiplotypes(alignment)
    self.taxonCounts = {}
    for population in self.populations.keys(): self.taxonCounts[population] = len(self.populations[population])
    self.siteCounts = {}
    for siteClass in self.siteClasses.keys(): self.siteCounts[siteClass] = len(self.siteClasses[siteClass])
    self.unambiguousSiteCounts = {}
    for siteClass in self.siteClasses.keys(): self.unambiguousSiteCounts[siteClass] = 0
    self.informativeSiteCounts = {}
    for siteClass in self.siteClasses.keys(): self.informativeSiteCounts[siteClass] = 0
    self.snds = []
    self.sndClasses = self.siteClasses.keys()
    self.simpleIndels = []
    self.complexIndels = []
    self.indelClasses = self.siteClasses.keys()
indelFlag = False
for site in self.siteClasses['_all']:
    nucleotideList = alignment.getNucleotides(site)
    nucleotideSet = set([nucleotideList[taxon] for taxon in self.populations['_all']])
    if len(nucleotideSet & ambiguousNucleotideSet) > 0: continue
    siteClassesThisSite = []
    for siteClass in self.siteClasses.keys():
        if site in self.siteClasses[siteClass]: siteClassesThisSite.append(siteClass)
    if '-' in nucleotideSet:
        if not indelFlag:
            indelFlag = True
            indelAlleles = [[nucleotide for nucleotide in nucleotideList] for siteClass in siteClassesThisSite]
            indelClasses = siteClassesThisSite
        else:
            if indelClasses == siteClassesThisSite:
                for taxon in range(alignment.getTaxonCount):
                    indelAlleles[taxon].append(nucleotideList[taxon])
            else:
                self._completeIndel(site, indelAlleles, indelClasses)
    else:
        self._completeIndel(site, indelAlleles, indelClasses)
        if len(nucleotideSet) > 1:
            self.snds.append(Snd(site, nucleotideList, self.populations, siteClassesThisSite))
        else:
            if indelFlag: self._completeIndel(self.siteCounts['_all'], indelAlleles, indelClasses)
    if 'coding' in self.siteClasses.keys():
        self.sndClasses += ['nonsynonymous', 'synonymous']
    if options.has_key('frame'):
        sequenceFrame = options['frame']
    else:
        sequenceFrame = 0
    for snd in self.snds:
        siteFrame = (sequenceFrame + snd.site) % 3
        if siteFrame == 0 and snd.site <= self.siteCounts['_all'] - 3:
            nucleotideList1 = alignment.getNucleotides(snd.site + 1)
            nucleotideSet1 = set([nucleotideList1[taxon] for taxon in self.populations['_all']])
            if len(nucleotideSet1 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet1:
                continue
            nucleotideList2 = alignment.getNucleotides(snd.site + 2)
            nucleotideSet2 = set([nucleotideList2[taxon] for taxon in self.populations['_all']])
            if len(nucleotideSet2 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet2:
                continue
            nucleotideList0 = alignment.getNucleotides(snd.site)
            nucleotideSet0 = set([nucleotideList0[taxon] for taxon in self.populations['_all']])
            codonSet = set([nucleotideList0[taxon] + nucleotideList1[taxon] + nucleotideList2[taxon] for taxon in self.populations['_all']])
            if len(set([aminoAcids[nucleotide + context[0] + context[1]] for nucleotide in nucleotideSet0])) > 1:
                snd.sndClasses.append('nonsynonymous')
break
else:
    snd.sndClasses.append('synonymous')

elif siteFrame == 1 and snd.site >= 1 and snd.site <= self.siteCounts["all"]-2:
    nucleotideList0 = alignment.getNucleotides(snd.site-1)
    nucleotideSet0 = set([nucleotideList0[taxon] for taxon in self.populations["all"]])
    if len(nucleotideSet0 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet0: continue
    nucleotideList1 = alignment.getNucleotides(snd.site+1)
    nucleotideSet1 = set([nucleotideList1[taxon] for taxon in self.populations["all"]])
    if len(nucleotideSet1 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet1: continue
    nucleotideList2 = alignment.getNucleotides(snd.site)
    nucleotideSet2 = set([nucleotideList2[taxon] for taxon in self.populations["all"]])
    if len(nucleotideSet2 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet2: continue

    codonSet = set([nucleotideList0[taxon]+nucleotideList1[taxon]+nucleotideList2[taxon] for taxon in self.populations["all"]])
    for context in set([codon[0]+codon[2] for codon in codonSet]):
        if len(set([aminoAcids[context[0]+context[2]+nucleotide] for nucleotide in nucleotideSet1])) > 1:
            snd.sndClasses.append('nonsynonymous')
    break
else:
    snd.sndClasses.append('synonymous')

elif siteFrame == 2 and snd.site >= 2:
    nucleotideList0 = alignment.getNucleotides(snd.site-2)
    nucleotideSet0 = set([nucleotideList0[taxon] for taxon in self.populations["all"]])
    if len(nucleotideSet0 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet0: continue
    nucleotideList1 = alignment.getNucleotides(snd.site-1)
    nucleotideSet1 = set([nucleotideList1[taxon] for taxon in self.populations["all"]])
    if len(nucleotideSet1 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet1: continue
    nucleotideList2 = alignment.getNucleotides(snd.site)
    nucleotideSet2 = set([nucleotideList2[taxon] for taxon in self.populations["all"]])
    if len(nucleotideSet2 & ambiguousNucleotideSet) > 0 or '-' in nucleotideSet2: continue

    codonSet = set([nucleotideList0[taxon]+nucleotideList1[taxon]+nucleotideList2[taxon] for taxon in self.populations["all"]])
    for context in set([codon[0]+codon[1] for codon in codonSet]):
        if len(set([aminoAcids[context[0]+context[1]+nucleotide] for nucleotide in nucleotideSet2])) > 1:
            snd.sndClasses.append('nonsynonymous')
    break
else:
    snd.sndClasses.append('synonymous')

self.gappedLength = self.siteCounts["all"] # Deprecated.
Self.ungappedLength = self.informativeSiteCounts["all"]

def _expandDiplotypes(self, alignment):
    newAlignment = MyAlignment()
    oldToNew = []
    for taxon in range(alignment.getTaxonCount()):
        sequence = alignment.getSequence(taxon)
        if taxon in self.options["expand_diplotypes"]:
            newSequence1 = list(sequence)
            newSequence2 = list(sequence)

            for site in range(len(sequence)):
                if sequence[site] == 'R':
                    newSequence1[site] = 'A'
                    newSequence2[site] = 'G'

            newAlignment.add_sequence(newSequence1)
            newAlignment.add_sequence(newSequence2)

    return newAlignment
elif sequence[site] == 'Y':
    newSequence1[site] = 'C'
    newSequence2[site] = 'T'
elif sequence[site] == 'W':
    newSequence1[site] = 'A'
    newSequence2[site] = 'T'
elif sequence[site] == 'S':
    newSequence1[site] = 'C'
    newSequence2[site] = 'G'
elif sequence[site] == 'M':
    newSequence1[site] = 'A'
    newSequence2[site] = 'C'
elif sequence[site] == 'K':
    newSequence1[site] = 'G'
    newSequence2[site] = 'T'
newSequence1 = ''.join(newSequence1)
newSequence2 = ''.join(newSequence2)
newAlignment.addSequence(newSequence1)
newAlignment.addSequence(newSequence2)
oldToNew.append([newTaxon, newTaxon+1])
newTaxon += 2
else:
    newAlignment.addSequence(sequence)
    oldToNew.append([newTaxon])
    newTaxon += 1
newPopulations = {}
for population in self.populations.keys():
    newPopulations[population] = reduce(operator.concat,
    [oldToNew[taxon] for taxon in self.populations[population]])
self.populations = newPopulations
return newAlignment

def _completeIndel(self, site, indelAlleles, indelClasses):
    indelAlleleList = [''.join(indelAllele) for indelAllele in indelAlleles]
    indelAlleleSet = set(indelAlleleList[taxon] for taxon in self.populations["_all"])
    indelSize = len(indelAlleleList[0])
    if len(indelAlleleSet) == 2 and '-'*indelSize in indelAlleleSet:
        self.simpleIndels.append(SimpleIndel(site-indelSize, indelSize, indelAlleleList, self.populations, indelClasses))
    else:
        self.complexIndels.append(ComplexIndel(site-indelSize, indelSize, indelAlleleList, self.populations, indelClasses))

class Statistics:
    def __init__(self, alignment, populations = {}, siteClasses = {}, options = {}):
        self.sample = Sample(alignment, populations, siteClasses, options)
        self.cache = {}
    def getTaxonCount(self, population = '_all'):
        return self.sample.taxonCounts[population]
    def getSiteCount(self, siteClass = '_all'):
        return self.sample.siteCounts[siteClass]
    def getUnambiguousSiteCount(self, siteClass = '_all'):
        return self.sample.unambiguousSiteCounts[siteClass]
    def getInformativeSiteCount(self, siteClass = '_all'):
        return self.sample.informativeSiteCounts[siteClass]
    def getSndCount(self, sndClass = '_all'):
sndCount = 0
for snd in self.sample.snds:
    if sndClass in snd.sndClasses: sndCount += 1
return sndCount

def getSimpleIndelCount(self, indelClass = '_all'):
    simpleIndelCount = 0
    for indel in self.sample.simpleIndels:
        if indelClass in indel.indelClasses: simpleIndelCount += 1
    return simpleIndelCount

def getComplexIndelCount(self, indelClass = '_all'):
    complexIndelCount = 0
    for indel in self.sample.complexIndels:
        if indelClass in indel.indelClasses: complexIndelCount += 1
    return complexIndelCount

def getIndelCount(self, indelClass = '_all'):
    indelCount = 0
    for indel in self.sample.simpleIndels + self.sample.complexIndels:
        if indelClass in indel.indelClasses: indelCount += 1
    return indelCount

# K is the number of polymorphic sites (also known as segregating sites) within a population (also
# known as S).
def getK(self, population = '_all', sndClass = '_all'):
    if self.cache.has_key(('K', population, sndClass)):
        return self.cache[('K', population, sndClass)]
    else:
        K = 0
        for snd in self.sample.snds:
            if len(snd.nucleotideFrequencies[population]) > 1 and sndClass in snd.sndClasses: K += 1
        self.cache[('K', population, sndClass)] = K
        return K

# thetaW is Watterson’s estimator of theta (4 N_e u).
def get_thetaW(self, population = '_all', sndClass = '_all'):
    if self.cache.has_key(('thetaW', population, sndClass)):
        return self.cache[('thetaW', population, sndClass)]
    else:
        a = 0.0
        for i in range(1, self.sample.taxonCounts[population]): a += 1.0/i
        if a > 0.0:
            thetaW = self.getK(population, sndClass)/a
        else:
            thetaW = 0.0
        self.cache[('thetaW', population, sndClass)] = thetaW
        return thetaW

def get_thetaW_PerSite(self, population = '_all', sndClass = '_all', siteClass = '_all'):
    length = self.sample.informativeSiteCounts[siteClass]
    if length > 0:
        thetaW_PerSite = self.get_thetaW(population, sndClass)/length
    else:
        thetaW_PerSite = 0.0
    return thetaW_PerSite

# pi is the average number of single nucleotide differences between two sequences from a population
# (also known as theta_pi).
Def get_pi(self, population = '_all', sndClass = '_all'):
if self.cache.has_key(('pi', population, sndClass)):
    return self.cache[('pi', population, sndClass)]
else:
    pi = 0
    n = self.sample.taxonCounts[population]
    if n > 1:
        for snd in self.sample.snds:
            if sndClass in snd.sndClasses:
                freqs = snd.nucleotideFrequencies[population]
                for nucl1 in freqs.keys():
                    for nucl2 in freqs.keys():
                        if nucl1 != nucl2:
                            pi += freqs[nucl1]*freqs[nucl2]
        pi /= 1.0*n*(n-1)
    self.cache[('pi', population, sndClass)] = pi
    return pi

def get_piPerSite(self, population = 'all', sndClass = '_all', siteClass = '_all'):
    length = self.sample.informativeSiteCounts[siteClass]
    if length > 0:
        piPerSite = self.get_pi(population, sndClass)/length
    else:
        piPerSite = 0.0
    return piPerSite

# D_T is Tajima's D.
def get_D_T(self, population = 'all', sndClass = '_all', siteClass = '_all'):
    n = self.sample.taxonCounts[population]
    K = self.getK(population, sndClass)
    if n > 1 and K > 0:
        a1 = 0.0
        a2 = 0.0
        for i in range(1, n):
            a1 += 1.0/i
            a2 += 1.0/(i*i)
        b1 = (n+1)/(3.0*n*(n-1))
        b2 = 2*(n*n + n + 3)/(9.0*n*(n-1))
        c1 = b1 - 1/a1
        c2 = b2 - (n+2)/(a1*n) + a2/(a1*a1)
        e1 = c1/a1
        e2 = c2/(a1*a1 + a2)
        D_T = (self.get_pi(population, sndClass)-self.get_thetaW(population, sndClass))/math.sqrt(e1*K + c2*K*(K-1))
    else:
        D_T = 0.0
    return D_T

# K_SimpleIndel is the number of simple indels polymorphic within a population.
def get_K_SimpleIndel(self, population = 'all', indelClass = '_all'):
    if self.cache.has_key(('K_SimpleIndel', population, indelClass)):
        return self.cache[('K_SimpleIndel', population, indelClass)]
    else:
        K = 0
        for indel in self.sample.simpleIndels:
            if len(indel.alleleFrequencies[population]) > 1 and indelClass in indel.indelClasses: K += 1
        self.cache[('K_SimpleIndel', population, indelClass)] = K
        return K

def getSingletonCountSimpleIndel(self, population = 'all', indelClass = '_all',...
singletonCount = 0
for indel in self.sample.simpleIndels:
    if len(indel.alleleFrequencies[population]) == 2 and 1 in indel.alleleFrequencies[population].values() and indelClass in indel.indelClasses:
        singletonCount += 1
return singletonCount

# thetaW_SimpleIndel is Watterson's estimator of theta (4 N_e u) for simple indels.
def get_thetaW_SimpleIndel(self, population = 'all', indelClass = '_all'):
    if self.cache.has_key(('thetaW_SimpleIndel', population, indelClass)):
        return self.cache[('thetaW_SimpleIndel', population, indelClass)]
    else:
        a = 0.0
        for i in range(1, self.sample.taxonCounts[population]): a += 1.0/i
        if a > 0.0:
            thetaW = self.getK_SimpleIndel(population, indelClass)/a
        else:
            thetaW = 0.0
        self.cache[('thetaW_SimpleIndel', population, indelClass)] = thetaW
    return thetaW

def get_thetaW_SimpleIndelPerSite(self, population = 'all', indelClass = '_all', siteClass = '_all'):
    length = self.sample.unambiguousSiteCounts[siteClass]
    if length > 0:
        thetaW_PerSite = self.get_thetaW_SimpleIndel(population, indelClass)/length
    else:
        thetaW_PerSite = 0.0
    return thetaW_PerSite

# piSimpleIndel is the average number of simple-indel differences between two sequences from a population.
def get_piSimpleIndel(self, population = 'all', indelClass = '_all'):
    if self.cache.has_key(('piSimpleIndel', population, indelClass)):
        return self.cache[('piSimpleIndel', population, indelClass)]
    else:
        pi = 0
        n = self.sample.taxonCounts[population]
        if n > 1:
            for indel in self.sample.simpleIndels:
                if indelClass in indel.indelClasses:
                    freqs = indel.alleleFrequencies[population]
                    for all in freqs.keys():
                        for al2 in freqs.keys():
                            if all != al2: pi += freqs[all]*freqs[al2]
                    pi /= 1.0*n*(n-1)
            self.cache[('piSimpleIndel', population, indelClass)] = pi
        return pi

def get_piSimpleIndelPerSite(self, population = 'all', indelClass = '_all', siteClass = '_all'):
    length = self.sample.unambiguousSiteCounts[siteClass]
    if length > 0:
        piPerSite = self.get_piSimpleIndel(population, indelClass)/length
    else:
        piPerSite = 0.0
    return piPerSite

# D_T_SimpleIndel is Tajima's D for simple indels.
def getD_T_SimpleIndel(self, population = 'all', indelClass = '_all'):
    n = self.sample.taxonCounts[population]
    K = self.getK_SimpleIndel(population, indelClass)
if n > 1 and K > 0:
    a1 = 0.0
    a2 = 0.0
    for i in range(1, n):
        a1 += 1.0/i
        a2 += 1.0/(i*i)
    b1 = (n+1)/(3.0*(n-1))
    b2 = 2*(n*n + n + 3)/(9.0*n*(n-1))
    c1 = b1 - 1/a1
    c2 = b2 - (n+2)/(a1*n) + a2/(a1*a1)
    e1 = c1/a1
    e2 = c2/(a1*a1 + a2)
    D_T = (self.get_piSimpleIndel(population, indelClass)-self.get_thetaW_SimpleIndel(population, indelClass))/math.sqrt(e1*K + e2*K*(K-l))
else:
    D_T = 0.0
return D_T

# piBW is Balhoff and Wray’s measure of indel polymorphism.
def get_piBW(self, population = 'all', indelClass = '_all'):
    if self.cache.has_key(('piBW', population, indelClass)):
        return self.cache[('piBW', population, indelClass)]
    else:
        piBW = 0.0
        n = self.sample.taxonCounts[population]
        if n > 1:
            for indel in self.sample.simpleIndels+self.sample.complexIndels:
                if indelClass in indel.indelClasses:
                    freqs = indel.alleleFrequencies[population]
                    for all in freqs.keys():
                        for al2 in freqs.keys():
                            if all != al2: piBW += freqs[all]*freqs[al2]*self._piBW_weight(all, al2, indel.size)
                            piBW /= 1.0*n*(n-l)
        self.cache[('piBW', population, indelClass)] = piBW
        return piBW

def _piBW_weight(self, all, al2, size):
    efSizes = []
    efSize1 = 0
    efSize2 = 0
    for site in range(size):
        if all[site] == '-':
            if al2[site] != '-': efSize1 += 1
        else:
            if efSize1 > 0:
                efSizes.append(efSize1)
                efSize1 = 0
            if al2[site] == '-':
                if all[site] != '-': efSize2 += 1
        else:
            if efSize2 > 0:
                efSizes.append(efSize2)
                efSize2 = 0
    if efSize1 > 0: efSizes.append(efSize1)
    if efSize2 > 0: efSizes.append(efSize2)
    weight = 0.0
for efSize in efSizes: weight += 1.0 + math.log(efSize, 10)
return weight
def get_piBW_PerSite(self, population = '_all', indelClass = '_all', siteClass = '_all'):
    length = self.sample.unambiguousSiteCounts[siteClass]
    if length > 0:
        piBW_PerSite = self.get_piBW(population, indelClass)/length
    else:
        piBW_PerSite = 0.0
    return piBW_PerSite

# L is the number of substituted sites (also known as fixed differences) between two populations (also
# known as D).
# Following the McDonald–Kreitman convention, if a site is polymorphic within either population, it
# isn’t substituted.
def getL(self, population1, population2, sndClass = '_all'):
    (population1, population2) = self._orderPopulations(population1, population2)
    if self.cache.has_key(('L', population1, population2, sndClass)):
        return self.cache[('L', population1, population2, sndClass)]
    else:
        L = 0
        for snd in self.sample.snds:
            if sndClass in snd.sndClasses:
                freqs1 = snd.nucleotideFrequencies[population1]
                freqs2 = snd.nucleotideFrequencies[population2]
                if len(freqs1) > 1 and len(freqs2) > 1 and freqs1.keys()[0] != freqs2.keys()[0]: L += 1
                self.cache[('L', population1, population2, sndClass)] = L
        return L

# D_FL is Fu and Li’s D.
# The first population is the ingroup, and the second is the outgroup.
# Following the DnaSP convention, polymorphic sites in the ingroup that can’t be polarized using the
# outgroup are ignored.
def getD_FL(self, population1, population2, sndClass = '_all'):
    n = self.sample.taxonCounts[population1]
    eta = 0
    eta_e = 0
    for snd in self.sample.snds:
        if sndClass in snd.sndClasses:
            freqs1 = snd.nucleotideFrequencies[population1]
            freqs2 = snd.nucleotideFrequencies[population2]
            if len(freqs1) > 1 and len(freqs2) > 1 and freqs2.keys()[0] in freqs1.keys():
                eta += 1
                if freqs1[freqs2.keys()[0]] == n-1: eta_e += 1
    if n > 1 and eta > 0:
        a = 0.0
        b = 0.0
        for i in range(1, n):
            a += 1.0/i
            b += 1.0/(i*i)
        if n == 2:
            c = 1.0
        else:
            c = 2*(n*a - 2*(n-1))/((n-1)*(n-2))
            v = 1 + a*a*(c - (n+1)/(1.0*(n-1)))/(b + a*a)
            u = a-1-v
            90
\[ D_{FL} = (\eta - a \times \eta_e) / \sqrt{u \times \eta + v \times \eta \times \eta} \]

else:
\[ D_{FL} = 0.0 \]

return \( D_{FL} \)

# \( F_{FL} \) is Fu and Li's \( F \).
# The first population is the ingroup, and the second is the outgroup.
# Following the DnaSP convention, polymorphic sites in the ingroup that can't be polarized using the outgroup are ignored.

def getF_FL(self, population1, population2, sndClass = '_all'):
    n = self.sample.taxonCounts[population1]
    pi = 0
    eta = 0
    eta_e = 0
    for snd in self.sample.snds:
        if sndClass in snd.sndClasses:
            freqs1 = snd.nucleotideFrequencies[population1]
            freqs2 = snd.nucleotideFrequencies[population2]
            if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                for nuc1 in freqs1.keys():
                    for nuc2 in freqs1.keys():
                        if nuc1 != nuc2:
                            pi += freqs1[nuc1] * freqs1[nuc2]
                            eta += 1
                            if freqs1[freqs2.keys()[0]] == n-1:
                                eta_e += 1
                        if n > 1 and eta > 0:
                            pi /= 1.0 * n * (n-1)
                            a = 0.0
                            b = 0.0
                            for i in range(1, n):
                                a += 1.0 / i
                                b += 1.0 / (i * i)
                            if n == 2:
                                c = 1.0
                            else:
                                c = 2 * (n * a - 2 * (n-1)) / ((n-1) * (n-2))
                            v = (c + 2.0 * (n * n + n + 3) / (9 * n * (n-1))) - 2.0 / (n-1) / (b + a * a)
                            u = (1 + (n+1) * (3.0 * (n-1)) - 4.0 * (n+1) * (a + 1.0 / n - 2.0 * n / (n+l)) / ((n-1)**2)) / a - v
                            F_FL = (pi - eta_e) / math.sqrt(u * eta + v * eta * eta)
            else:
                F_FL = 0.0
    return F_FL

# \( \theta_{H} \) is Fay and Wu’s estimator of \( \theta \) (\( 4 \times N_e \times u \)).
# The first population is the ingroup, and the second is the outgroup.
# Following the DnaSP convention, polymorphic sites in the ingroup that can’t be polarized using the outgroup are ignored.

def get_thetaH(self, population1, population2, sndClass = '_all'):
    if self.cache.has_key(('thetaH', population1, population2, sndClass)):
        return self.cache[('thetaH', population1, population2, sndClass)]
    else:
        thetaH = 0.0
        n = self.sample.taxonCounts[population1]
        if n > 1:
            S = {}
            for i in range(1, n):
                S[i] = 0
            return self.cache[('thetaH', population1, population2, sndClass)]
        else:
            self.cache[('thetaH', population1, population2, sndClass)] = thetaH
            return thetaH
if sndClass in snd.sndClasses:
    freqs1 = snd.nucleotideFrequencies[population1]
    freqs2 = snd.nucleotideFrequencies[population2]
    if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
        S[n-freqs1[freqs2.keys()[0]]] += 1
        for i in range(1, n):
            thetaH += S[i]*i*i
        thetaH *= 2.0/(n*(n-1))
        self.cache[('thetaH', population1, population2, sndClass)] = thetaH
        return thetaH

def get_thetaH_PerSite(self, population1, population2, sndClass = '_all', siteClass = '_all'):
    length = self.sample.informativeSiteCounts[siteClass]
    if length > 0:
        thetaH_PerSite = self.get_thetaH(population1, population2, sndClass)/length
    else:
        thetaH_PerSite = 0.0
    return thetaH_PerSite

# H is Fay and Wu's H.
# The first population is the ingroup, and the second is the outgroup.
# Following the DnaSP convention, polymorphic sites in the ingroup that can't be polarized using the
# outgroup are ignored.
def getH(self, population1, population2, sndClass = '_all'):
    n = self.sample.taxonCounts[population1]
    theta_pi = 0
    if n > 1:
        S = {}
        for i in range(1, n):
            S[i] = 0
        for snd in self.sample.snds:  
            if sndClass in snd.sndClasses:
                freqs1 = snd.nucleotideFrequencies[population1]
                freqs2 = snd.nucleotideFrequencies[population2]
                if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                    S[n-freqs1[freqs2.keys()[0]]] += 1
                    for i in range(1, n):
                        theta_pi += S[i]*i*(n-i)
                    theta_pi *= 2.0/(n*(n-1))
        H = theta_pi-self.get_thetaH(population1, population2, sndClass)
        return H

def getH_PerSite(self, population1, population2, sndClass = '_all', siteClass = '_all'):
    length = self.sample.informativeSiteCounts[siteClass]
    if length > 0:
        H_PerSite = self.getH(population1, population2, sndClass)/length
    else:
        H_PerSite = 0.0
    return H_PerSite

# HKApValue is Hudson–Kreitman–Aguade p-value for two “loci”.
# The first SND class is the first “locus”, and the second SND class is the second “locus”.
def getHKApValue(self, population1, population2, sndClass1, sndClass2):
    n1 = self.sample.taxonCounts[population1]
    n2 = self.sample.taxonCounts[population2]
    if n1 > n2:
        population2, population1 = population1, population2
    n2, n1 = n1, n2
    if n1 > 1:
        K11 = 1.0*self.getK(population1, sndClass1)
K12 = 1.0*self.getK(population1, sndClass2)
K1 = K11+K12
if K1 > 0.0:
    K21 = 1.0*self.getK(population2, sndClass1)
    K22 = 1.0*self.getK(population2, sndClass2)
    K2 = K21+K22
    D1 = 1.0*self.getL(population1, population2, sndClass1)
    D2 = 1.0*self.getL(population1, population2, sndClass2)
    D = D1+D2
    a1 = a2 = b1 = b2 = 0.0
    for i in range(1, n1):
        a1 += 1.0/i
        b1 += 1.0/(i*i)
    for i in range(1, n2):
        a2 += 1.0/i
        b2 += 1.0/(i*i)
    f = (K2/a2)/(K1/a1)
    g = (1.0+f)/2.0
    T = D/(K1/a1) - g
    theta1 = (K11+K21+D1)/(a1 + f*a2 + g + T)
    theta2 = (K12+K22+D2)/(a1 + f*a2 + g + T)
    EK11 = a1*theta1
    EK12 = a2*theta1
    EK21 = a1*theta2
    EK22 = a2*theta2
    ED1 = (g+T)*theta1
    ED2 = (g+T)*theta2
    VK11 = EK11 + bl*theta1**2
    VK12 = EK12 + bl*theta2**2
    VK21 = EK21 + b2*theta1**2
    VK22 = EK22 + b2*theta2**2
    VD1 = ED1 + (g*theta1)**2
    VD2 = ED2 + (g*theta2)**2
    X2 = 0.0
    if VK11 != 0.0: X2 += (K11-EK11)**2/VK11
    if VK12 != 0.0: X2 += (K12-EK12)**2/VK12
    if VK21 != 0.0: X2 += (K21-EK21)**2/VK21
    if VK22 != 0.0: X2 += (K22-EK22)**2/VK22
    if VD1 != 0.0: X2 += (D1-ED1)**2/VD1
    if VD2 != 0.0: X2 += (D2-ED2)**2/VD2
    p = chiSquared.pValue(X2, 2)
else:
    p = float('nan')
else:
    if n2 > 1:
        K1 = 1.0*self.getK(population2, sndClass1)
        K2 = 1.0*self.getK(population2, sndClass2)
        K = K1+K2
        if K > 0.0:
            D1 = 1.0*self.getL(population1, population2, sndClass1)
            D2 = 1.0*self.getL(population1, population2, sndClass2)
            D = D1+D2
            a = b = 0.0
            for i in range(1, n2):
                a += 1.0/i

93
b += 1.0/(i*i)

T = D/(K/a) - 1.0

theta1 = (K1+D1)/(a + 1.0 + T)
theta2 = (K2+D2)/(a + 1.0 + T)

EK1 = a*theta1
EK2 = a*theta2
ED1 = (1.0+T)*theta1
ED2 = (1.0+T)*theta2

VK1 = EK1 + b*theta1**2
VK2 = EK2 + b*theta2**2
VD1 = ED1 + theta1**2
VD2 = ED2 + theta2**2

X2 = 0.0
if VK1 != 0.0: X2 += (K1-EK1)**2/VK1
if VK2 != 0.0: X2 += (K2-EK2)**2/VK2
if VD1 != 0.0: X2 += (D1-ED1)**2/VD1
if VD2 != 0.0: X2 += (D2-ED2)**2/VD2

p = chiSquared.pValue(X2, 1)
else:
    p = float('nan')
else:
    p = float('nan')
return p

# D_FL_SimpleIndel is Fu and Li's D for simple indels.
# The first population is the ingroup, and the second is the outgroup.
# Simple indels in the ingroup that can't be polarized using the outgroup are ignored.
def getD_FL_SimpleIndel(self, population1, population2, indelClass = 'all'):
    n = self.sample.taxonCounts[population1]
    eta = 0
    eta_e = 0
    for indel in self.sample.simpleIndels:
        if indelClass in indel.indelClasses:
            ffeqsl = indel.alleleFrequencies[population1]
            ffeqs2 = indel.alleleFrequencies[population2]
            if len(ffeqsl) > 1  and len(ffeqs2) == 1  and ffeqs2.keys()[0] in ffeqsl.keys():
                eta += 1
                if ffeqsl[ffeqs2.keys()[0]] == n-1: eta_e += 1
            if n > 1  and eta > 0:
                a = 0.0
                b = 0.0
                for i in range(1, n):
                    a += 1.0/i
                    b += 1.0/(i*i)
                if n == 2:
                    c = 1.0
                else:
                    c = 2*(n*a - 2*(n-1))/((n-1)*(n-2))
                    v = 1 + a*a*(c - (n+1)/(1.0*(n-1)))/(b + a*a)
                    u = a-1-v
                    D_FL = (eta - a*eta_e)/math.sqrt(u*eta + v*eta*eta)
                else:
                    D_FL = 0.0
            else:
                D_FL = 0.0
return D_FL
# F_FL_SimpleIndel is Fu and Li’s F for simple indels.
# The first population is the ingroup, and the second is the outgroup.
# Simple indels in the ingroup that can’t be polarized using the outgroup are ignored.
Def getF_FL_SimpleIndel(self, population1, population2, indelClass = '_all'):
    n = self.sample.taxonCounts[population1]
    pi = 0
    eta = 0
    eta_e = 0
    for indel in self.sample.simpleIndels:
        if indelClass in indel.indelClasses:
            freqs1 = indel.alleleFrequencies[population1]
            freqs2 = indel.alleleFrequencies[population2]
            if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                for all in freqs1.keys():
                    for al2 in freqs1.keys():
                        if all != al2: pi += freqs1[all]*freqs1[al2]
                    eta += 1
                    if freqs1[freqs2.keys()[0]] == n-1: eta_e += 1
            if n > 1 and eta > 0:
                pi /= 1.0*n*(n-1)
                a = 0.0
                b = 0.0
                for i in range(1, n):
                    a += 1.0/i
                    b += 1.0/(i*i)
                if n == 2:
                    c = 2*(n*a - 2*(n-1))/((n-1)*(n-2))
                else:
                    c = 2*(n*a - 2*(n-1))/((n-1)*(n-2))
                v = (c + 2.0*(n*n + n + 3)/(9*n*(n-1)) - 2.0/(n-1))/(b + a*a)
                u = (1 + (n+1)/((3.0*(n-1)) - 4.0*(n+1)*(a + 1.0/n - 2.0*n/(n+1)))/((n-1)**2))/a - v
                F_FL = (pi - eta_e)/math.sqrt(u*eta + v*eta*eta)
        else:
            F_FL = 0.0
    return F_FL_SimpleIndel

# thetaH_SimpleIndel is Fay and Wu’s estimator of theta (4 N_e u) for simple indels.
# The first population is the ingroup, and the second is the outgroup.
# Simple indels in the ingroup that can’t be polarized using the outgroup are ignored.
Def get_thetaH_SimpleIndel(self, population1, population2, indelClass = '_all'):
    if self.cache.has_key(('thetaH_SimpleIndel', population1, population2, indelClass)):
        return self.cache[('thetaH_SimpleIndel', population1, population2, indelClass)]
    else:
        thetaH = 0.0
        n = self.sample.taxonCounts[population1]
        if n > 1:
            S = {}
            for i in range(1, n): S[i] = 0
            for indel in self.sample.simpleIndels:
                if indelClass in indel.indelClasses:
                    freqs1 = indel.alleleFrequencies[population1]
                    freqs2 = indel.alleleFrequencies[population2]
                    if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                        S[n-freqs1[freqs2.keys()[0]]] += 1
        else:
            thetaH = 0.0
            n = self.sample.taxonCounts[population1]
            if n > 1:
                S = {}
                for i in range(1, n): S[i] = 0
                for indel in self.sample.simpleIndels:
                    if indelClass in indel.indelClasses:
                        freqs1 = indel.alleleFrequencies[population1]
                        freqs2 = indel.alleleFrequencies[population2]
                        if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                            S[n-freqs1[freqs2.keys()[0]]] += 1
35
for i in range(1, n): thetaH += S[i]*i*i
thetaH *= 2.0/(n*(n-1))
self.cache[('thetaH_SimpleIndel', population1, population2, indelClass)] = thetaH
return thetaH

def get_thetaH_SimpleIndelPerSite(self, population1, population2, indelClass = 'all', siteClass = 'all'):
    length = self.sample.unambiguousSiteCounts[siteClass]
    if length > 0:
        thetaH_PerSite = self.get_thetaH_SimpleIndel(population1, population2, indelClass)/length
    else:
        thetaH_PerSite = 0.0
    return thetaH_PerSite

# H_SimpleIndel is Fay and Wu’s H for simple indels.
# The first population is the ingroup, and the second is the outgroup.
# Simple indels in the ingroup that can’t be polarized using the outgroup are ignored.

def getH_SimpleIndel(self, population1, population2, indelClass = 'all'):
    n = self.sample.taxonCounts[population1]
    theta_pi = 0
    if n > 1:
        S = {}
        for i in range(1, n): S[i] = 0
        for indel in self.sample.simpleIndels:
            if indelClass in indel.indelClasses:
                freqs1 = indel.alleleFrequencies[population1]
                freqs2 = indel.alleleFrequencies[population2]
                if len(freqs1) > 1 and len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys(): S[n-freqs2.keys()[0]] += 1
        for i in range(1, n): theta_pi += S[i]*i*(n-i)
    theta_pi *= 2.0/(n*(n-1))
    H = theta_pi-self.get_thetaH_SimpleIndel(population1, population2, indelClass)
    return H

def getH_SimpleIndelPerSite(self, population1, population2, indelClass = 'all', siteClass = 'all'):
    length = self.sample.unambiguousSiteCounts[siteClass]
    if length > 0:
        H_PerSite = self.getH_SimpleIndel(population1, population2, indelClass)/length
    else:
        H_PerSite = 0.0
    return H_PerSite

class Slider:
    def __init__(self, alignment, halfWidth, slide, populations = {}, siteClasses = {}, options = {}):
        self.halfWidth = halfWidth
        self.slide = slide
        self.statisticsHash = {}
        length = alignment.getSiteCount()
        center = 0
        while center < length:
            windowAlignment = MyAlignment()
            left = max(center-halfWidth, 0)
            right = min(center+halfWidth+1, length)
            for taxon in range(alignment.getTaxonCount()):
                windowAlignment.addSequence(alignment.getSequence(taxon)[left:right])
            windowSiteClasses = {}
            for siteClass in siteClasses:
windowSiteClasses[siteClass] = []
for site in siteClasses[siteClass]:
    if left <= site and site < right:
        windowSiteClasses[siteClass].append(site-left)
self.statisticsHash[center] = Statistics(windowAlignment, populations, windowSiteClasses, options)
center += slide
def get_thetaW_PerSite(self, population = '_all', sndClass = '_all', siteClass = '_all'):
pairs = [(center, self.statisticsHash[center].get_thetaW_PerSite(population, sndClass, siteClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs
def get_piPerSite(self, population = '_all', sndClass = '_all', siteClass = '_all'):
pairs = [(center, self.statisticsHash[center].get_piPerSite(population, sndClass, siteClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs
def getD_T(self, population = '_all', sndClass = '_all'):
pairs = [(center, self.statisticsHash[center].getD_T(population, sndClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs
def get_thetaW_SimpleIndelPerSite(self, population = '_all', indelClass = '_all', siteClass = '_all'):
pairs = [(center, self.statisticsHash[center].get_thetaW_SimpleIndelPerSite(population, indelClass, siteClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs
def getD_T_SimpleIndel(self, population = '_all', indelClass = '_all'):
pairs = [(center, self.statisticsHash[center].getD_T_SimpleIndel(population, indelClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs
def getD_FL(self, population1, population2, sndClass = '_all'):
pairs = [(center, self.statisticsHash[center].getD_FL(population1, population2, sndClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs
def getF_FL(self, population1, population2, sndClass = '_all'):
pairs = [(center, self.statisticsHash[center].getF_FL(population1, population2, sndClass)) for center in self.statisticsHash.keys()]
pairs.sort()
return pairs

def get_thetaH_PerSite(self, population1, population2, sndClass = '_all', siteClass = '_all'):
    pairs = [(center, self.statisticsHash[center].get_thetaH_PerSite(population1, population2, sndClass, siteClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def getH_PerSite(self, population1, population2, sndClass = '_all', siteClass = '_all'):
    pairs = [(center, self.statisticsHash[center].getH_PerSite(population1, population2, sndClass, siteClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def getF_ST(self, population, sndClass = '_all'):
    pairs = [(center, self.statisticsHash[center].getF_ST(population, sndClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def getPairwiseF_ST(self, population1, population2, sndClass = '_all'):
    pairs = [(center, self.statisticsHash[center].getPairwiseF_ST(population1, population2, sndClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def get_rhSimpleIndelPerSite(self, population1, population2, indelClass = '_all', siteClass = '_all'):
    pairs = [(center, self.statisticsHash[center].get_rhSimpleIndelPerSite(population1, population2, indelClass, siteClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def getD_FL_SimpleIndel(self, population1, population2, indelClass = '_all'):
    pairs = [(center, self.statisticsHash[center].getD_FL_SimpleIndel(population1, population2, indelClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def getF_FL_SimpleIndel(self, population1, population2, indelClass = '_all'):
    pairs = [(center, self.statisticsHash[center].getF_FL_SimpleIndel(population1, population2, indelClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def get_thetaH_SimpleIndelPerSite(self, population1, population2, indelClass = '_all', siteClass = '_all'):
    pairs = [(center, self.statisticsHash[center].get_thetaH_SimpleIndelPerSite(population1, population2, indelClass, siteClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

def getH_SimpleIndelPerSite(self, population1, population2, indelClass = '_all', siteClass = '_all'):
    pairs = [(center, self.statisticsHash[center].getH_SimpleIndelPerSite(population1, population2, indelClass, siteClass)) for center in self.statisticsHash.keys()]
    pairs.sort()
    return pairs

# A hap file is an input file to Hudson's program exhap (Hudson, 2001, Genetics 159:1805–1817; http://home.uchicago.edu/~rhudson1).
# The first population is the ingroup, and the second is the outgroup.
Def AlignmentToHapFile(alignment, ilename, populations, population1 = '_all', population2 = None):
    sample = Sample(alignment, populations)
    alleleLists = {}}
ancestralAlleles = {}
for snd in sample.snds:
    freqs1 = snd.nucleotideFrequencies[population1]
    if len(freqs1) == 2:
        site = snd.site
        alleleLists[site] = snd.nucleotides[population1]
        ancestralAlleles[site] = '?'
        if population2 != None:
            freqs2 = snd.nucleotideFrequencies[population2]
            if len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                ancestralAlleles[site] = freqs2.keys()[0]
for indel in sample.simpleIndels+sample.complexIndels:
    freqs1 = indel.alleleFrequencies[population1]
    if len(freqs1) == 2:
        site = indel.site
        alleleLists[site] = indel.alleles[population1]
        ancestralAlleles[site] = '?'
        if population2 != None:
            freqs2 = indel.alleleFrequencies[population2]
            if len(freqs2) == 1 and freqs2.keys()[0] in freqs1.keys():
                ancestralAlleles[site] = freqs2.keys()[0]
sites = alleleLists.keys()
sites.sort()
siteCount = len(sites)
taxonCount = sample.taxonCounts[population1]
handle = open(ilename, 'w')
handle.write("%d %d\n" % (taxonCount, siteCount))
for site in sites: handle.write("%d " % site)
handle.write("\n")
handle.write("a ")
for site in sites: handle.write("%s " % ancestralAlleles[site])
handle.write("\n")
for taxon in range(taxonCount):
    handle.write("%d " % taxon)
    for site in sites: handle.write("%s " % alleleLists[site][taxon])
    handle.write("\n")
handle.close()
Generating statistical values

Our analyses was performed with three different loci with alleles sequenced from a natural *X. laevis* (XLL) population and several closely related *Xenopus* species (e.g. *X. amieti*; XA). In order to run these analyses you must first generate a FASTA or PHYLP (non-interleaved) alignment. The range of population sequences in the alignment is specified by the range "XLL" (27) and the location of the outgroup [27] . After some basic analyses, a sliding-window analyses using a window of 101 (50+1+50) bases and a slide of 10 bases is conducted with several of the summary statistics. Finally, a .hap file is generated with our data, which is useful if you have Richard Hudson’s programs *exhap* and *maxhap* for estimating recombination (http://home.uchicago.edu/~rhudson1).

```python
#! /usr/bin/env python

from populationGenetics import *

alignment = FASTA_FileToMyAlignment("GAD67.promoter.TBAa.txt")
populations = {'XLL': range(27), 'XA': [27]}
statistics = Statistics(alignment, populations)
print "XLL sequences: %d" % statistics.getTaxonCount('XLL')
print "XA sequences: %d" % statistics.getTaxonCount('XA')
print "total length: %d" % statistics.getSiteCount() 
print "unambiguous length: %d" % statistics.getUnambiguousSiteCount()
print "informative length: %d" % statistics.getInformativeSiteCount()
print "K: %d" % statistics.getK('XLL')
print "singleton count: %d" % statistics.getSingletonCount('XLL')
print "thetaW per site: %g" % statistics.get_thetaW_PerSite('XLL')
print "pi per site: %g" % statistics.get_piPerSite('XLL')
print "D_T: %g" % statistics.getD_T('XLL')
print "L: %d" % statistics.getL('XLL', 'XA')
print "rho per site: %g" % statistics.get_rhoPerSite('XLL', 'XA')
print "H: %g" % statistics.getH('XLL', 'XA')
print "thetaH_Persite: %g" % statistics.get_thetaH('XLL', 'XA')
print "pi: %g" % statistics.get_pi('XLL')
print "K_SimpleIndel: %g" % statistics.getK_SimpleIndel('XLL')
print "thetaW_SimpleIndel: %g" % statistics.get_thetaW_SimpleIndel('XLL')
print "piSimpleIndel: %g" % statistics.get_piSimpleIndel('XLL')
print "piSimpleIndelPerSite: %g" % statistics.get_piSimpleIndelPerSite('XLL')
```

print "D_T_SimpleIndel: %g" % statistics.getD_T_SimpleIndel('XLL')
print "D_FL_SimpleIndel: %g" % statistics.getD_FL_SimpleIndel('XLL', 'XA')
print "piBW: %g" % statistics.get_piBW('XLL')
print "piBW_PerSite: %g" % statistics.get_piBW_PerSite('XLL')
slider = Slider(alignment, 50, 10, populations)
D_Ts = slider.getD_T('XLL')
file = open("GAD67promoter_XA_D_T", 'w')
for D_T in D_Ts: file.write("%d %g\n"%D_T)
file.close()
D_FLs = slider.getD_FL('XLL', 'XA')
file = open("GAD67promoter_XA_D_FL", 'w')
for D_FL in D_FLs: file.write("%d %g\n"%D_FL)
file.close()
H_PerSites = slider.getH_PerSite('XLL', 'XA')
file = open("GAD67promoter_XA_H_PerSite", 'w')
for H_PerSite in H_PerSites: file.write("%d %g\n"%H_PerSite)
file.close()
pi_PerSites = slider.get_piPerSite('XLL')
file = open("GAD67promoter_XA_pi_PerSite", 'w')
for pi_PerSite in pi_PerSites: file.write("%d %g\n"%pi_PerSite)
file.close()
piBW_PerSites = slider.get_piBW_PerSite('XLL')
file = open("GAD67promoter_XA_piBW_PerSite", 'w')
for piBW_PerSite in piBW_PerSites: file.write("%d %g\n"%piBW_PerSite)
file.close()
D_T_SimpleIndels= slider.getD_T_SimpleIndel('XLL')
file = open("GAD67promoter_XA_D_T_SimpleIndel", 'w')
for D_T_SimpleIndel in D_T_SimpleIndels: file.write("%d %g\n"%D_T_SimpleIndel)
file.close()
D_FL_SimpleIndels = slider.getD_FL_SimpleIndel('XLL', 'XA')
file = open("GAD67promoter_XA_D_FL_SimpleIndel", 'w')
for D_FL_SimpleIndel in D_FL_SimpleIndels: file.write("%d %g\n"%D_FL_SimpleIndel)
file.close()
AlignmentToHapFile(alignment, "GAD67.promoter_XA.hap", populations, 'XLL', 'XA')
Source code for running the HKA test

Summary code for implementing the HKA test of a neutral model is provided below. This program requires two different FASTA alignment files: One alignment of a region of evolutionary interests (i.e. promoter) and is defined as bindingAlignment, and a second alignment that will serve as a neutral proxy (i.e. intron) and is defined as nonbindingAlignment. A chi-squared distribution is implement in C and not provided here.

```python
#! /usr/bin/env python
from populationGenetics import *

bindingAlignment = FASTA_FileToMyAlignment("GAD67.promoter.HKA.txt")
bindingLength = bindingAlignment.getSiteCount()
nonbindingAlignment = FASTA_FileToMyAlignment("GAD67.Intron15.CLUSTALa.txt")
nonbindingLength = nonbindingAlignment.getSiteCount()
alignment = MyAlignment.concatenation(bindingAlignment, nonbindingAlignment)
populations = {'XLL': range(24), 'XA': [24]}
siteClasses = {'binding': range(bindingLength), 'nonbinding': range(bindingLength, bindingLength+nonbindingLength)}
statistics = Statistics(alignment, populations, siteClasses)
print "HKA p-value: %g" % statistics.getHKApValue('XLL', 'XA', 'binding', 'nonbinding')
```
**Estimating \( \rho \) with Hudson’s exhap and maxhap programs**

To estimate \( \rho \) from sequence alignment data, we used R.H. software to generate a haplotype file (e.g. filename.hap), and ran the following command in an UNIX shell

\[
\% \text{exhap} \text{< YourData.hap > YourData.pairs}
\]

to convert a haplotype data file into a file with the necessary Hudson pairs data format. The command

\[
\% \text{maxhap 1 h27rho 0.01 100 0.01 0 0 1 0 < YourData.pairs}
\]

is instructs maxhap to estimate \( \rho \) for a dataset. The command line arguments tell maxhap to use one two-site configuration file of two-locus sampling probabilities for 27 samples. Maxhap will begin to search for the maximum composite-likelihood score starting from 0.01 x 0.01 to a maximum value 100 x 0.01 in equally spaced increments. The last four values specify gene conversion parameters which were not included in our analysis.
Generating sample populations with Hudson’s ms by coalescent simulation

Hudson’s ms program is available online (http://home.uchicago.edu/~rhudson1/). To run the program download the ms tarball and cd to the downloaded directory in a UNIX shell. The command

```bash
% ms 27 100000 -s 46 -r 0 973 | sample_stats > SummaryStats_sample_stats
```

uses ms to generate random genealogies for 100,000 replicate sample populations of 27 haploid individuals at a 973 bp locus. A total of 46 mutations will be Poisson distributed along each replicate sample genealogy. The output data is then submitted to sample_stats to extract summary statistics for each replicate sample, which is imported into the file SummaryStats_sample_stats. To assess the statistical significance of an observed test statistic run

```bash
% cut -f 6 SummaryStats_sample_stats | ./stats 0.024
```

which uses the UNIX command cut to extract the 6th column of data from the SummaryStats_sample_stats file and assess the value at the 2.4 percentile.
**TBA evolutionary tree**

In order to run a TBA alignment, a binary tree must be specified in a modified Newick format that details the evolutionary relationship of the sequence included. Branch lengths are not required. A tree for 27 *X. laevis* alleles ('XLL') and four outgroups species ('XLS', 'XA', 'XC', 'XM') is provided. The 'XLL1 XLL2' group merely seeds the alignment.

Example:

(((((((((((((XLL1 XLL2) XLL3) XLL4) XLL5) XLL6 XLL7) XLL8) XLL9) XLL10) XLL11) XLL12) XLL13) XLL14) XLL15) XLL16) XLL17) XLL18) XLL19) XLL20) XLL21) XLL22) XLL23) XLL24) XLL25) XLL26) XLL27) XLS) XA) XC) XM)
**ClustalX 2.0.11 alignment of the upstream region of xGAD67.**

Sequences for 27 *X. laevis* alleles (XLL1 – XLL26) and four closely related outgroup species (XA, *X. amieti*; XC, *X. clivii*; XLS, *X. l. sudanensis*; XM, *X. muelleri*) were aligned under default parameters: gap opening and extension penalties 15 and 6.66, respectively. The transcriptional start site is indicated by +1 and the initiating methionine is underlined. Sequences proceed from 5' to 3' and include a portion of the first exon. Conserved sites are indicated by an asterisk (*).
**TBA alignment of the upstream region of xGAD67.**

Twenty seven X. l. laevis alleles (XLL1 – XLL26) and four closely related outgroup species (XA, X. amieti; XC, X. clivii; XLS, X. l. sudanensis; XM, X. muelleri) were aligned using TBA with a modified Newick format parameter file (provided in the Appendix) describing the evolutionary relationship of outgroup species (Evans et al., 2004). The transcriptional start site is indicated by +1 and the initiating methionine is underlined. Sequences proceed from 5’ to 3’ and include a portion of the first exon. Conserved sites are indicated by an asterisk (*).
CTATGCACCTATAGCAAGCCTCTTGCCATTGCGAGAACCAC---

-------      CCCACCCTGCCCACTAAATACTGACTTTCTATGGCACCTTA

---------- CCCACCTGCCCACTAAATACTGACTTTCTATGGCACCTTATAGCAGCCCCTCTGGCCTTTGCCAGAACCCAC

----------------------------------
Xenopus Phylogeny

The phylogenetic relationship of African clawed frogs was investigated by Evans et al. (2004) and reproduced here. A phylogenetic tree is provided where branch lengths are proportional to divergence time as estimated from the maximum likelihood topology and a relaxed molecular clock (Evans et al., 2004).


