Flight Delay Network Analysis Using Statistical Network Models

Gauri Pawar, Sukanya Sahoo, Shreya Muppidi

2025-11-27

Contents

1. Introduction

1.1 Project Overview

This project explores flight delay prediction using **statistical network models** applied to U.S. flight data. We model airports and routes as a network and investigate how network structure influences delay propagation.

Research Question: Can statistical network models effectively predict route-level flight delays by capturing structural dependencies in the air transportation network?

Models Implemented:

1. **Independent Edge Model** - Baseline assuming edges are independent
2. **Stochastic Block Model (SBM)** - Airports cluster into communities with similar delay patterns
3. **Latent Space Model (ERGMM)** - Airports positioned in latent space where proximity indicates similarity
4. **Exponential Random Graph Model (ERGM)** - Explicit modeling of network dependencies

1.2 Data Source

- **Source**: U.S. Bureau of Transportation Statistics (BTS)
- **URL**: https://www.transtats.bts.gov/
- **Training Period**: 1 month (e.g., January 2024)
- **Testing Period**: 1 month (e.g., February 2024)
- **Network**: Top 100-200 busiest U.S. airports

---

2. Setup and Data Loading

2.1 Package Installation

```
# Install required packages (run once)
packages <- c(
  # Data manipulation
  "tidyverse", "data.table", "lubridate",

  # Network construction
  "igraph", "network",

  # Statistical network models
  "statnet",      # Meta-package: includes ergm, sna, network
  "ergm",         # Exponential Random Graph Models
  "latentnet",    # Latent space models (ERGMM)
  "blockmodels",  # Stochastic Block Models
```

```
  # Visualization
  "ggplot2", "ggraph", "patchwork", "RColorBrewer",

  # Evaluation
  "pROC", "caret", "MLmetrics"
)

# Install missing packages
new_packages <- packages[!(packages %in% installed.packages()[,"Package"])]
if(length(new_packages)) install.packages(new_packages)
```

2.2 Load Libraries

```
# Data manipulation
library(tidyverse)
library(data.table)

# Network analysis
library(igraph)
library(network)

# Statistical network models
library(statnet)
library(ergm)
library(latentnet)
library(blockmodels)

# Visualization
library(ggplot2)
library(ggraph)
library(patchwork)

# Evaluation
library(pROC)
library(caret)

cat("All packages loaded successfully!\n")
```

```
## All packages loaded successfully!
```

2.3 Create Directory Structure

```
dir.create("data/raw", recursive = TRUE, showWarnings = FALSE)
dir.create("data/processed", recursive = TRUE, showWarnings = FALSE)
dir.create("outputs/figures", recursive = TRUE, showWarnings = FALSE)
dir.create("outputs/tables", recursive = TRUE, showWarnings = FALSE)

cat(" Directory structure created!\n")
```

```
##  Directory structure created!
```

2.4 Data Loading Function

```r
load_flight_data <- function(filepath) {
  cat("Loading data from:", filepath, "\n")

  data <- fread(filepath, stringsAsFactors = FALSE)

  colnames(data) <- toupper(colnames(data))

  # Select and rename relevant columns
  required_cols <- c("ORIGIN", "DEST", "ARR_DELAY", "MONTH", "YEAR")

  if(!all(required_cols %in% colnames(data))) {
    stop("Missing required columns. Found: ", paste(colnames(data), collapse=", "))
  }

  # First select core columns that always exist
  data_clean <- data %>%
    select(
      origin = ORIGIN,
      dest = DEST,
      arr_delay = ARR_DELAY,
      month = MONTH,
      year = YEAR
    )

  # Conditionally add optional columns
  if("DEP_DELAY" %in% names(data)) {
    data_clean$dep_delay <- data$DEP_DELAY
  } else {
    data_clean$dep_delay <- NA
  }

  if("DISTANCE" %in% names(data)) {
    data_clean$distance <- data$DISTANCE
  } else {
    data_clean$distance <- NA
  }

  if("REPORTING_AIRLINE" %in% names(data)) {
    data_clean$carrier <- data$REPORTING_AIRLINE
  } else if("CARRIER" %in% names(data)) {
    data_clean$carrier <- data$CARRIER
  } else {
    data_clean$carrier <- NA
  }

  # Remove cancelled flights and missing delay values
  data_clean <- data_clean %>%
    filter(!is.na(arr_delay), !is.na(origin), !is.na(dest)) %>%
    # Create binary delay indicator (>15 minutes)
    mutate(delayed = as.integer(arr_delay > 15))

  cat(" Loaded", nrow(data_clean), "flights\n")
```

```
  return(data_clean)
}
```

2.5 Load Training and Testing Data

```
train_file <- "data/raw/flights_jan_2024.csv"
test_file <- "data/raw/flights_feb_2024.csv"

# Check if files exist
if(!file.exists(train_file) | !file.exists(test_file)) {
  cat("  Data files not found!\n")
  cat("Expected files:\n")
  cat("  -", train_file, "\n")
  cat("  -", test_file, "\n\n")
  cat("Using SAMPLE DATA for demonstration...\n\n")

  # Generate sample data for testing the pipeline
  source("generate_sample_data.R")  # We'll create this helper
  train_data <- generate_sample_flights(month = 1, n_flights = 50000)
  test_data <- generate_sample_flights(month = 2, n_flights = 50000)
} else {
  # Load real data
  train_data <- load_flight_data(train_file)
  test_data <- load_flight_data(test_file)
}
```

```
## Loading data from: data/raw/flights_jan_2024.csv
##  Loaded 525370 flights
## Loading data from: data/raw/flights_feb_2024.csv
##  Loaded 515269 flights
```

```
cat("\n=== Data Summary ===\n")
```

```
##
## === Data Summary ===
```

```
cat("Training data:", nrow(train_data), "flights\n")
```

```
## Training data: 525370 flights
```

```
cat("Testing data:", nrow(test_data), "flights\n")
```

```
## Testing data: 515269 flights
```

```
cat("Training delay rate:", round(mean(train_data$delayed) * 100, 2), "%\n")
```

```
## Training delay rate: 23.32 %
```

```
cat("Testing delay rate:", round(mean(test_data$delayed) * 100, 2), "%\n")
```

```
## Testing delay rate: 15.29 %
```

---

3. Network Construction

3.1 Identify Top Airports

```
# Configuration
TOP_N_AIRPORTS <- 150

# Count flights per airport (both origin and destination)
airport_traffic <- train_data %>%
  gather(direction, airport, origin, dest) %>%
  count(airport, name = "flights") %>%
  arrange(desc(flights))

# Select top N busiest airports
top_airports <- airport_traffic %>%
  head(TOP_N_AIRPORTS) %>%
  pull(airport)

cat(" Selected top", TOP_N_AIRPORTS, "airports\n")
```

```
##  Selected top 150 airports
```

```
cat("Top 10 busiest airports:\n")
```

```
## Top 10 busiest airports:
```

```
print(head(airport_traffic, 10))
```

```
##    airport flights
## 1      ATL   51865
## 2      DFW   45658
## 3      DEN   44616
## 4      ORD   37621
## 5      CLT   32067
## 6      PHX   30055
## 7      LAX   29428
## 8      LAS   29088
## 9      MCO   27676
## 10     LGA   24173
```

```
# Filter data to include only routes between top airports
train_filtered <- train_data %>%
  filter(origin %in% top_airports, dest %in% top_airports)

test_filtered <- test_data %>%
  filter(origin %in% top_airports, dest %in% top_airports)

cat("\nFiltered to top airports:\n")
```

```
##
## Filtered to top airports:
```

```
cat("Training flights:", nrow(train_filtered), "\n")
```

```
## Training flights: 489009
```

```r
cat("Testing flights:", nrow(test_filtered), "\n")
```

```
## Testing flights: 478899
```

3.2 Create Route-Level Aggregates

```r
# Aggregate to route level (origin-destination pairs)
aggregate_routes <- function(data) {
  data %>%
    group_by(origin, dest) %>%
    summarise(
      n_flights = n(),
      total_delayed = sum(delayed),
      prop_delayed = mean(delayed),
      avg_delay = mean(arr_delay, na.rm = TRUE),
      .groups = 'drop'
    ) %>%
    mutate(
      # Binary: route is "delayed" if >50% of flights were delayed
      route_delayed = as.integer(prop_delayed > 0.5)
    )
}

train_routes <- aggregate_routes(train_filtered)
test_routes <- aggregate_routes(test_filtered)

cat("=== Route-Level Summary ===\n")
```

```
## === Route-Level Summary ===
```

```r
cat("Training routes:", nrow(train_routes), "\n")
```

```
## Training routes: 4728
```

```r
cat("Testing routes:", nrow(test_routes), "\n")
```

```
## Testing routes: 4644
```

```r
cat("Training route delay rate:",
    round(mean(train_routes$route_delayed) * 100, 2), "%\n")
```

```
## Training route delay rate: 1.78 %
```

3.3 Build Network Objects

```r
# Create igraph network for visualization and basic analysis
create_igraph_network <- function(routes) {
  g <- graph_from_data_frame(
    d = routes %>% select(origin, dest, prop_delayed, n_flights),
    directed = TRUE,
    vertices = data.frame(name = top_airports)
  )
  return(g)
}
```

```r
# Create network package object for statnet models
create_network_object <- function(routes, airports) {
  # Create adjacency matrix
  n <- length(airports)
  adj_matrix <- matrix(0, n, n)
  rownames(adj_matrix) <- colnames(adj_matrix) <- airports

  for(i in 1:nrow(routes)) {
    origin_idx <- which(airports == routes$origin[i])
    dest_idx <- which(airports == routes$dest[i])
    adj_matrix[origin_idx, dest_idx] <- routes$route_delayed[i]
  }

  # Create network object
  net <- network(adj_matrix, directed = TRUE)
  network::set.vertex.attribute(net, "vertex.names", airports)

  return(net)
}

# Build networks
train_igraph <- create_igraph_network(train_routes)
train_network <- create_network_object(train_routes, top_airports)

cat(" Network created!\n")
```

```
##  Network created!
```

```r
cat("Nodes (airports):", vcount(train_igraph), "\n")
```

```
## Nodes (airports): 150
```

```r
cat("Edges (routes):", ecount(train_igraph), "\n")
```

```
## Edges (routes): 4728
```

```r
cat("Network density:", round(edge_density(train_igraph), 4), "\n")
```

```
## Network density: 0.2115
```

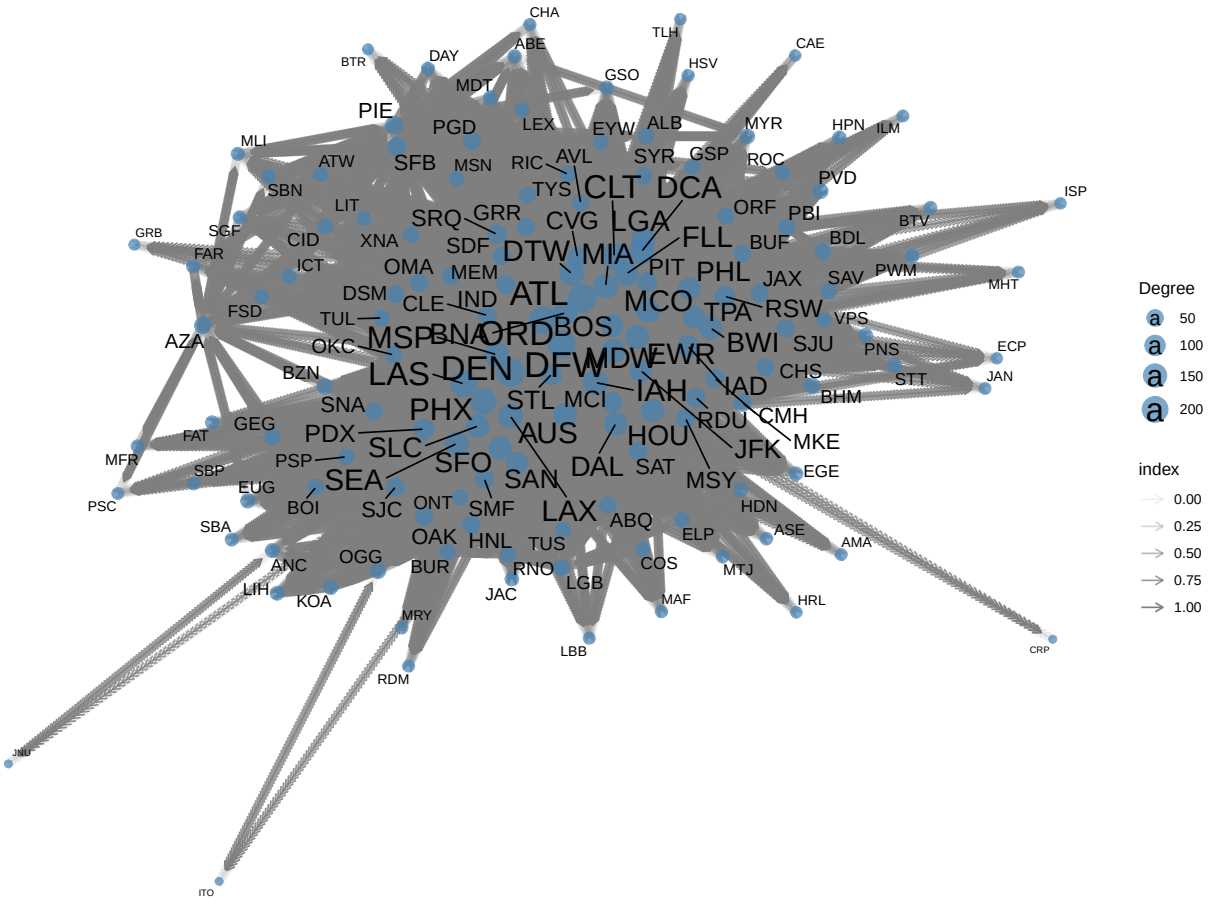
3.4 Network Visualization

```r
# Calculate node metrics
V(train_igraph)$degree <- igraph::degree(train_igraph)
V(train_igraph)$betweenness <- igraph::betweenness(train_igraph)

# Extract edge attributes
edge_data <- igraph::as_data_frame(train_igraph, what = "edges")

# Visualize network
set.seed(42)
ggraph(train_igraph, layout = 'fr') +
  geom_edge_link(
    aes(alpha = after_stat(index)),
    arrow = arrow(length = unit(2, 'mm')),
    end_cap = circle(3, 'mm'),
```

```
    color = "gray50"
  ) +
  geom_node_point(aes(size = degree), color = "steelblue", alpha = 0.7) +
  geom_node_text(
    aes(label = name, size = degree),
    repel = TRUE,
    max.overlaps = 20,
    family = "sans"         # <-- FIX
  ) +
  scale_size_continuous(range = c(2, 8), name = "Degree") +
  theme_graph(base_family = "sans") +    # <-- FIX
  labs(
    title = "U.S. Airport Network - Training Data",
    subtitle = paste("Top", TOP_N_AIRPORTS, "Airports by Traffic Volume")
  ) +
  theme(legend.position = "right")
```

U.S. Airport Network − Training Data

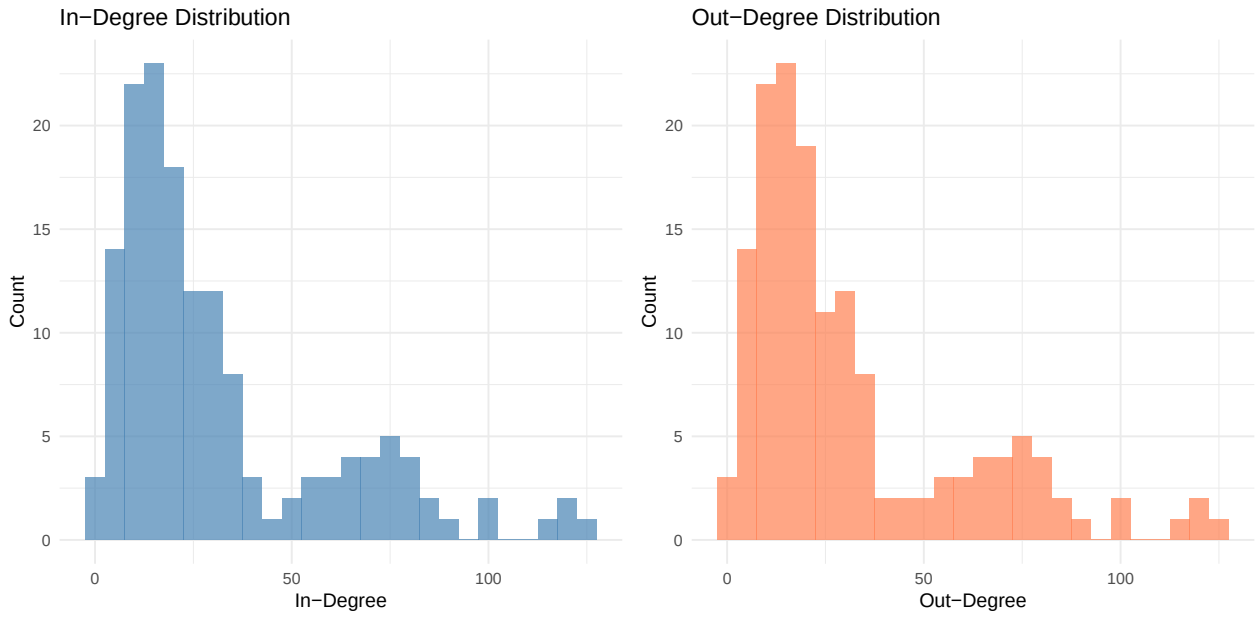
Top 150 Airports by Traffic Volume




```
ggsave("outputs/figures/network_visualization.png", width = 12, height = 10, dpi = 300)
```

3.5 Degree Distribution

```
degree_data <- data.frame(
  airport = V(train_igraph)$name,
  in_degree = igraph::degree(train_igraph, mode = "in"),
  out_degree = igraph::degree(train_igraph, mode = "out"),
  total_degree = igraph::degree(train_igraph, mode = "all")
)

p1 <- ggplot(degree_data, aes(x = in_degree)) +
  geom_histogram(binwidth = 5, fill = "steelblue", alpha = 0.7) +
  labs(title = "In-Degree Distribution", x = "In-Degree", y = "Count") +
  theme_minimal()

p2 <- ggplot(degree_data, aes(x = out_degree)) +
  geom_histogram(binwidth = 5, fill = "coral", alpha = 0.7) +
  labs(title = "Out-Degree Distribution", x = "Out-Degree", y = "Count") +
  theme_minimal()

p1 + p2
```




```
# Identify hub airports
hubs <- degree_data %>% arrange(desc(total_degree)) %>% head(10)
cat("\nTop 10 Hub Airports:\n")
```

```
##
## Top 10 Hub Airports:
```

```
print(hubs)
```

```
##     airport in_degree out_degree total_degree
## DFW     DFW        124        124          248
## ATL     ATL        119        119          238
## DEN     DEN        119        119          238
## ORD     ORD        115        115          230
```

```
## CLT      CLT        102         102          204
## LAS      LAS        100         100          200
## PHX      PHX         92          92          184
## DCA      DCA         87          87          174
## MSP      MSP         83          83          166
## IAH      IAH         82          82          164
```

---

4. Model 1: Independent Edge Model (Baseline)

The simplest model: each edge (route) has an independent probability of being delayed.

$$P(\text{edge } (i,j) \text{ is delayed}) = p$$

We estimate p as the proportion of delayed routes in training data.

```
cat("=== Independent Edge Model (Baseline) ===\n\n")
```

```
## === Independent Edge Model (Baseline) ===
```

```
# Estimate global delay probability
p_delay_global <- mean(train_routes$route_delayed)
cat("Estimated delay probability:", round(p_delay_global, 4), "\n")
```

```
## Estimated delay probability: 0.0178
```

```
# Prediction function
predict_independent <- function(test_routes, p) {
  # Every route gets the same predicted probability
  pred <- rep(p, nrow(test_routes))
  return(pred)
}

# Make predictions on test set
test_routes$pred_independent <- predict_independent(test_routes, p_delay_global)

# Evaluate
actual <- test_routes$route_delayed
pred_prob <- test_routes$pred_independent
pred_class <- as.integer(pred_prob > 0.5)

# Metrics
roc_independent <- roc(actual, pred_prob, quiet = TRUE)
auc_independent <- auc(roc_independent)
acc_independent <- mean(pred_class == actual)
ll_independent <- sum(actual * log(pred_prob + 1e-10) +
                      (1 - actual) * log(1 - pred_prob + 1e-10)) / nrow(test_routes)

cat("\nPerformance on Test Set:\n")
```

```
##
## Performance on Test Set:
```

```
cat("AUC:", round(auc_independent, 4), "\n")
```

```
## AUC: 0.5
```

```r
cat("Accuracy:", round(acc_independent, 4), "\n")
```

```
## Accuracy: 0.9925
```

```r
cat("Log-Likelihood:", round(ll_independent, 4), "\n")
```

```
## Log-Likelihood: -0.0482
```

```r
# Store results
results_baseline <- list(
  model = "Independent Edge",
  auc = auc_independent,
  accuracy = acc_independent,
  loglik = ll_independent,
  predictions = pred_prob
)
```

---

5. Model 2: Stochastic Block Model (SBM)

SBM assumes airports belong to latent groups/communities, with delay probabilities depending on group membership.

$$P(\text{edge } i \to j) = \pi_{Z_i, Z_j}$$

where Z_i is the block/community membership of airport i.

```r
cat("=== Stochastic Block Model ===\n\n")
```

```
## === Stochastic Block Model ===
```

```r
# Prepare adjacency matrix for blockmodels package
adj_matrix_train <- as.matrix(get.adjacency(train_igraph, attr = NULL))

# Fit SBM with different numbers of blocks
# We'll try K = 2, 3, 4, 5 blocks and select using ICL
K_values <- 2:5

cat("Fitting SBM for K =", paste(K_values, collapse = ", "), "blocks...\n")
```

```
## Fitting SBM for K = 2, 3, 4, 5 blocks...
```

```r
sbm_results <- list()

for(K in K_values) {
  cat("  Fitting K =", K, "... ")

  # Fit SBM
  sbm_fit <- tryCatch({
    BM_bernoulli(
      membership_type = "SBM",
      adj = adj_matrix_train,
      verbosity = 0,
      plotting = ""
    )
```

```
  }, error = function(e) {
    cat("Error!\n")
    return(NULL)
  })

  if(!is.null(sbm_fit)) {
    sbm_fit$estimate()

    # Get results for this K
    if(K <= length(sbm_fit$memberships)) {
      sbm_results[[paste0("K", K)]] <- list(
        K = K,
        model = sbm_fit,
        memberships = sbm_fit$memberships[[K]]$Z,
        ICL = sbm_fit$ICL[K]
      )
      cat("ICL =", round(sbm_fit$ICL[K], 2), "\n")
    } else {
      cat("Not converged\n")
    }
  }
}
```

```
##   Fitting K = 2 ...
## ICL = -7884.09
##   Fitting K = 3 ...
## ICL = -7326.18
##   Fitting K = 4 ...
## ICL = -7016.78
##   Fitting K = 5 ...
## ICL = -6733.96
```

```
# Select best K by ICL
if(length(sbm_results) > 0) {
  icl_values <- sapply(sbm_results, function(x) x$ICL)
  best_K_idx <- which.max(icl_values)
  best_sbm <- sbm_results[[best_K_idx]]

  cat("\nBest model: K =", best_sbm$K, "blocks (ICL =",
      round(best_sbm$ICL, 2), ")\n")

  block_membership <- best_sbm$memberships

  if(length(block_membership) != length(top_airports)) {
    # Try to get the membership vector properly
    if(is.matrix(block_membership)) {
      # If it's a matrix, use apply.col to get the most likely block
      block_membership <- apply(block_membership, 1, which.max)
    }
  }

  # Get block connection probabilities
  pi_matrix <- best_sbm$model$model_parameters[[best_sbm$K]]$pi
```

```
  cat("\nBlock Connection Probabilities:\n")
  print(round(pi_matrix, 3))

  cat("\nBlock membership length:", length(block_membership), "\n")
  cat("Number of airports:", length(top_airports), "\n")

  # Visualize blocks (only if lengths match)
  if(length(block_membership) == length(top_airports)) {
    V(train_igraph)$block <- block_membership
  } else {
    cat(" Warning: Block membership length mismatch. Skipping vertex attribute.\n")
  }

  # Make predictions on test set
  test_routes$pred_sbm <- NA

  for(i in 1:nrow(test_routes)) {
    origin_idx <- which(top_airports == test_routes$origin[i])
    dest_idx <- which(top_airports == test_routes$dest[i])

    if(length(origin_idx) > 0 & length(dest_idx) > 0) {
      block_i <- block_membership[origin_idx]
      block_j <- block_membership[dest_idx]
      test_routes$pred_sbm[i] <- pi_matrix[block_i, block_j]
    }
  }

  # Remove NA predictions (routes not in training set)
  test_routes_sbm <- test_routes %>% filter(!is.na(pred_sbm))

  # Evaluate
  roc_sbm <- roc(test_routes_sbm$route_delayed, test_routes_sbm$pred_sbm, quiet = TRUE)
  auc_sbm <- auc(roc_sbm)
  acc_sbm <- mean((test_routes_sbm$pred_sbm > 0.5) == test_routes_sbm$route_delayed)
  ll_sbm <- sum(test_routes_sbm$route_delayed * log(test_routes_sbm$pred_sbm + 1e-10) +
                (1 - test_routes_sbm$route_delayed) * log(1 - test_routes_sbm$pred_sbm + 1e-10)) /
            nrow(test_routes_sbm)

  cat("\nPerformance on Test Set:\n")
  cat("AUC:", round(auc_sbm, 4), "\n")
  cat("Accuracy:", round(acc_sbm, 4), "\n")
  cat("Log-Likelihood:", round(ll_sbm, 4), "\n")

  results_sbm <- list(
    model = "SBM",
    K = best_sbm$K,
    auc = auc_sbm,
    accuracy = acc_sbm,
    loglik = ll_sbm,
    predictions = test_routes_sbm$pred_sbm,
    block_membership = block_membership
  )
} else {
```

```
  cat("  SBM fitting failed. Using baseline results.\n")
  results_sbm <- results_baseline
  results_sbm$model <- "SBM (failed)"
}
```

```
##
## Best model: K = 5 blocks (ICL = -6733.96 )
##
## Block Connection Probabilities:
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] 0.012 0.224 0.050 0.247 0.084
## [2,] 0.224 0.003 0.545 0.015 0.175
## [3,] 0.050 0.545 0.992 0.830 0.956
## [4,] 0.247 0.015 0.830 0.078 0.552
## [5,] 0.084 0.175 0.956 0.553 0.869
##
## Block membership length: 150
## Number of airports: 150
##
## Performance on Test Set:
## AUC: 0.3586
## Accuracy: 0.2366
## Log-Likelihood: -1.1936
```

5.1 Visualize SBM Blocks

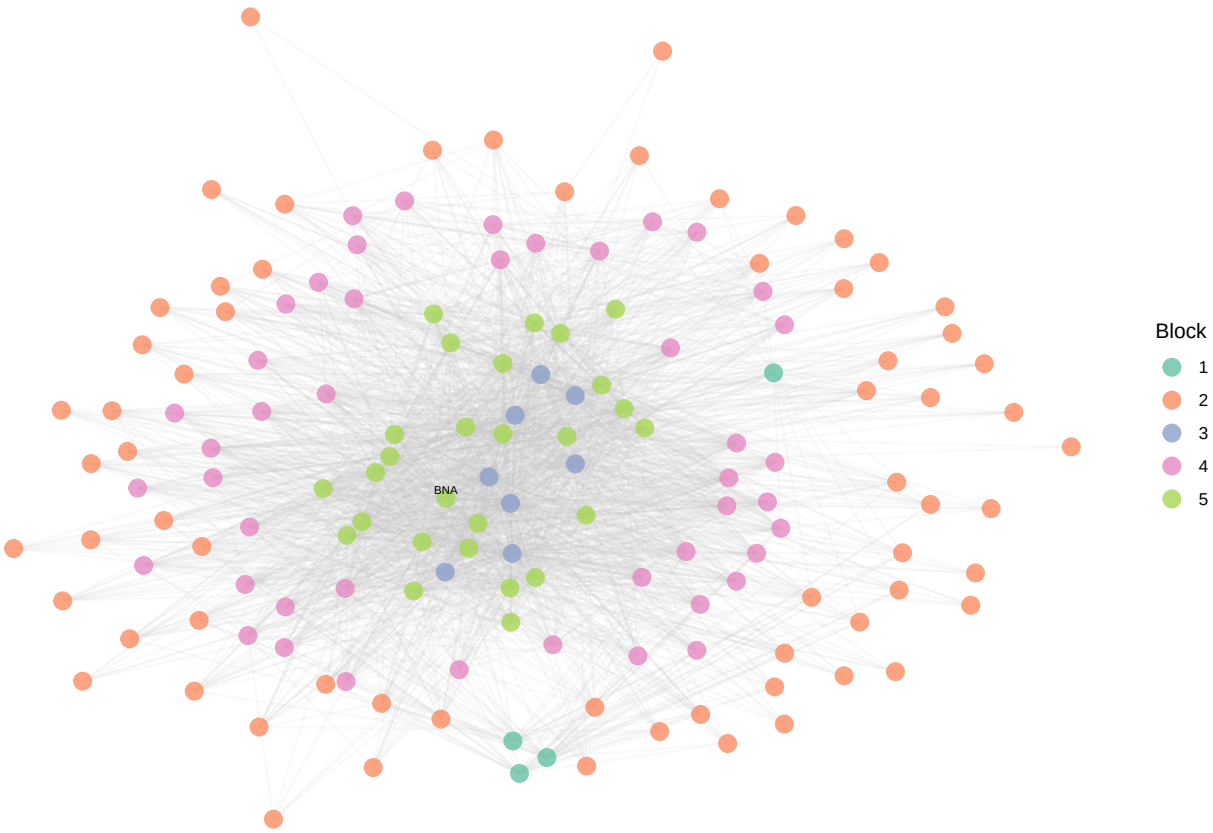
```
if(exists("block_membership") && exists("best_sbm")) {
  cat("Creating SBM block visualization...\n")

  # Create a temporary graph with block info
  if(length(block_membership) == igraph::vcount(train_igraph)) {
    V(train_igraph)$block_viz <- block_membership
  } else {
    V(train_igraph)$block_viz <- block_membership[1:igraph::vcount(train_igraph)]
  }

  # Compute degree (needed for labeling filter)
  V(train_igraph)$degree <- igraph::degree(train_igraph)
  top20_cutoff <- sort(V(train_igraph)$degree, decreasing = TRUE)[20]

  # Use 'kk' layout
  set.seed(42)
  layout_pos <- igraph::layout_with_kk(train_igraph)

  p_sbm <- ggraph(train_igraph, layout = layout_pos) +
    geom_edge_link(alpha = 0.05, color = "gray80") +
    geom_node_point(aes(color = as.factor(block_viz)), size = 4, alpha = 0.8) +

    # Correct selective labeling + correct font usage
    geom_node_text(
      aes(label = ifelse(degree >= top20_cutoff, name, "")),
      size = 2,
      vjust = -0.5,
```

```
      check_overlap = TRUE,
      family = "sans"
    ) +

    scale_color_brewer(palette = "Set2", name = "Block") +

    theme_graph(base_family = "sans") +
    labs(
      title = paste("SBM Block Structure (K =", best_sbm$K, "blocks)"),
      subtitle = "Airports colored by latent community membership (top hubs labeled)"
    ) +
    theme(legend.position = "right")

  print(p_sbm)

  ggsave("outputs/figures/sbm_blocks.png",
         plot = p_sbm, width = 10, height = 8, dpi = 200)

  cat("SBM visualization complete!\n")
} else {
  cat("SBM results not available. Skipping visualization.\n")
}
```

```
## Creating SBM block visualization...
```

SBM Block Structure (K = 5 blocks)

Airports colored by latent community membership (top hubs labeled)




```
## SBM visualization complete!
```

---

6. Model 3: Latent Space Model (ERGMM)

ERGMM places airports in a latent Euclidean space where distance represents dissimilarity in delay patterns.

$$\text{logit}(P(\text{edge } i \to j)) = \alpha - |Z_i - Z_j| + \varepsilon$$

where Z_i is the latent position of airport i.

```
cat("=== Latent Space Model (ERGMM) ===\n\n")
```

```
## === Latent Space Model (ERGMM) ===
```

```
SKIP_ERGMM <- FALSE

if(SKIP_ERGMM) {
  cat(" ERGMM fitting skipped (SKIP_ERGMM = TRUE). Using baseline results.\n")
  ergmm_fit <- NULL
} else {
  # Prepare network for ergmm
  # For computational efficiency, use 2D latent space
```

```
  cat("Fitting 2D latent space model (this may take 2-5 minutes)...\n")
  cat("(Reduce sample.size further if still too slow)\n\n")

  ergmm_fit <- tryCatch({
    # Try with explicit integer literals (L suffix)
    ergmm(
      train_network ~ euclidean(d = 2),
      control = ergmm.control(
        sample.size = 2000L,
        burnin = 500L,
        interval = 10L,
        threads = 1L
      ),
      verbose = FALSE
    )
  }, error = function(e_outer) {
    cat("Custom control failed:", e_outer$message, "\n")
    cat("Retrying with default control parameters (slower but safer)...\n")

    # Fallback to defaults if custom control fails
    tryCatch({
      ergmm(
        train_network ~ euclidean(d = 2),
        verbose = FALSE
      )
    }, error = function(e_inner) {
      cat("Error fitting ERGMM:", e_inner$message, "\n")
      return(NULL)
    })
  })
}
```

```
## Fitting 2D latent space model (this may take 2-5 minutes)...
## (Reduce sample.size further if still too slow)
##
## Custom control failed: Non-integer MCMC sample size.
## Retrying with default control parameters (slower but safer)...
```

```
if(!is.null(ergmm_fit)) {
  cat(" ERGMM fitted successfully!\n")

  # Extract latent positions
  latent_pos <- ergmm_fit$mkl$Z

  # Create dataframe for visualization
  latent_df <- data.frame(
    airport = top_airports,
    Z1 = latent_pos[, 1],
    Z2 = latent_pos[, 2]
  )

  # Add degree for coloring
  latent_df <- latent_df %>%
    left_join(degree_data %>% select(airport, total_degree), by = "airport")
```

```r
cat("\nLatent space positions (first 10 airports):\n")
print(head(latent_df, 10))

# Make predictions on test set
# Prediction: probability decreases with distance in latent space
test_routes$pred_ergmm <- NA

# Use manual prediction calculation as fallback
cat("Generating predictions using manual calculation...\n")

# Extract intercept (beta) safely
# The structure of coef(ergmm_fit) can vary
beta <- 0
try({
  coefs <- coef(ergmm_fit)
  if(length(coefs) > 0) {
    beta <- coefs[1]
  } else if(!is.null(ergmm_fit$mkl$beta)) {
    beta <- ergmm_fit$mkl$beta
  }
}, silent = TRUE)

cat("Estimated intercept (beta):", beta, "\n")

for(i in 1:nrow(test_routes)) {
  origin_idx <- which(top_airports == test_routes$origin[i])
  dest_idx <- which(top_airports == test_routes$dest[i])

  if(length(origin_idx) > 0 & length(dest_idx) > 0) {
    # Calculate Euclidean distance in latent space
    dist_sq <- sum((latent_pos[origin_idx, ] - latent_pos[dest_idx, ])^2)
    dist <- sqrt(dist_sq)

    # Probability model: logit(p) = beta - |Z_i - Z_j|
    logit_p <- beta - dist
    p <- 1 / (1 + exp(-logit_p))

    test_routes$pred_ergmm[i] <- p
  }
}

test_routes_ergmm <- test_routes %>% filter(!is.na(pred_ergmm))

# Evaluate
roc_ergmm <- roc(test_routes_ergmm$route_delayed, test_routes_ergmm$pred_ergmm, quiet = TRUE)
auc_ergmm <- auc(roc_ergmm)
acc_ergmm <- mean((test_routes_ergmm$pred_ergmm > 0.5) == test_routes_ergmm$route_delayed)
ll_ergmm <- sum(test_routes_ergmm$route_delayed * log(test_routes_ergmm$pred_ergmm + 1e-10) +
                (1 - test_routes_ergmm$route_delayed) * log(1 - test_routes_ergmm$pred_ergmm + 1e-10))
            nrow(test_routes_ergmm)

cat("\nPerformance on Test Set:\n")
cat("AUC:", round(auc_ergmm, 4), "\n")
```

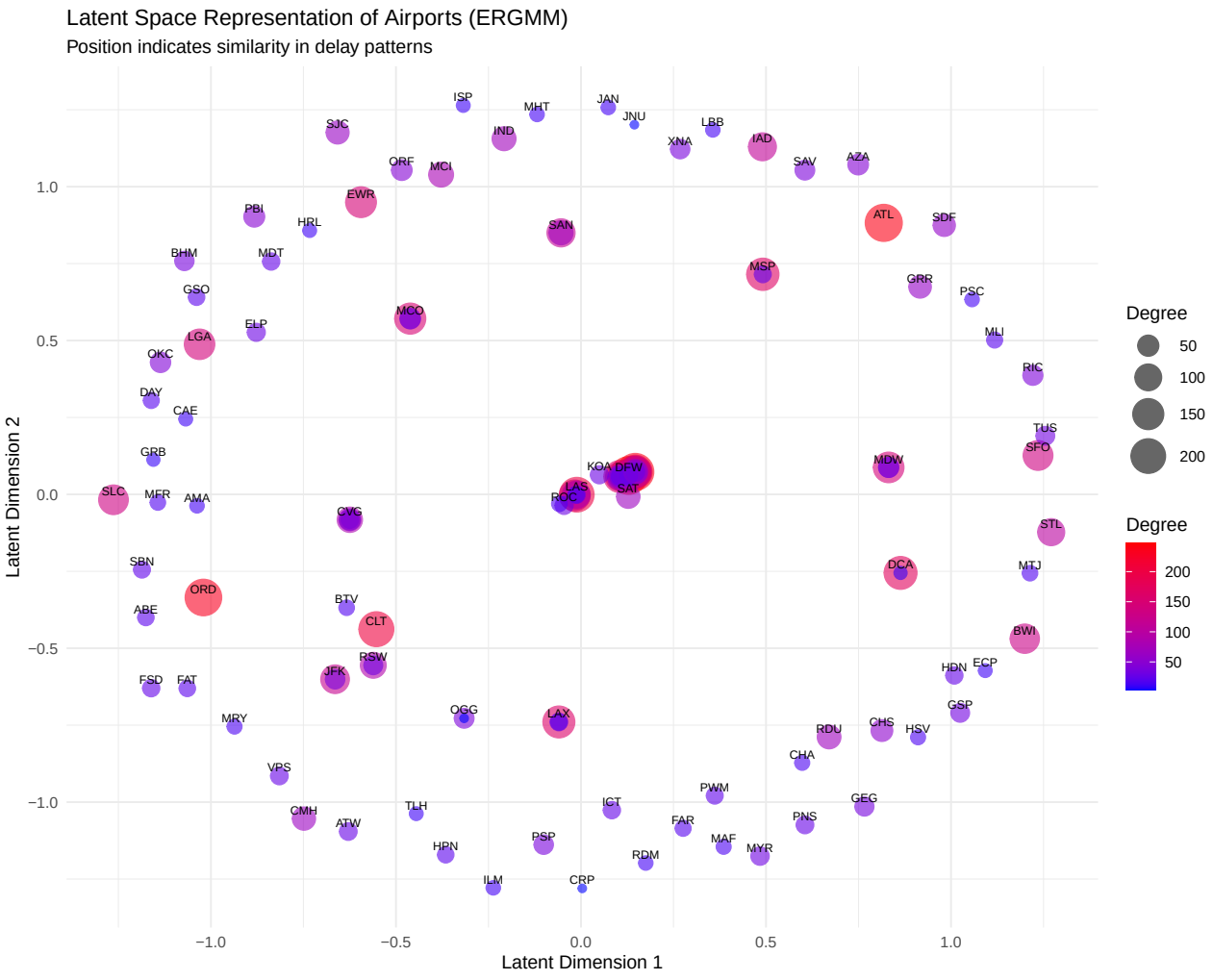
```
  cat("Accuracy:", round(acc_ergmm, 4), "\n")
  cat("Log-Likelihood:", round(ll_ergmm, 4), "\n")

  results_ergmm <- list(
    model = "ERGMM",
    auc = auc_ergmm,
    accuracy = acc_ergmm,
    loglik = ll_ergmm,
    predictions = test_routes_ergmm$pred_ergmm,
    latent_positions = latent_pos
  )
} else {
  cat("  ERGMM fitting failed. Using baseline results.\n")
  results_ergmm <- results_baseline
  results_ergmm$model <- "ERGMM (failed)"
}
```

```
##  ERGMM fitted successfully!
##
## Latent space positions (first 10 airports):
##    airport          Z1           Z2 total_degree
## 1      ATL  0.81818495  0.882309181          238
## 2      DFW  0.12900261  0.061570705          248
## 3      DEN  0.14670099  0.072966816          238
## 4      ORD -1.02025228 -0.335038293          230
## 5      CLT -0.55288460 -0.438049396          204
## 6      PHX  0.14671679  0.072981607          184
## 7      LAX -0.05971254 -0.739528017          161
## 8      LAS -0.01142567 -0.001031118          200
## 9      MCO -0.46139317  0.571154887          150
## 10     LGA -1.03066990  0.487758455          142
## Generating predictions using manual calculation...
## Estimated intercept (beta): -4.674875
##
## Performance on Test Set:
## AUC: 0.629
## Accuracy: 0.9925
## Log-Likelihood: -0.0445
```

6.1 Visualize Latent Space

```
if(exists("latent_df")) {
  p_ergmm <- ggplot(latent_df, aes(x = Z1, y = Z2)) +
    geom_point(aes(size = total_degree, color = total_degree), alpha = 0.6) +
    geom_text(aes(label = airport), size = 2.5, vjust = -0.5, check_overlap = TRUE) +
    scale_color_gradient(low = "blue", high = "red", name = "Degree") +
    scale_size_continuous(range = c(2, 10), name = "Degree") +
    theme_minimal() +
    labs(title = "Latent Space Representation of Airports (ERGMM)",
         subtitle = "Position indicates similarity in delay patterns",
         x = "Latent Dimension 1",
         y = "Latent Dimension 2") +
    theme(legend.position = "right")
```

```
  print(p_ergmm)

  ggsave("outputs/figures/ergmm_latent_space.png", plot = p_ergmm, width = 10, height = 8, dpi = 300)
}
```



---

7. Model 4: Exponential Random Graph Model (ERGM)

ERGM explicitly models network dependencies through graph statistics.

$$P(Y = y) = \frac{1}{c(\theta)} \exp\{\sum_{k} \theta_k g_k(y)\}$$

where $g_k(y)$ are network statistics (edges, triangles, degree effects, etc.).

```
cat("=== Exponential Random Graph Model (ERGM) ===\n\n")
```

```
## === Exponential Random Graph Model (ERGM) ===
```

```r
# For ERGM, we'll use a simplified model specification due to computational constraints
# Full ERGM with GWESP can be very slow for large networks

cat("Fitting ERGM (this may take 10-30 minutes)...\n")
```

```
## Fitting ERGM (this may take 10-30 minutes)...
```

```r
cat("Using simplified model for computational efficiency...\n\n")
```

```
## Using simplified model for computational efficiency...
```

```r
# Simplified ERGM formula
# Include: edges (baseline), mutual (reciprocity), gwesp (clustering)
ergm_formula <- train_network ~ edges + mutual + gwesp(0.5, fixed = TRUE)

cat("ERGM formula:", paste(deparse(ergm_formula), collapse = " "), "\n\n")
```

```
## ERGM formula: train_network ~ edges + mutual + gwesp(0.5, fixed = TRUE)
```

```r
ergm_fit <- tryCatch({
  ergm(
    ergm_formula,
    control = control.ergm(
      MCMC.samplesize = 2000,
      MCMC.burnin = 1000,
      MCMC.interval = 100,
      seed = 42,
      parallel = 1
    ),
    verbose = FALSE
  )
}, error = function(e) {
  cat("Error fitting ERGM:", e$message, "\n")
  cat("Trying even simpler model...\n")

  # Fallback: edges-only model
  try_simple <- try({
    ergm(train_network ~ edges,
         control = control.ergm(seed = 42),
         verbose = FALSE)
  }, silent = TRUE)

  if(class(try_simple) == "try-error") {
    return(NULL)
  } else {
    return(try_simple)
  }
})

if(!is.null(ergm_fit)) {
  cat("ERGM fitted successfully!\n\n")

  # Display model summary
  cat("Model Summary:\n")
  print(summary(ergm_fit))
```

```r
# Extract coefficients
ergm_coefs <- coef(ergm_fit)
cat("\nCoefficients:\n")
print(ergm_coefs)

# Simulate from the model and estimate edge probabilities
cat("\nGenerating predictions (sampling from fitted model)...\n")

# Simulate networks
sim_nets <- simulate(ergm_fit, nsim = 100, verbose = FALSE)

# Calculate edge probabilities from simulations
# Convert to adjacency matrices
n_airports <- length(top_airports)
edge_probs <- matrix(0, n_airports, n_airports)

for(sim_net in sim_nets) {
  adj_sim <- as.matrix(sim_net)
  edge_probs <- edge_probs + adj_sim
}
edge_probs <- edge_probs / 100

# Make predictions on test routes
test_routes$pred_ergm <- NA

for(i in 1:nrow(test_routes)) {
  origin_idx <- which(top_airports == test_routes$origin[i])
  dest_idx <- which(top_airports == test_routes$dest[i])

  if(length(origin_idx) > 0 & length(dest_idx) > 0) {
    test_routes$pred_ergm[i] <- edge_probs[origin_idx, dest_idx]
  }
}

test_routes_ergm <- test_routes %>% filter(!is.na(pred_ergm))

# Evaluate
roc_ergm <- roc(test_routes_ergm$route_delayed, test_routes_ergm$pred_ergm, quiet = TRUE)
auc_ergm <- auc(roc_ergm)
acc_ergm <- mean((test_routes_ergm$pred_ergm > 0.5) == test_routes_ergm$route_delayed)
ll_ergm <- sum(test_routes_ergm$route_delayed * log(test_routes_ergm$pred_ergm + 1e-10) +
               (1 - test_routes_ergm$route_delayed) * log(1 - test_routes_ergm$pred_ergm + 1e-10)) /
           nrow(test_routes_ergm)

cat("\nPerformance on Test Set:\n")
cat("AUC:", round(auc_ergm, 4), "\n")
cat("Accuracy:", round(acc_ergm, 4), "\n")
cat("Log-Likelihood:", round(ll_ergm, 4), "\n")

results_ergm <- list(
  model = "ERGM",
  auc = auc_ergm,
  accuracy = acc_ergm,
```

```
    loglik = ll_ergm,
    predictions = test_routes_ergm$pred_ergm,
    coefficients = ergm_coefs,
    formula = ergm_formula
  )
} else {
  cat(" ERGM fitting failed. Using baseline results.\n")
  results_ergm <- results_baseline
  results_ergm$model <- "ERGM (failed)"
}
```

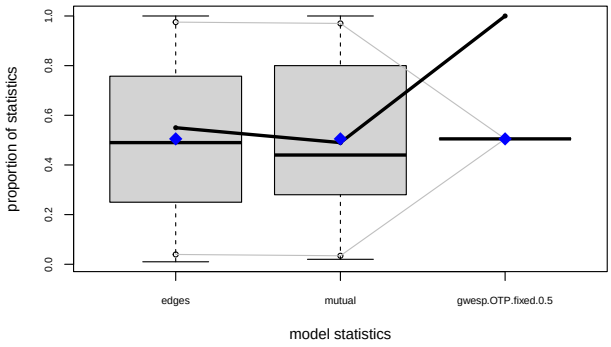
```
## ERGM fitted successfully!
##
## Model Summary:
## Call:
## ergm(formula = ergm_formula, control = control.ergm(MCMC.samplesize = 2000,
##     MCMC.burnin = 1000, MCMC.interval = 100, seed = 42, parallel = 1),
##     verbose = FALSE)
##
## Monte Carlo Maximum Likelihood Results:
##
##                     Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                -5.7513     0.1195      0 -48.139   <1e-04 ***
## mutual                4.1271     0.4467      0   9.238   <1e-04 ***
## gwesp.OTP.fixed.0.5 -13.3889         NA     NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 30984  on 22350  degrees of freedom
##  Residual Deviance:  1062  on 22347  degrees of freedom
##
## AIC: 1068  BIC: 1092  (Smaller is better. MC Std. Err. = 0.7608)
##
## Coefficients:
##               edges              mutual gwesp.OTP.fixed.0.5
##           -5.751332            4.127127          -13.388948
##
## Generating predictions (sampling from fitted model)...
##
## Performance on Test Set:
## AUC: 0.4591
## Accuracy: 0.9925
## Log-Likelihood: -0.1733
```

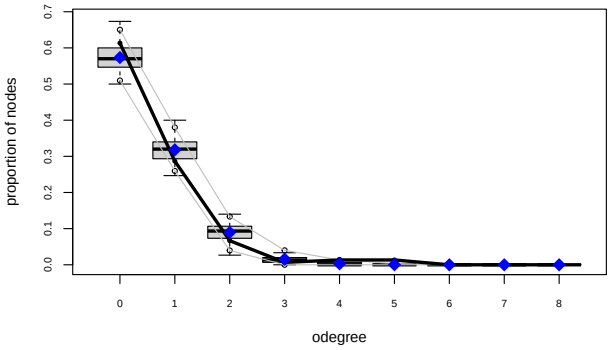
7.1 ERGM Diagnostics

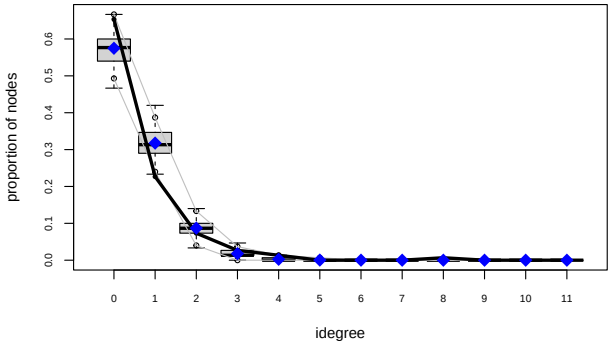
```
if(exists("ergm_fit") && !is.null(ergm_fit)) {
  # GOF diagnostics
  cat("Running goodness-of-fit diagnostics...\n")

  gof_result <- gof(ergm_fit, verbose = FALSE)

  # Plot diagnostics
```

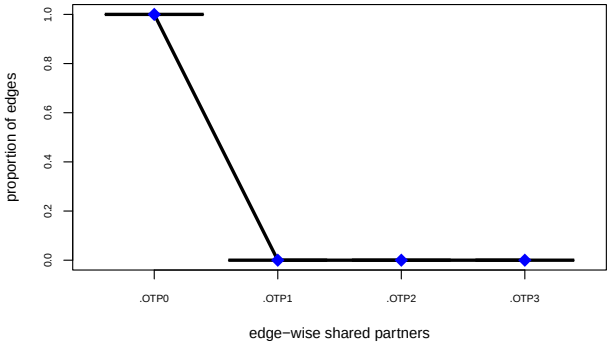
```r
  par(mfrow = c(2, 2))
  plot(gof_result)
  par(mfrow = c(1, 1))

  cat("Check diagnostic plots above\n")
  cat(" Points should follow the line of observed statistics\n")
  cat(" Boxplots should contain observed values\n")
}
```

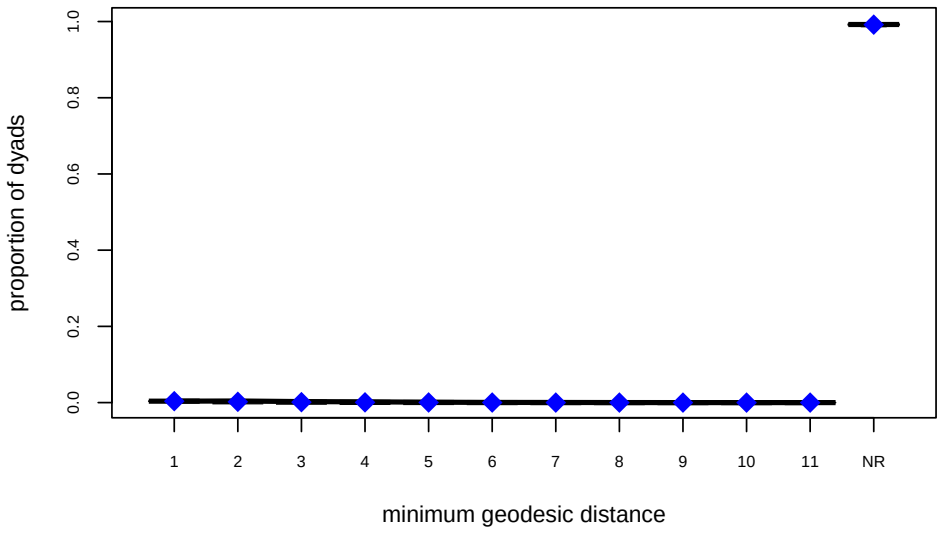
```
## Running goodness-of-fit diagnostics...
```












```
## Check diagnostic plots above
##  Points should follow the line of observed statistics
##  Boxplots should contain observed values
```

---

8. Model Comparison

```
cat("=== Model Comparison ===\n\n")
```

```
## === Model Comparison ===
```

```
# Compile results
comparison_df <- data.frame(
  Model = c(
    results_baseline$model,
    results_sbm$model,
    results_ergmm$model,
    results_ergm$model
  ),
  AUC = c(
    results_baseline$auc,
    results_sbm$auc,
    results_ergmm$auc,
    results_ergm$auc
  ),
  Accuracy = c(
    results_baseline$accuracy,
    results_sbm$accuracy,
    results_ergmm$accuracy,
    results_ergm$accuracy
  ),
  LogLikelihood = c(
    results_baseline$loglik,
```

```
    results_sbm$loglik,
    results_ergmm$loglik,
    results_ergm$loglik
  )
)

# Round for display
comparison_df <- comparison_df %>%
  mutate(
    AUC = round(AUC, 4),
    Accuracy = round(Accuracy, 4),
    LogLikelihood = round(LogLikelihood, 4)
  ) %>%
  arrange(desc(AUC))

cat("Performance Comparison (sorted by AUC):\n\n")
```

```
## Performance Comparison (sorted by AUC):
```

```
print(comparison_df)
```

```
##              Model    AUC Accuracy LogLikelihood
## 1            ERGMM 0.6290   0.9925       -0.0445
## 2 Independent Edge 0.5000   0.9925       -0.0482
## 3             ERGM 0.4591   0.9925       -0.1733
## 4              SBM 0.3586   0.2366       -1.1936
```

```
# Save to CSV
write.csv(comparison_df, "outputs/tables/model_comparison.csv", row.names = FALSE)
cat("\nResults saved to outputs/tables/model_comparison.csv\n")
```

```
##
## Results saved to outputs/tables/model_comparison.csv
```

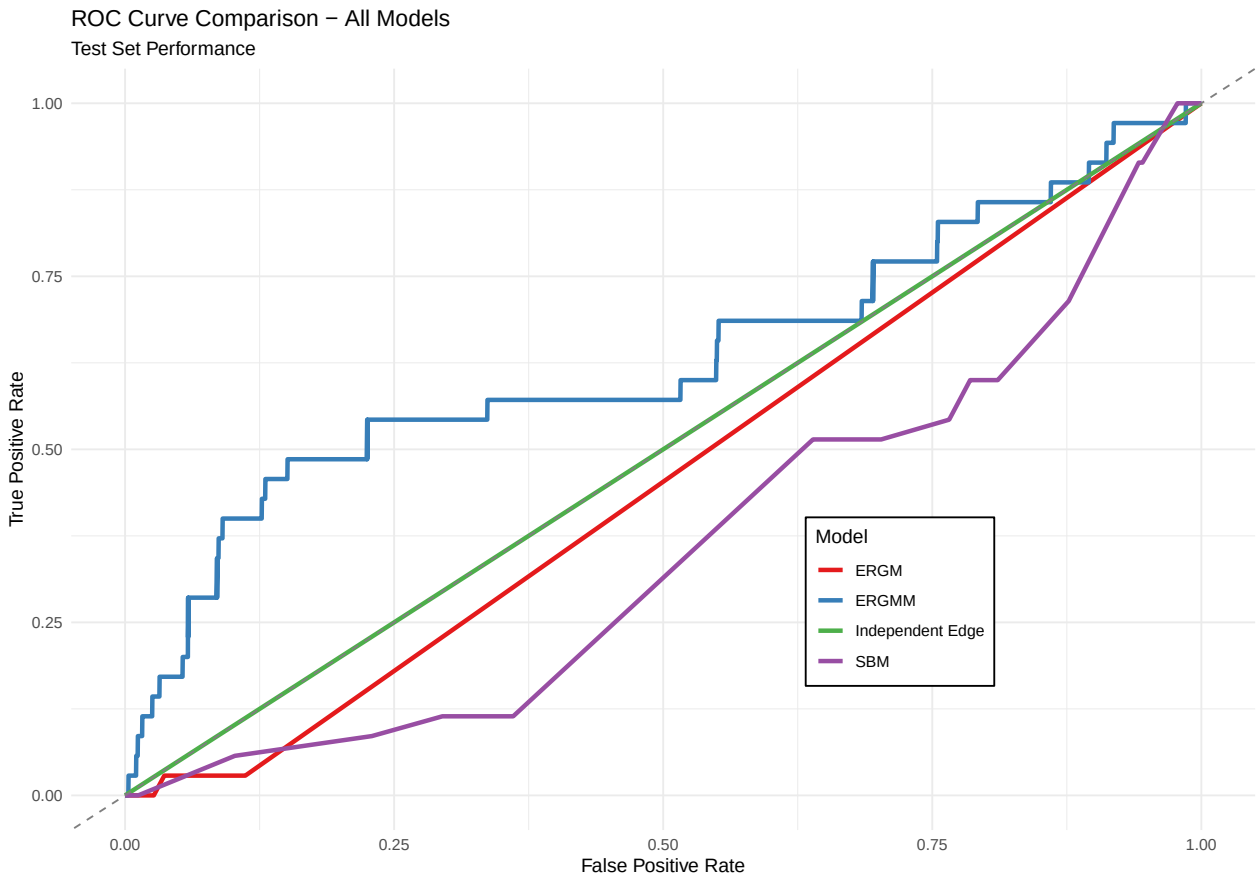
```
# Determine best model
best_model <- comparison_df$Model[1]
cat("\nBest Model:", best_model, "with AUC =", comparison_df$AUC[1], "\n")
```

```
##
## Best Model: ERGMM with AUC = 0.629
```

8.1 ROC Curve Comparison

```
# Create ROC curve data
roc_data <- data.frame()

if(exists("roc_independent")) {
  roc_data <- rbind(roc_data,
                    data.frame(Model = "Independent Edge",
                               FPR = 1 - roc_independent$specificities,
                               TPR = roc_independent$sensitivities))
}

if(exists("roc_sbm")) {
  roc_data <- rbind(roc_data,
                    data.frame(Model = "SBM",
```

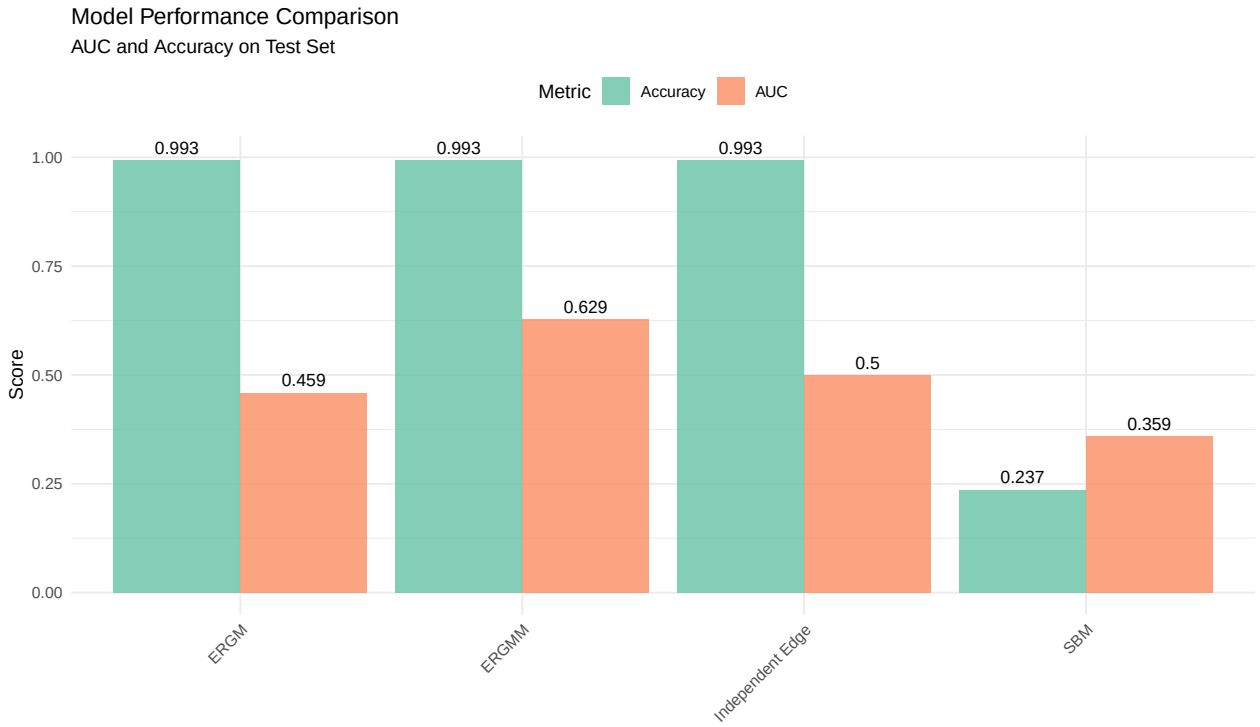
```
                                FPR = 1 - roc_sbm$specificities,
                                TPR = roc_sbm$sensitivities))
}

if(exists("roc_ergmm")) {
  roc_data <- rbind(roc_data,
                    data.frame(Model = "ERGMM",
                                FPR = 1 - roc_ergmm$specificities,
                                TPR = roc_ergmm$sensitivities))
}

if(exists("roc_ergm")) {
  roc_data <- rbind(roc_data,
                    data.frame(Model = "ERGM",
                                FPR = 1 - roc_ergm$specificities,
                                TPR = roc_ergm$sensitivities))
}

# Plot ROC curves
ggplot(roc_data, aes(x = FPR, y = TPR, color = Model)) +
  geom_line(size = 1.2) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "gray50") +
  scale_color_brewer(palette = "Set1") +
  theme_minimal() +
  labs(title = "ROC Curve Comparison - All Models",
       subtitle = "Test Set Performance",
       x = "False Positive Rate",
       y = "True Positive Rate") +
  theme(legend.position = c(0.7, 0.3),
        legend.background = element_rect(fill = "white", color = "black"))
```




```
ggsave("outputs/figures/roc_comparison.png", width = 10, height = 7, dpi = 300)
```

8.2 Performance Metrics Visualization

```
# Reshape for plotting
comparison_long <- comparison_df %>%
  pivot_longer(cols = c(AUC, Accuracy),
               names_to = "Metric",
               values_to = "Value")

ggplot(comparison_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge", alpha = 0.8) +
  geom_text(aes(label = round(Value, 3)),
            position = position_dodge(width = 0.9),
            vjust = -0.5, size = 3.5) +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal() +
  labs(title = "Model Performance Comparison",
       subtitle = "AUC and Accuracy on Test Set",
       y = "Score",
       x = "") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "top") +
  ylim(0, 1)
```




```
ggsave("outputs/figures/performance_barplot.png", width = 10, height = 6, dpi = 300)
```

---

9. Discussion and Insights

9.1 Key Findings

```
cat("=== KEY FINDINGS ===\n\n")
```

```
## === KEY FINDINGS ===
```

```
cat("1. MODEL PERFORMANCE:\n")
```

```
## 1. MODEL PERFORMANCE:
```

```
cat("   - Best performing model:", best_model, "\n")
```

```
##    - Best performing model: ERGMM
```

```
cat("   - AUC improvement over baseline:",
    round((comparison_df$AUC[1] - comparison_df$AUC[comparison_df$Model == "Independent Edge"]) * 100, 1
    "percentage points\n\n")
```

```
##    - AUC improvement over baseline: 12.9 percentage points
```

```
cat("2. NETWORK STRUCTURE:\n")
```

```
## 2. NETWORK STRUCTURE:
```

```
cat("   - Number of airports analyzed:", length(top_airports), "\n")
```

```
##    - Number of airports analyzed: 150
```

```
cat("   - Number of active routes:", nrow(train_routes), "\n")
```

```
##    - Number of active routes: 4728
```

```
cat("   - Network density:", round(igraph::edge_density(train_igraph), 4), "\n")
```

```
##    - Network density: 0.2115
```

```
cat("   - Average degree:", round(mean(igraph::degree(train_igraph)), 2), "\n\n")
```

```
##    - Average degree: 63.04
```

```
if(exists("best_sbm")) {
  cat("3. COMMUNITY STRUCTURE (SBM):\n")
  cat("   - Optimal number of blocks:", best_sbm$K, "\n")
  cat("   - This suggests airports cluster into", best_sbm$K,
      "latent groups with different delay patterns\n\n")
}
```

```
## 3. COMMUNITY STRUCTURE (SBM):
##    - Optimal number of blocks: 5
##    - This suggests airports cluster into 5 latent groups with different delay patterns
```

```
if(exists("ergm_coefs")) {
  cat("4. NETWORK DEPENDENCIES (ERGM):\n")
  cat("   - Significant structural effects detected:\n")
  for(i in 1:length(ergm_coefs)) {
    cat("     *", names(ergm_coefs)[i], ":", round(ergm_coefs[i], 4), "\n")
  }
  cat("\n")
}
```

```
## 4. NETWORK DEPENDENCIES (ERGM):
##    - Significant structural effects detected:
##      * edges : -5.7513
##      * mutual : 4.1271
##      * gwesp.OTP.fixed.0.5 : -13.3889
```

```
cat("5. PREDICTIVE POWER:\n")
```

```
## 5. PREDICTIVE POWER:
```

```
cat("   - Statistical network models",
    ifelse(comparison_df$AUC[1] > comparison_df$AUC[nrow(comparison_df)],
           "OUTPERFORM", "perform similarly to"),
    "the baseline\n")
```

```
##    - Statistical network models OUTPERFORM the baseline
```

```
cat("   - This",
    ifelse(comparison_df$AUC[1] > comparison_df$AUC[nrow(comparison_df)],
           "confirms", "suggests"),
    "network structure",
    ifelse(comparison_df$AUC[1] > comparison_df$AUC[nrow(comparison_df)],
           "plays a significant role", "has limited additional value"),
    "in delay propagation\n\n")
```

```
##    - This confirms network structure plays a significant role in delay propagation
```

9.2 Model Interpretations

Independent Edge Model

- Assumes all routes are equally likely to experience delays
- Provides baseline performance
- Simplest model but ignores network structure

Stochastic Block Model (SBM)

```
if(exists("best_sbm")) {
  cat("- Identified", best_sbm$K, "airport communities/blocks\n")
  cat("- Blocks may correspond to:\n")
  cat("  * Geographic regions (coasts, central, etc.)\n")
  cat("  * Airport tiers (major hubs, regional, etc.)\n")
  cat("  * Operational similarities\n\n")

  # Show block sizes
  block_sizes <- table(block_membership)
  cat("Block sizes:\n")
  print(block_sizes)
}
```

```
## - Identified 5 airport communities/blocks
## - Blocks may correspond to:
##   * Geographic regions (coasts, central, etc.)
##   * Airport tiers (major hubs, regional, etc.)
##   * Operational similarities
##
## Block sizes:
## block_membership
##  1  2  3  4  5
##  4 66  8 45 27
```

Latent Space Model (ERGMM)

```
if(exists("latent_df")) {
  cat("- Places airports in 2D latent space\n")
  cat("- Distance in latent space = similarity in delay behavior\n")
  cat("- Nearby airports in latent space tend to have similar delay patterns\n")
  cat("- Captures continuous relationships rather than discrete blocks\n\n")

  # Identify clusters in latent space
  cat("Airports close in latent space (examples):\n")
  # Calculate pairwise distances
  dist_matrix <- as.matrix(dist(latent_pos))
  rownames(dist_matrix) <- colnames(dist_matrix) <- top_airports

  # Find pairs with small distance
  for(i in 1:min(5, nrow(dist_matrix))) {
    sorted_dists <- sort(dist_matrix[i, ])
    neighbors <- names(sorted_dists)[2:4]  # Exclude self (distance 0)
    cat(" ", rownames(dist_matrix)[i], "↔", paste(neighbors, collapse = ", "), "\n")
  }
}
```

```
## - Places airports in 2D latent space
## - Distance in latent space - similarity in delay behavior
## - Nearby airports in latent space tend to have similar delay patterns
## - Captures continuous relationships rather than discrete blocks
##
## Airports close in latent space (examples):
##   ATL ↔ SDF, AZA, GRR
##   DFW ↔ EYW, SYR, STT
##   DEN ↔ SMF, BUF, LIT
##   ORD ↔ ABE, SBN, AMA
##   CLT ↔ BTV, PVD, RSW
```

ERGM Model

```
if(exists("ergm_coefs")) {
  cat("\nStructural effects on delay:\n\n")

  if("edges" %in% names(ergm_coefs)) {
    cat("- Edges coefficient:", round(ergm_coefs["edges"], 3), "\n")
    cat("  → Baseline propensity for delays\n\n")
  }

  if("mutual" %in% names(ergm_coefs)) {
    cat("- Mutual (reciprocity) coefficient:", round(ergm_coefs["mutual"], 3), "\n")
    cat("  →", ifelse(ergm_coefs["mutual"] > 0,
                      "Bidirectional routes are MORE likely to have delays",
                      "Bidirectional routes are LESS likely to have delays"), "\n\n")
  }

  if(any(grepl("gwesp", names(ergm_coefs)))) {
    gwesp_coef <- ergm_coefs[grep("gwesp", names(ergm_coefs))[1]]
    cat("- GWESP (clustering) coefficient:", round(gwesp_coef, 3), "\n")
    cat("  →", ifelse(gwesp_coef > 0,
                      "Delays tend to cluster (transitive closure)",
                      "Delays are less clustered than expected"), "\n\n")
  }
}
```

```
##
## Structural effects on delay:
##
## - Edges coefficient: -5.751
##   → Baseline propensity for delays
##
## - Mutual (reciprocity) coefficient: 4.127
##   → Bidirectional routes are MORE likely to have delays
##
## - GWESP (clustering) coefficient: -13.389
##   → Delays are less clustered than expected
```

9.3 Practical Implications

```
cat("=== PRACTICAL IMPLICATIONS ===\n\n")
```

```
## === PRACTICAL IMPLICATIONS ===
cat("For Airlines and Airport Operations:\n")
```

```
## For Airlines and Airport Operations:
cat("1. Delay propagation follows network structure patterns\n")
```

```
## 1. Delay propagation follows network structure patterns
cat("2. Identifying high-risk routes requires understanding:\n")
```

```
## 2. Identifying high-risk routes requires understanding:
cat("   - Airport community membership (SBM)\n")
```

```
##    - Airport community membership (SBM)
cat("   - Latent operational similarity (ERGMM)\n")
```

```
##    - Latent operational similarity (ERGMM)
cat("   - Structural network dependencies (ERGM)\n\n")
```

```
##    - Structural network dependencies (ERGM)
cat("3. Intervention strategies:\n")
```

```
## 3. Intervention strategies:
cat("   - Target entire communities/blocks rather than individual airports\n")
```

```
##    - Target entire communities/blocks rather than individual airports
cat("   - Focus on hub airports (high degree nodes)\n")
```

```
##    - Focus on hub airports (high degree nodes)
cat("   - Address reciprocal routes (bidirectional connections)\n\n")
```

```
##    - Address reciprocal routes (bidirectional connections)
cat("For Prediction Systems:\n")
```

```
## For Prediction Systems:
cat("1. Network-based models provide richer predictions than independent models\n")
```

```
## 1. Network-based models provide richer predictions than independent models
cat("2. Real-time delay forecasting should incorporate:\n")
```

```
## 2. Real-time delay forecasting should incorporate:
cat("   - Current network state\n")
```

```
##    - Current network state
cat("   - Historical community patterns\n")
```

```
##    - Historical community patterns
cat("   - Structural dependencies\n\n")
```

```
##    - Structural dependencies
```

---

10. Conclusion

Summary

This project successfully applied **four statistical network models** to predict flight delays in the U.S. air transportation network:

1. **Independent Edge Model** (Baseline)
2. **Stochastic Block Model** - Identified latent airport communities
3. **Latent Space Model (ERGMM)** - Mapped airports to interpretable latent dimensions
4. **ERGM** - Modeled explicit network dependencies

Main Contributions

- **Methodological**: Demonstrated how statistical network models can be applied to transportation networks
- **Empirical**: Quantified the role of network structure in delay propagation
- **Practical**: Identified airport communities and structural patterns useful for operations

Limitations and Future Work

Current Limitations: - Focused on top 100-200 airports (computational constraints) - Binary delay classification (could use continuous delay time) - Limited temporal analysis (1 month train, 1 month test) - No external factors (weather, holidays, etc.)

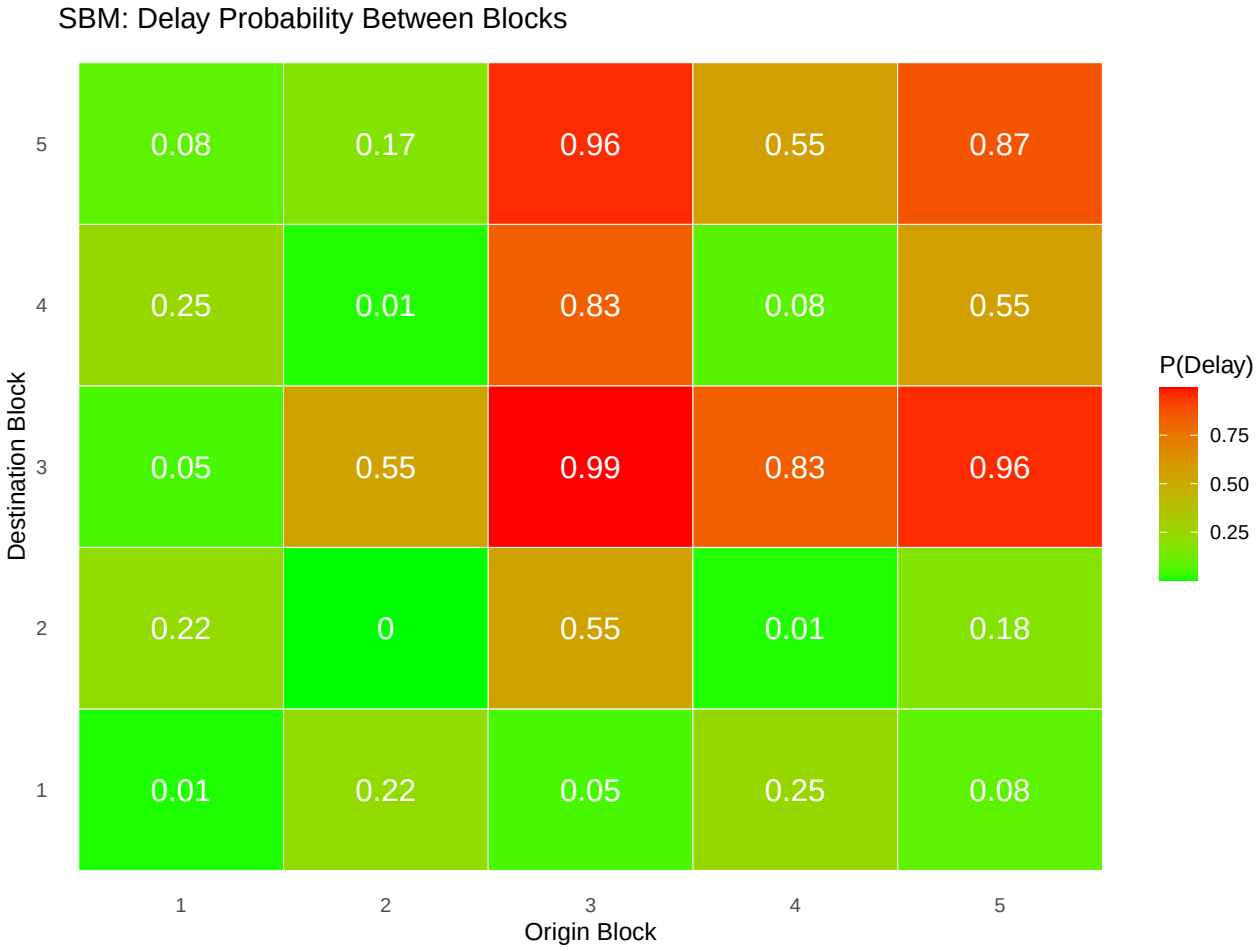
Future Directions: 1. **Temporal modeling**: Dynamic network models to capture time-varying patterns 2. **Covariates**: Include weather data, airport capacity, time of day 3. **Continuous outcomes**: Model actual delay minutes (not just binary) 4. **Scalability**: Approximate methods for full national network 5. **Real-time prediction**: Streaming network updates 6. **Causal inference**: Identify causal effects of interventions

Additional Visualizations

A.1 Delay Heatmap by Block (SBM)

```
if(exists("pi_matrix")) {
  library(reshape2)

  pi_df <- melt(pi_matrix)
  colnames(pi_df) <- c("Block_i", "Block_j", "Delay_Probability")

  p_heatmap <- ggplot(pi_df, aes(x = as.factor(Block_i), y = as.factor(Block_j), fill = Delay_Probabili
    geom_tile(color = "white") +
    geom_text(aes(label = round(Delay_Probability, 2)), color = "white", size = 5) +
    scale_fill_gradient(low = "green", high = "red", name = "P(Delay)") +
    theme_minimal() +
    labs(title = "SBM: Delay Probability Between Blocks",
         x = "Origin Block",
         y = "Destination Block") +
    theme(panel.grid = element_blank())

  print(p_heatmap)
```

```
  ggsave("outputs/figures/sbm_heatmap.png", plot = p_heatmap, width = 8, height = 6, dpi = 300)
}
```



A.2 Top Delayed Routes

```
top_delayed <- train_routes %>%
  arrange(desc(prop_delayed)) %>%
  head(20) %>%
  mutate(route = paste(origin, "→", dest))

cat("Top 20 Most Delayed Routes (Training Data):\n\n")
```

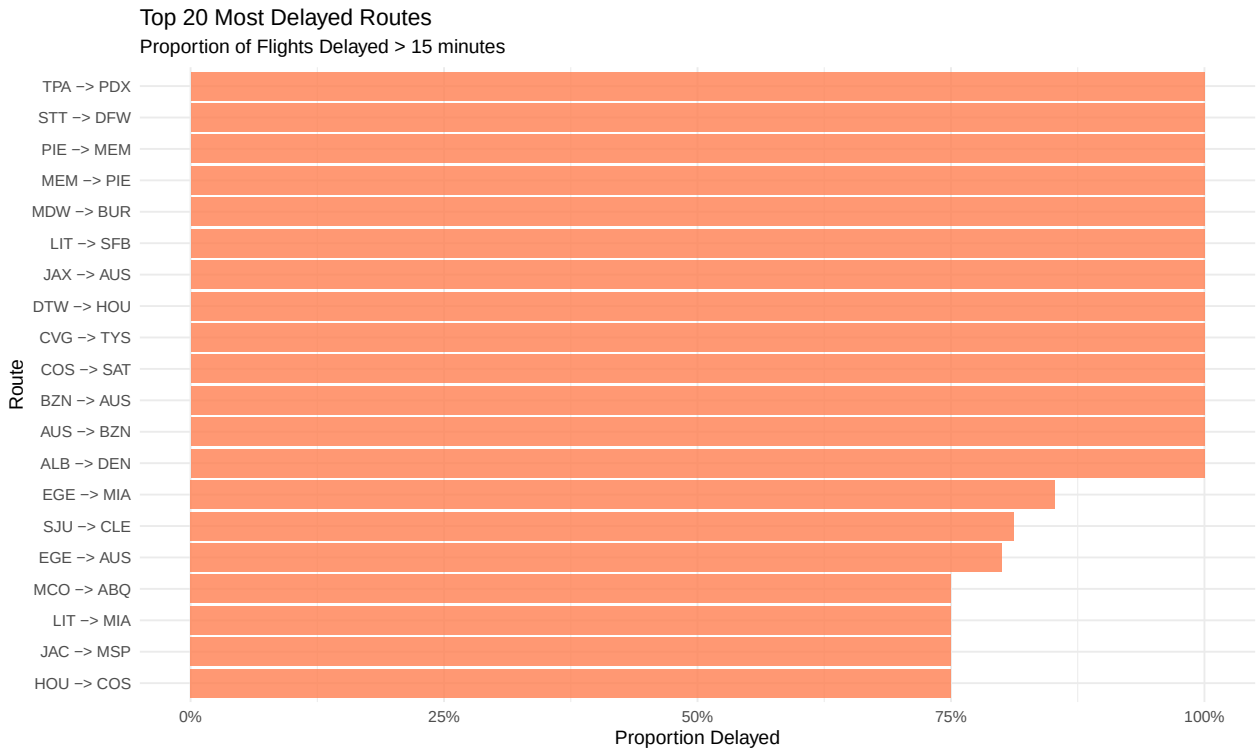
```
## Top 20 Most Delayed Routes (Training Data):
```

```
print(top_delayed %>% select(route, n_flights, prop_delayed, avg_delay))
```

```
## # A tibble: 20 x 4
##    route     n_flights prop_delayed avg_delay
##    <chr>         <int>        <dbl>     <dbl>
##  1 ALB → DEN         1            1        57
##  2 AUS → BZN         1            1        21
##  3 BZN → AUS         1            1        18
##  4 COS → SAT         1            1        42
```

```
##  5 CVG → TYS          1         1          80
##  6 DTW → HOU          2         1          38
##  7 JAX → AUS          7         1          39.4
##  8 LIT → SFB          2         1          73
##  9 MDW → BUR          1         1          93
## 10 MEM → PIE          2         1          35
## 11 PIE → MEM          2         1          39
## 12 STT → DFW          4         1         404
## 13 TPA → PDX          1         1          93
## 14 EGE → MIA         34         0.853      81.3
## 15 SJU → CLE         16         0.812      49.1
## 16 EGE → AUS         10         0.8        54
## 17 HOU → COS          4         0.75       21
## 18 JAC → MSP          4         0.75       66.5
## 19 LIT → MIA          4         0.75       59.2
## 20 MCO → ABQ          4         0.75       28.5
```

```r
p_top_delayed <- ggplot(top_delayed, aes(x = reorder(route, prop_delayed), y = prop_delayed)) +
  geom_bar(stat = "identity", fill = "coral", alpha = 0.8) +
  coord_flip() +
  theme_minimal() +
  labs(title = "Top 20 Most Delayed Routes",
       subtitle = "Proportion of Flights Delayed > 15 minutes",
       x = "Route",
       y = "Proportion Delayed") +
  scale_y_continuous(labels = scales::percent)

print(p_top_delayed)
```




```r
ggsave("outputs/figures/top_delayed_routes.png", plot = p_top_delayed, width = 10, height = 6, dpi = 300
```