**APPENDIX S3. COMMUNITY OCCUPANCY MODEL JAGS CODE**

Below we present R code for importing three data files (BirdCounts.csv, HabitatVars.csv, and SurveyVars.csv), and formatting those data for analysis in our community occupancy model. We then provide the full specification of the hierarchical model, and associated R code for running the model in JAGS (Plummer, 2003) using the R package ‘jagsUI’ (v. 1.4.2; Kellner, 2016). Though we also fit a model that included an interaction term between forest patch size and forest amount, the model presented excludes this term, because it was non-significant for 98% of species examined.

In order to assess model fit, we used a Bayesian p-value approach (Gelman, Meng, & Stern, 1996; Zipkin, DeWan, & Royle, 2009; Kéry & Royle, 2016). At each iteration of the MCMC chain, we calculated a discrepancy statistic (D) as the sum of the squared differences between each observed data point (yijklm, below), and its expected value (Eijklm) under the fitted model. We then simulated a unique dataset based on the fitted model (ysim) and calculated the same discrepancy statistic (Dsim) for this dataset as well. Thus, the posterior distribution of Dsim provides a reference distribution against which to compare D. We calculated our p-value as the proportion of the time (out of 6000 retained MCMC iterations) D was larger than Dsim.

**References**

Gelman, A., Meng, X.-L., & Stern, H. (1996) Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica*, 6, 733-760.

Kellner, K. (2016) jagsUI: a wrapper around ‘rjags’ to streamline ‘JAGS’ analyses. R package version 1.4.2. http://CRAN.R-project.org/package=jagsUI.

Kéry, M., & Royle, J.A (2016) *Applied Hierarchical Modeling in Ecology, Volume 1*. London, United Kingdom: Elsevier Inc.

Plummer, M. (2003) JAGS: a program for analysis of Bayesian graphical models using Gibbs sampling. In K. Hornik, F. Leisch, & A. Zeileis (Eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing* (pp. 1-10). Vienna, Austria: Technische Universität Wien.

Zipkin, E.F., DeWan, A., & Royle, J.A. (2009) Impacts of forest fragmentation on species richness: A hierarchical approach to community modelling. *Journal of Applied Ecology*, 46, 815–822.

**R and JAGS code**

#Load jagsUI package

library(jagsUI)

#Import data and create a unique line for each combination of year, patch, point,

#survey, and species

birds = read.csv("BirdCounts.csv")

habitat = read.csv("HabitatVars.csv")

data = merge(birds, habitat, by=c("num.patch", "num.point", "year", "num.year"))

tmp1 = expand.grid("num.year"=unique(data$num.year), "num.patch"=unique(data$num.patch), "num.point" = unique(data$num.point), "surveyorder"=unique(data$surveyorder), "num.species"=unique(data$num.species))

data = merge(tmp1, data, by=c("num.year", "num.patch", "num.point", "surveyorder", "num.species"), all=T)

#Create arrays for edge distance, patch size, and habitat amount.

tmp1 = unique(data[, c("num.patch", "num.point", "edge.dist", "amount.prop", "Area\_ha")])

tmp1$amount.prop = scale(tmp1$amount.prop, center=T, scale=T)

tmp1$edge.dist = scale(tmp1$edge.dist, center=T, scale=T)

tmp1$Area\_ha = scale(log(tmp1$Area\_ha), center=T, scale=T)

tmp1 = tmp1[order(tmp1$num.point, tmp1$num.patch),]

edge = array(tmp1$edge.dist, c(202, 10))

amount = array(tmp1$amount.prop, c(202, 10))

size = array(tmp1$Area\_ha, c(202, 10))

#Conduct principal components analysis on local vegetation variables, and create

#arrays for PC1, PC2, PC3, and tree basal area

tmp1 = unique(data[which(!is.na(data$shrubtotal)),c("num.patch", "num.point", "num.year", "shrubtotal", "tottrees", "canopy\_vvt", "leafmean", "vfd5m")])

tmp1$shrubtotal = scale(tmp1$shrubtotal, center=T, scale=T)

tmp1$tottrees = scale(tmp1$tottrees, center=T, scale=T)

tmp1$canopy\_vvt = scale(tmp1$canopy\_vvt, center=T, scale=T)

tmp1$vfd5m = scale(tmp1$vfd5m, center=T, scale=T)

tmp1$leafmean = scale(tmp1$leafmean, center=T, scale=T)

tmp2 = prcomp(~tottrees+shrubtotal+canopy\_vvt+vfd5m+leafmean, data=tmp1)

tmp1 = cbind(tmp1, tmp2$x)

tmp2 = expand.grid("num.year"=c(1:3), "num.point"=c(1:10), "num.patch"=c(1:202))

tmp1 = merge(tmp1, tmp2, by=c("num.year", "num.point", "num.patch"), all=T)

tmp1 = tmp1[order(tmp1$num.year, tmp1$num.point, tmp1$num.patch),]

pc1 = array(tmp1$PC1, c(202, 10, 3))

pc2 = array(tmp1$PC2, c(202, 10, 3))

pc3 = array(tmp1$PC3, c(202, 10, 3))

trees = array(tmp1$tottrees, c(202, 10, 3))

#Create array for response variable

data$count = ifelse(data$count > 0, 1, 0)

data = data[order(data$surveyorder, data$num.year, data$num.point, data$num.patch, data$num.species),]

y = array(data$count, c(52,202,10,3,3))

#Create vector for number of sampling sites (points) in each patch

tmp1 = data[which(!is.na(data$count) & data$num.year==1),]

tmp1 = aggregate(num.point ~ num.patch, data=tmp1, FUN="max")

tmp1 = tmp1[order(tmp1$num.patch),]

nsites = tmp1$num.point

#Create vector for group membership of each species

tmp1 = unique(data[-which(is.na(data$num.group)),c("num.species", "num.group")])

tmp1 = tmp1[order(tmp1$num.species),]

group = tmp1$num.group

#Read in Julian dates and create array

tmp = read.csv("SurveyVars.csv")

tmp$julian = scale(tmp$julian, center=T, scale=T)

tmp2 = expand.grid("num.year"=unique(tmp$num.year), "num.patch"=unique(tmp$num.patch), "num.point" = unique(tmp$num.point), "surveyorder"=unique(tmp$surveyorder))

tmp = merge(tmp, tmp2, by=c("num.year", "num.patch", "num.point", "surveyorder"), all=T)

tmp = tmp[order(tmp$surveyorder, tmp$num.year, tmp$num.point, tmp$num.patch),]

julian = array(tmp$julian, c(202, 10, 3, 3))

#Model specification

sink("Model.txt")

cat("model{

mu.alpha0 ~ dnorm(0, 0.001)

sigma.alpha0 ~ dunif(0, 5)

tau.alpha0 <- 1/(sigma.alpha0\*sigma.alpha0)

mu.alpha1 ~ dnorm(0, 0.001)

sigma.alpha1 ~ dunif(0, 5)

tau.alpha1 <- 1/(sigma.alpha1\*sigma.alpha1)

mu.alpha2 ~ dnorm(0, 0.001)

sigma.alpha2 ~ dunif(0, 5)

tau.alpha2 <- 1/(sigma.alpha2\*sigma.alpha2)

for(h in 1:ngroups){

sigma.patch[h] ~ dunif(0, 5)

tau.patch[h] <- 1/(sigma.patch[h]\*sigma.patch[h])

mu.beta0[h] ~ dnorm(0, 0.001)

sigma.beta0[h] ~ dunif(0, 5)

tau.beta0[h] <- 1/(sigma.beta0[h]\*sigma.beta0[h])

mu.beta1[h] ~ dnorm(0, 0.001)

sigma.beta1[h] ~ dunif(0, 5)

tau.beta1[h] <- 1/(sigma.beta1[h]\*sigma.beta1[h])

mu.beta2[h] ~ dnorm(0, 0.001)

sigma.beta2[h] ~ dunif(0, 5)

tau.beta2[h] <- 1/(sigma.beta2[h]\*sigma.beta2[h])

mu.beta3[h] ~ dnorm(0, 0.001)

sigma.beta3[h] ~ dunif(0, 5)

tau.beta3[h] <- 1/(sigma.beta3[h]\*sigma.beta3[h])

mu.beta4[h] ~ dnorm(0, 0.001)

sigma.beta4[h] ~ dunif(0, 5)

tau.beta4[h] <- 1/(sigma.beta4[h]\*sigma.beta4[h])

mu.beta5[h] ~ dnorm(0, 0.001)

sigma.beta5[h] ~ dunif(0, 5)

tau.beta5[h] <- 1/(sigma.beta5[h]\*sigma.beta5[h])

mu.beta6[h] ~ dnorm(0, 0.001)

sigma.beta6[h] ~ dunif(0, 5)

tau.beta6[h] <- 1/(sigma.beta6[h]\*sigma.beta6[h])

mu.gamma0[h] ~ dnorm(0, 0.001)

sigma.gamma0[h] ~ dunif(0, 5)

tau.gamma0[h] <- 1/(sigma.gamma0[h]\*sigma.gamma0[h])

mu.gamma1[h] ~ dnorm(0, 0.001)

sigma.gamma1[h] ~ dunif(0, 5)

tau.gamma1[h] <- 1/(sigma.gamma1[h]\*sigma.gamma1[h])

for(j in 1:npatch){

for(l in 1:nyears){

beta0.patch[h,j,l] ~ dnorm(0, tau.patch[h])

}

}

}

for(i in 1:nspecies){

alpha0[i] ~ dnorm(mu.alpha0, tau.alpha0)

alpha1[i] ~ dnorm(mu.alpha1, tau.alpha1)

alpha2[i] ~ dnorm(mu.alpha2, tau.alpha2)

beta0[i] ~ dnorm(mu.beta0[group[i]], tau.beta0[group[i]])

beta1[i] ~ dnorm(mu.beta1[group[i]], tau.beta1[group[i]])

beta2[i] ~ dnorm(mu.beta2[group[i]], tau.beta2[group[i]])

beta3[i] ~ dnorm(mu.beta3[group[i]], tau.beta3[group[i]])

beta4[i] ~ dnorm(mu.beta4[group[i]], tau.beta4[group[i]])

beta5[i] ~ dnorm(mu.beta5[group[i]], tau.beta5[group[i]])

beta6[i] ~ dnorm(mu.beta6[group[i]], tau.beta6[group[i]])

gamma0[i] ~ dnorm(mu.gamma0[group[i]], tau.gamma0[group[i]])

gamma1[i] ~ dnorm(mu.gamma1[group[i]], tau.gamma1[group[i]])

for(j in 1:npatch){

for(k in 1:nsites[j]){

Z[i,j,k,1] ~ dbern(psi[i,j,k,1])

logit(psi[i,j,k,1]) <- beta0[i] + beta0.patch[group[i], j, 1] +

beta1[i]\*pc1[j,k,1] + beta2[i]\*pc2[j,k,1] +

beta3[i]\*pc3[j,k,1] + beta4[i]\*edge[j,k] + beta5[i]\*size[j,k] +

beta6[i]\*amount[j,k]

for(l in 2:nyears){

Z[i,j,k,l] ~ dbern(psi[i,j,k,l])

logit(psi[i,j,k,l]) <- gamma0[i] + gamma1[i]\*Z[i,j,k,l-1] +

beta0.patch[group[i], j, l] +

beta1[i]\*pc1[j,k,l] + beta2[i]\*pc2[j,k,l] + beta3[i]\*pc3[j,k,l] +

beta4[i]\*edge[j,k] + beta5[i]\*size[j,k] + beta6[i]\*amount[j,k]

}

for(l in 1:nyears){

for(m in 1:nsurveys){

logit(p[i,j,k,l,m]) <- alpha0[i] + alpha1[i]\*julian[j,k,l,m] +

alpha2[i]\*trees[j,k,l]

y[i,j,k,l,m] ~ dbern(Z[i,j,k,l]\*p[i,j,k,l,m])

ysim[i,j,k,l,m] ~ dbern(Z[i,j,k,l]\*p[i,j,k,l,m])

E[i,j,k,l,m] <- Z[i,j,k,l]\*p[i,j,k,l,m]

D[i,j,k,l,m] <- pow(y[i,j,k,l,m]-E[i,j,k,l,m],2)

Dsim[i,j,k,l,m] <- pow(ysim[i,j,k,l,m]-E[i,j,k,l,m],2)

}

D.1[i,j,k,l] <- sum(D[i,j,k,l,1:nsurveys])

Dsim.1[i,j,k,l] <- sum(Dsim[i,j,k,l,1:nsurveys])

}

D.2[i,j,k] <- sum(D.1[i,j,k,1:nyears])

Dsim.2[i,j,k] <- sum(Dsim.1[i,j,k,1:nyears])

}

D.3[i,j] <- sum(D.2[i,j,1:nsites[j]])

Dsim.3[i,j] <- sum(Dsim.2[i,j,1:nsites[j]])

}

D.4[i] <- sum(D.3[i,1:npatch])

Dsim.4[i] <- sum(Dsim.3[i,1:npatch])

}

fit.data <- sum(D.4[1:nspecies])

fit.sim <- sum(Dsim.4[1:nspecies])

bpv <- fit.data - fit.sim

}", fill=T)

sink()

#Specify model values

win.data = list(y=y, nsites=nsites, nsurveys=3, nyears=3, npatch=202, nspecies=52, ngroups=3, group=group,size=size, amount=amount, edge=edge, pc1=pc1, pc2=pc2, pc3=pc3, julian=julian, trees=trees)

#Initial values for Z

Zst = aggregate(count~num.year+num.point+num.patch+num.species, data=data, FUN="max")

tmp1 = expand.grid("num.year"= c(1:3), "num.point" = c(1:10), "num.patch" = c(1:202), "num.species"=c(1:52))

Zst = merge(Zst, tmp1, by=c("num.year", "num.point", "num.patch", "num.species"), all=T)

Zst = Zst[order(Zst$num.year, Zst$num.point, Zst$num.patch, Zst$num.species),]

Zst = array(Zst$count, c(52, 202, 10, 3))

#Specify the remainder of the initial values

inits=function()list(Z=Zst, mu.alpha0=rnorm(1), sigma.alpha0=runif(1), mu.alpha1=rnorm(1), sigma.alpha1=runif(1), mu.alpha2=rnorm(1), sigma.alpha2=runif(1), mu.beta0=rnorm(3), sigma.beta0=runif(3), mu.beta1=rnorm(3), sigma.beta1=runif(3), mu.beta2=rnorm(3), sigma.beta2=runif(3), mu.beta3=rnorm(3), sigma.beta3=runif(3), mu.beta4=rnorm(3), sigma.beta4=runif(3), mu.beta5=rnorm(3), sigma.beta5=runif(3), mu.beta6=rnorm(3), sigma.beta6=runif(3), mu.gamma0=rnorm(3), sigma.gamma0=runif(3), mu.gamma1=rnorm(3), sigma.gamma1=runif(3), sigma.patch=runif(3))

#Identify parameters to track

params=c("alpha0", "mu.alpha0", "sigma.alpha0", "alpha1", "mu.alpha1", "sigma.alpha1", "alpha2", "mu.alpha2", "sigma.alpha2", "beta0", "mu.beta0", "sigma.beta0", "gamma0", "mu.gamma0", "sigma.gamma0", "gamma1", "mu.gamma1", "sigma.gamma1", "beta1", "mu.beta1", "sigma.beta1", "beta2", "mu.beta2", "sigma.beta2", "beta3", "mu.beta3", "sigma.beta3", "beta4", "mu.beta4", "sigma.beta4", "beta5", "mu.beta5", "sigma.beta5", "beta6", "mu.beta6", "sigma.beta6", "sigma.patch", "fit.data", "fit.sim", "bpv")

#Chains, iterations, burnin, and thinning

nc=3; ni=400000; nb=200000; nt=100

#Submit model to JAGS. Remove "parallel=T" argument for non-parallel processing

out = jags(win.data, inits, params, "Model.txt", n.chains=nc, n.thin=nt, n.iter=ni, n.burnin=nb, parallel=T)