Methods

Matt Williamson

6/24/2019

Methods

Our interest was in evaluating the bias that arises from (potentially unmodeled) variation in the probability of reporting an easement (given that it is there), the probability that a spatial location is available for an easement, and spatial autocorrelation in both occupancy and reporting probability.

Generating covariate observations

We used the tracts and block groups for Iowa as the basis for developing our simulated datasets. We generated random data values for each of three predictors at the tract-level and two predictors at the block group-level. Because predictors may also be spatially autocorrelated, we simulated these data from $x_i \sim \mathcal{N}_{\|}(\boldsymbol{\mu} = 0, \boldsymbol{\Sigma} = \phi)$ where $\phi \sim \mathcal{N}(0, [\tau(D - \rho W)]^{-1})$ and τ, the spatially varying precision paramater is 1; ρ, the strength of spatial dependence is 0.3, D is a diagonal matrix containing the number of neighbors for a given location, and W is an adjacency matrix. We defined adjacency by determining the minimum distance necessary to ensure that all locations had at least one neighbor and considering to locations to be adjacent if they were within that distance from each other.

```
# get a dummy geography
prj <- "+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +towgs84=0,
st <- "IA"
geog.tct <- tracts(state = st) %>% as(., "sf") %>% st_transform(.,
    crs = prj)
geog.bg <- block_groups(state = st) %>% as(., "sf") %>% st_transform(.,
    crs = prj)
m.tct <- 4  #number of tract-level params on occurence
m.bg <- 3  #Number of block-group level params on reporting
# function to generate the precision matrix
gen_sp_precMatx <- function(geog, rho, tau) {
    geog.order <- geog[order(geog$GEOID), ] %>% as(., "Spatial")
    coords.geog <- coordinates(geog.order)
    # estimate distance to first neareset neighbor
    geog.nearnb <- knn2nb(knearneigh(coords.geog, k = 1), row.names = geog.order$GEOID,
        sym = TRUE)
    dis.nb.list <- dnearneigh(coords.geog, 0, max(unlist(nbdists(geog.nearnb,
        coords.geog))), row.names = geog.order$GEOID)
    # neighbors within the minimum distance to ensure at least 1
    # neighbor converts list of neighbors to matrix with 0s on
    # diagonal (i.e., no self-neighbors)
    geog.W.matx <- nb2mat(dis.nb.list, style = "B", zero.policy = TRUE)
    # creates a matrix with the number of neighbors as the
    # diagonal
    D.geog <- diag(rowSums(geog.W.matx))
    prec.matx <- tau * (D.geog - rho * geog.W.matx)
    return(list(geog.W.matx, prec.matx))
}
# generate tract-level precision matrix for observation data
```

```r
obs.prec.matx.tct <- gen_sp_precMatx(geog = geog.tct, rho = 0.3,
    tau = 1)
# convert to covariance matrix
obs.cov.matx.tct <- solve(obs.prec.matx.tct[[2]])
# generate block group-level precision matrix for observation
# data
obs.prec.matx.bg <- gen_sp_precMatx(geog = geog.bg, rho = 0.3,
    tau = 1)
obs.cov.matx.bg <- solve(obs.prec.matx.bg[[2]])
# create a tract-level matrix with an intercept and
# observations for m.tct predictors
X.tct <- matrix(c(rep(1, nrow(geog.tct)), mvnfast::rmvn(m.tct -
    1, rep(0, NROW(obs.cov.matx.tct)), obs.cov.matx.tct, ncores = 10)),
    ncol = m.tct)
# scale non-intercept predictors
X.tct[, 2:m.tct] <- apply(X.tct[, 2:m.tct], 2, scale)
# create a block group matrix with an intercept and
# observations for m.bg predictors
X.bg <- matrix(c(rep(1, nrow(geog.bg)), mvnfast::rmvn(m.bg -
    1, rep(0, NROW(obs.cov.matx.bg)), obs.cov.matx.bg, ncores = 10)),
    ncol = m.bg)
X.bg[, 2:m.bg] <- apply(X.bg[, 2:m.bg], 2, scale)
```

Latin Hypercube Sampling

We used a Latin hypercube design to evaluate model performance across a range of plausible parameter values. For each simulation replicate, we drew 300 uniformly distributed samples across the multi-dimensional space described by the limits in Table 1.

```r
### Generate the paramater values from the Latin hypercube
set.seed(82980)
h <- 300  #number of samples to draw from lhs space
latin.hyp <- maximinLHS(h, 6)
# set the range for each value
occ.min <- lhs.params[1, 2]
occ.max <- lhs.params[1, 3]
p.min <- lhs.params[2, 2]
p.max <- lhs.params[2, 3]
occ.rho.min <- lhs.params[3, 2]
occ.rho.max <- lhs.params[3, 3]
p.rho.min <- lhs.params[4, 2]  #per M. Hooten anything below 0.5 is not likely to matter
p.rho.max <- lhs.params[4, 3]
tau.min <- lhs.params[5, 2]
tau.max <- lhs.params[5, 3]
sub.occ.min <- lhs.params[6, 2]  #this is the propbability that at subunit is available
sub.occ.max <- lhs.params[6, 3]
# Map params to to hypercube
params.set <- cbind(occRHO = latin.hyp[, 1] * (occ.rho.max -
    occ.rho.min) + occ.rho.min, pRHO = latin.hyp[, 2] * (p.rho.max -
    p.rho.min) + p.rho.min, subOcc = latin.hyp[, 3] * (sub.occ.max -
    sub.occ.min) + sub.occ.min, occ = latin.hyp[, 4] * (occ.max -
    occ.min) + occ.min, p = latin.hyp[, 5] * (p.max - p.min) +
    p.min, tau = latin.hyp[, 6] * (tau.max - tau.min) + tau.min)
```

Generating observations

We simulated true easement occurrence for each sample generating a probability of easement occurrence for each tract by multiplying ψ (the mean occupancy probability for a given Latin hypercube sample [i.e., the intercept]) and three randomly generated regression coefficients by the tract-level design matrix and adding ϕ_Ψ where ϕ is described above and τ and ρ_Ψ were taken from the appropriate Latin hypercube sample. This resulted in true easement occurence (z_i) for each tract which we then converted to an observed occupancy dataset by generating an estimate of p (the block group reporting probability) by multiplying p (the mean reporting probability for a given Latin hypercube sample [i.e., the intercept]) and two randomly generated regression coefficents by the block group-level design matrix and adding ϕ_p with τ and ρ_p as above. Finally, because the probability of an easement being reported (conditional on its existence) depends on both the probability of reporting (p) and the probability that a block group is available for an easement (α), we simulated easement observations following $y_{i,j} \sim \text{Bern}(\alpha p)$.

Table 1: Range of values used in Latin Hypercube sampler where Ψ is the probability of easement occurence, p is detection probability, $\rho_{...}$ is the strength of spatial autocorrelation for Ψ or p, τ is the precision for the conditional autoregressive term, and α is the probability that a location is available for an easement.

Variables	Lower	Upper
Ψ	0.20	0.80
p	0.30	0.98
ρ_Ψ	0.50	1.00
ρ_p	0.50	1.00
τ	0.10	1.00
α	0.20	0.80

```r
# generate regression coeffiecient values, intercept needs to
# be converted to logit scale
b.tct <- c(unname(boot::logit(params.set[inputn, 4])), runif(m.tct -
    1, -3, 3))
inputn <- 1  # counter for batch-processing on our cluster, setting to 1
# generate precision matrix for occurrence
err.prec.matx.tct <- gen_sp_precMatx(geog = geog.tct, rho = params.set[inputn,
    1], tau = params.set[inputn, 6])
err.cov.matx.tct <- solve(err.prec.matx.tct[[2]])
# generate values of phi from mutlivariate normal dist with
# Sigma = phi
phi.occ <- mvnfast::rmvn(1, rep(0, nrow(err.cov.matx.tct)), err.cov.matx.tct,
    ncores = 10)
logit.psi <- X.tct %*% b.tct + t(phi.occ)
# genrate z
z <- rbinom(nrow(geog.tct), size = 1, prob = boot::inv.logit(logit.psi))
# Generate detection data generate regression coeffiecient
# values, intercept needs to be converted to logit scale
b.bg <- c(unname(boot::logit(params.set[inputn, 5])), runif(m.bg -
    1, -3, 3))
# count the number of blockgroups within each tract (surveys
# within site)
survey_summary <- geog.bg %>% group_by(., STATEFP, COUNTYFP,
    TRACTCE) %>% summarise(count = n())
n_survey <- as.vector(survey_summary$count)
total_surveys <- nrow(geog.bg)
# generate phi for reporting process
err.prec.matx.bg <- gen_sp_precMatx(geog = geog.bg, rho = params.set[inputn,
    2], tau = params.set[inputn, 6])
err.cov.matx.bg <- solve(err.prec.matx.bg[[2]])
# generate values of phi from mutlivariate normal dist with
# Sigma = phi
phi.p <- mvnfast::rmvn(1, rep(0, nrow(err.cov.matx.bg)), err.cov.matx.bg,
    ncores = 10)
```

```r
logit.p <- X.bg %*% b.bg + t(phi.p)
# for each block group within a tract assign whether it is
# available based on avail ~ Binomial(n_survey, alpha)
a <- unlist(lapply(n_survey, function(x) rbinom(x, 1, params.set[inputn,
    3])))
p <- boot::inv.logit(logit.p)
# y is 0 if z=0 or if a = 0
survey.df <- tibble(site = rep(1:nrow(geog.tct), n_survey), siteID = rep(geog.tct$GEOID,
    n_survey)) %>% mutate(y = rbinom(n = total_surveys, size = 1,
    prob = z[site] * a * p))
# get start and end indices to extract slices of y for each
# site
start_idx <- rep(0, nrow(geog.tct))
end_idx <- rep(0, nrow(geog.tct))
for (i in 1:nrow(geog.tct)) {
    if (n_survey[i] > 0) {
        site_indices <- which(survey.df$site == i)
        start_idx[i] <- site_indices[1]
        end_idx[i] <- site_indices[n_survey[i]]
    }
}
any_seen <- rep(0, nrow(geog.tct))
for (i in 1:nrow(geog.tct)) {
    if (n_survey[i] > 0) {
        any_seen[i] <- max(survey.df$y[start_idx[i]:end_idx[i]])
    }
}
```

Fitting models

We generated 50 replicates [only showing results from 4 here] of each Latin hypercube sample (1500 simulated datasets total) and fit each of five models (a standard occupancy model, a binomial model with a conditional autoregressive [CAR] term for spatial autocorrelation, an occupancy model with a CAR term for occurrence only, an occupancy model with a CAR term for reporting only, and an occupancy model with a CAR terms for both components) to the simulated dataset. All models were fit in R using rstan, a wrapper to the Stan language (stan models are at end of document). Models were fit using an adaptation parameter of 0.98 and and a maximum treedepth of 16 with four chains each run for 3700 iterations (with a warmup period of 3200 iterations). We calcaulated the relative bias for the intercepts and regression coefficients as:

$$RelBias = \frac{\hat{\beta} - \beta}{|\beta|}$$

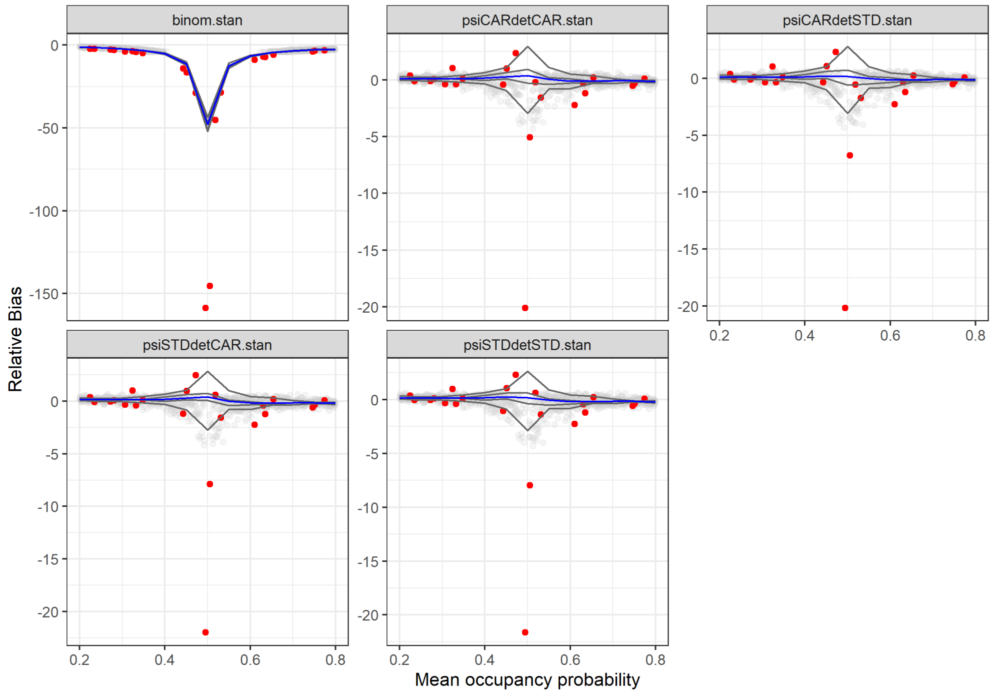

Figure 1: Relative bias of the posterior estimates of the intercept in the binomial model and the intercept in the occupancy component of the occupancy modes for each simulated dataset. Gray dots are median estimates of relative bias from each model run, gray lines are replicate-specific the relationship between median values of relative bias and the simulated mean occupancy probability, blue lines depict the relationship between median values of relative bias and occupancy probability across all simulation replicates, red dots indicate simulation runs where both the probability of detection and the probability of availibility were in the lower decile.

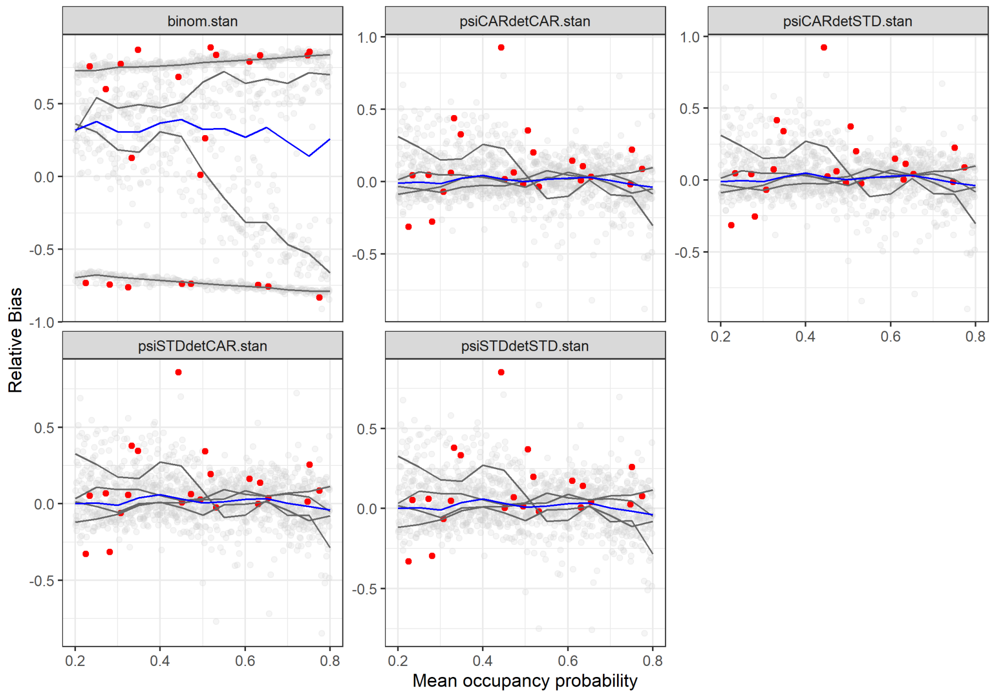

Figure 2: Relative bias of the posterior estimates of the regression coefficients in the binomial model and the intercept in the occupancy component of the occupancy modes for each simulated dataset. Gray dots are median estimates of relative bias from each model run, gray lines are replicate-specific the relationship between median values of relative bias and the simulated mean occupancy probability, blue lines depict the relationship between median values of relative bias and occupancy probability across all simulation replicates, red dots indicate simulation runs where both the probability of detection and the probability of availibility were in the lower decile.

Stan models

Binomial

```
functions{
/**
* Return log probability of a unit-scale proper conditional autoregressive
  * (CAR) prior with a sparse representation for the adjacency matrix
  *
  * @param phi Vector containing the parameters with a CAR prior
  * @param alpha Dependence (usually spatial) parameter for the CAR prior (real)
  * @param W_sparse Sparse representation of adjacency matrix (int array)
  * @param n Length of phi (int)
  * @param W_n Number of adjacent pairs (int)
  * @param D_sparse Number of neighbors for each location (vector)
  * @param lambda Eigenvalues of D^{-1/2}*W*D^{-1/2} (vector)
  *
  * @return Log probability density of CAR prior up to additive constant
  */
  real sparse_car_lpdf(vector phi, real alpha,
    int[,] W_sparse, vector D_sparse, vector lambda, int nobs, int W_n) {
      row_vector[nobs] phit_D; // phi' * D
      row_vector[nobs] phit_W; // phi' * W
      vector[nobs] ldet_terms;

      phit_D = (phi .* D_sparse)';
      phit_W = rep_row_vector(0, nobs);
      for (i in 1:W_n) {
        phit_W[W_sparse[i, 1]] += phi[W_sparse[i, 2]];
        phit_W[W_sparse[i, 2]] += phi[W_sparse[i, 1]];
      }

      for (i in 1:nobs) ldet_terms[i] = log1m(alpha * lambda[i]);
      return 0.5 * (sum(ldet_terms)
                    - (phit_D * phi - alpha * (phit_W * phi)));
  }
}
data {
  // site-level occupancy covariates
  int<lower = 1> n_site;
  int<lower = 1> m_psi;
  matrix[n_site, m_psi] X_tct;


  // summary of whether species is known to be present at each site
  int<lower = 0, upper = 1> any_seen[n_site];

  // number of surveys at each site
  int<lower = 0> n_survey[n_site];

  matrix<lower = 0, upper = 1>[n_site, n_site] W_tct; //adjacency matrix tract
  int W_n_tct; //number of adjacent pairs
}
transformed data {
  int W_sparse_occ[W_n_tct, 2];   // adjacency pairs
  vector[n_site] D_sparse_occ;     // diagonal of D (number of neigbors for each site)

  vector[n_site] lambda_occ;       // eigenvalues of invsqrtD * W * invsqrtD
  { // generate sparse representation for W
  int counter_occ;
  counter_occ = 1;
  // loop over upper triangular part of W to identify neighbor pairs
    for (i in 1:(n_site - 1)) {
      for (j in (i + 1):n_site) {
        if (W_tct[i, j] == 1) {
          W_sparse_occ[counter_occ, 1] = i;
          W_sparse_occ[counter_occ, 2] = j;
          counter_occ += 1;
        }
      }
    }
  }
  for (i in 1:n_site) D_sparse_occ[i] = sum(W_tct[i]);
  {
    vector[n_site] invsqrtD_occ;
    for (i in 1:n_site) {
      invsqrtD_occ[i] = 1 / sqrt(D_sparse_occ[i]);
    }
    lambda_occ = eigenvalues_sym(quad_form(W_tct, diag_matrix(invsqrtD_occ)));
  }

}
parameters {
  vector[n_site] phi_occ;
  real<lower = 0, upper =0.999> alpha_occ;
  real<lower = 0> sigma_occ;
  vector[m_psi] beta_psi;
}
transformed parameters {
  vector[n_site] logit_psi = X_tct * beta_psi + phi_occ * sigma_occ;
}
model {
   phi_occ ~ sparse_car(alpha_occ, W_sparse_occ, D_sparse_occ, lambda_occ, n_site, W_n_tct);
  alpha_occ ~ beta(4,1);
  sigma_occ ~ normal(0, 1.5);
  beta_psi ~ student_t(7.763,0, 1.566);

  any_seen ~ binomial_logit(n_survey, logit_psi);
}
```

Occupancy without CAR

Note that although this code has the functions to estimate ϕ, the actual model does not do so.

```
functions{
/**
* Return log probability of a unit-scale proper conditional autoregressive
  * (CAR) prior with a sparse representation for the adjacency matrix
  *
  * @param phi Vector containing the parameters with a CAR prior
  * @param alpha Dependence (usually spatial) parameter for the CAR prior (real)
  * @param W_sparse Sparse representation of adjacency matrix (int array)
  * @param n Length of phi (int)
  * @param W_n Number of adjacent pairs (int)
  * @param D_sparse Number of neighbors for each location (vector)
  * @param lambda Eigenvalues of D^{-1/2}*W*D^{-1/2} (vector)
  *
  * @return Log probability density of CAR prior up to additive constant
  */
  real sparse_car_lpdf(vector phi, real alpha,
    int[,] W_sparse, vector D_sparse, vector lambda, int nobs, int W_n) {
      row_vector[nobs] phit_D; // phi' * D
      row_vector[nobs] phit_W; // phi' * W
      vector[nobs] ldet_terms;

      phit_D = (phi .* D_sparse)';
      phit_W = rep_row_vector(0, nobs);
      for (i in 1:W_n) {
        phit_W[W_sparse[i, 1]] += phi[W_sparse[i, 2]];
        phit_W[W_sparse[i, 2]] += phi[W_sparse[i, 1]];
      }

      for (i in 1:nobs) ldet_terms[i] = log1m(alpha * lambda[i]);
      return 0.5 * (sum(ldet_terms)
                    - (phit_D * phi - alpha * (phit_W * phi)));
  }
}
data {
  // site-level occupancy covariates
  int<lower = 1> n_site;
  int<lower = 1> m_psi;
  matrix[n_site, m_psi] X_tct;

  // survey-level detection covariates
  int<lower = 1> total_surveys;
  int<lower = 1> m_p;
  matrix[total_surveys, m_p] X_bg;

  // survey level information
  int<lower = 1, upper = n_site> site[total_surveys];
  int<lower = 0, upper = 1> y[total_surveys];
  int<lower = 0, upper = total_surveys> start_idx[n_site];
  int<lower = 0, upper = total_surveys> end_idx[n_site];

  // summary of whether species is known to be present at each site
  int<lower = 0, upper = 1> any_seen[n_site];

  // number of surveys at each site
  int<lower = 0> n_survey[n_site];

}
parameters {
  vector[m_psi] beta_psi;
  vector[m_p] beta_p;
}
transformed parameters {
  vector[total_surveys] logit_p = X_bg * beta_p;
  vector[n_site] logit_psi = X_tct * beta_psi;
}
model {
  vector[n_site] log_psi = log_inv_logit(logit_psi);
  vector[n_site] log1m_psi = log1m_inv_logit(logit_psi);

  beta_psi ~ student_t(7.763,0, 1.566);
  beta_p ~ student_t(7.763,0, 1.566);

  for (i in 1:n_site) {
    if (n_survey[i] > 0) {
      if (any_seen[i]) {
        // site is occupied
        target += log_psi[i]
                  + bernoulli_logit_lpmf(y[start_idx[i]:end_idx[i]] |
                                         logit_p[start_idx[i]:end_idx[i]]);
      } else {
        // site may or may not be occupied
        target += log_sum_exp(
          log_psi[i] + bernoulli_logit_lpmf(y[start_idx[i]:end_idx[i]] |
                                            logit_p[start_idx[i]:end_idx[i]]),
          log1m_psi[i]
        );
      }
    }
  }
}
```

Occupancy with CAR on both components

```
functions{
/**
* Return log probability of a unit-scale proper conditional autoregressive
  * (CAR) prior with a sparse representation for the adjacency matrix
  *
  * @param phi Vector containing the parameters with a CAR prior
  * @param alpha Dependence (usually spatial) parameter for the CAR prior (real)
  * @param W_sparse Sparse representation of adjacency matrix (int array)
  * @param n Length of phi (int)
  * @param W_n Number of adjacent pairs (int)
  * @param D_sparse Number of neighbors for each location (vector)
  * @param lambda Eigenvalues of D^{-1/2}*W*D^{-1/2} (vector)
  *
  * @return Log probability density of CAR prior up to additive constant
  */
  real sparse_car_lpdf(vector phi, real alpha,
    int[,] W_sparse, vector D_sparse, vector lambda, int nobs, int W_n) {
      row_vector[nobs] phit_D; // phi' * D
      row_vector[nobs] phit_W; // phi' * W
      vector[nobs] ldet_terms;

      phit_D = (phi .* D_sparse)';
      phit_W = rep_row_vector(0, nobs);
      for (i in 1:W_n) {
        phit_W[W_sparse[i, 1]] += phi[W_sparse[i, 2]];
        phit_W[W_sparse[i, 2]] += phi[W_sparse[i, 1]];
      }

      for (i in 1:nobs) ldet_terms[i] = log1m(alpha * lambda[i]);
      return 0.5 * (sum(ldet_terms)
                    - (phit_D * phi - alpha * (phit_W * phi)));
  }
}
data {
  // site-level occupancy covariates
  int<lower = 1> n_site;
  int<lower = 1> m_psi;
  matrix[n_site, m_psi] X_tct;

  // survey-level detection covariates
  int<lower = 1> total_surveys;
  int<lower = 1> m_p;
  matrix[total_surveys, m_p] X_bg;

  // survey level information
  int<lower = 1, upper = n_site> site[total_surveys];
  int<lower = 0, upper = 1> y[total_surveys];
  int<lower = 0, upper = total_surveys> start_idx[n_site];
  int<lower = 0, upper = total_surveys> end_idx[n_site];

  // summary of whether species is known to be present at each site
  int<lower = 0, upper = 1> any_seen[n_site];
  // number of surveys at each site
  int<lower = 0> n_survey[n_site];

  //adjacency data
  matrix<lower = 0, upper = 1>[n_site, n_site] W_tct; //adjacency matrix tract
  int W_n_tct; //number of adjacent pairs
  matrix<lower = 0, upper = 1>[total_surveys, total_surveys] W_bg; //adjacency matrix bg
  int W_n_bg; //number of adjacent pairs bg
  //real<lower = 0, upper =1> alpha_occ;
  //real<lower = 0, upper =1> alpha_det;

}
transformed data {
  int W_sparse_occ[W_n_tct, 2];   // adjacency pairs
  int W_sparse_det[W_n_bg, 2];
  vector[n_site] D_sparse_occ;     // diagonal of D (number of neigbors for each site)
  vector[total_surveys] D_sparse_det;

  vector[n_site] lambda_occ;       // eigenvalues of invsqrtD * W * invsqrtD
  vector[total_surveys] lambda_det;
  { // generate sparse representation for W
  int counter_occ;
  counter_occ = 1;
  // loop over upper triangular part of W to identify neighbor pairs
    for (i in 1:(n_site - 1)) {
      for (j in (i + 1):n_site) {
        if (W_tct[i, j] == 1) {
          W_sparse_occ[counter_occ, 1] = i;
          W_sparse_occ[counter_occ, 2] = j;
          counter_occ += 1;
        }
      }
    }
  }
  for (i in 1:n_site) D_sparse_occ[i] = sum(W_tct[i]);
  {
    vector[n_site] invsqrtD_occ;
    for (i in 1:n_site) {
      invsqrtD_occ[i] = 1 / sqrt(D_sparse_occ[i]);
    }
    lambda_occ = eigenvalues_sym(quad_form(W_tct, diag_matrix(invsqrtD_occ)));
  }
  { // generate sparse representation for W_det
  int counter_det;
  counter_det = 1;
  // loop over upper triangular part of W to identify neighbor pairs
    for (i in 1:(total_surveys - 1)) {
      for (j in (i + 1):total_surveys) {
        if (W_bg[i, j] == 1) {
          W_sparse_det[counter_det, 1] = i;
          W_sparse_det[counter_det, 2] = j;
          counter_det += 1;
        }
      }
    }
  }
  for (i in 1:total_surveys) D_sparse_det[i] = sum(W_bg[i]);
  {
    vector[total_surveys] invsqrtD_det;
    for (i in 1:total_surveys) {
      invsqrtD_det[i] = 1 / sqrt(D_sparse_det[i]);
    }
    lambda_det = eigenvalues_sym(quad_form(W_bg, diag_matrix(invsqrtD_det)));
  }
}

parameters {
  vector[n_site] phi_occ;
  vector[total_surveys] phi_det;
  real<lower = 0, upper =0.999> alpha_occ;
  real<lower = 0, upper =0.999> alpha_det;
  real<lower = 0> sigma_occ;
  real<lower = 0> sigma_det;

  vector[m_psi] beta_psi;
  vector[m_p] beta_p;
}
transformed parameters {
  vector[total_surveys] logit_p = X_bg * beta_p + phi_det * sigma_det;
  vector[n_site] logit_psi = X_tct * beta_psi + phi_occ * sigma_occ;
}
model {
  vector[n_site] log_psi = log_inv_logit(logit_psi);
  vector[n_site] log1m_psi = log1m_inv_logit(logit_psi);

  phi_occ ~ sparse_car(alpha_occ, W_sparse_occ, D_sparse_occ, lambda_occ, n_site, W_n_tct);
  alpha_occ ~ beta(4,1);
  sigma_occ ~ normal(0, 1.5);
  phi_det ~ sparse_car(alpha_det, W_sparse_det, D_sparse_det, lambda_det, total_surveys, W_n_bg);
  alpha_det ~ beta(4,1);
  sigma_det ~ normal(0, 1.5);

  beta_psi ~ student_t(7.763,0, 1.566);
  beta_p ~ student_t(7.763,0, 1.566);

  for (i in 1:n_site) {
    if (n_survey[i] > 0) {
      if (any_seen[i]) {
        // site is occupied
        target += log_psi[i]
                  + bernoulli_logit_lpmf(y[start_idx[i]:end_idx[i]] |
                                         logit_p[start_idx[i]:end_idx[i]]);
      } else {
        // site may or may not be occupied
        target += log_sum_exp(
          log_psi[i] + bernoulli_logit_lpmf(y[start_idx[i]:end_idx[i]] |
                                         logit_p[start_idx[i]:end_idx[i]]),
          log1m_psi[i]
        );
      }
    }
  }
}
```