Relazione per Advanced Data Science

Analisi di una rete

Pietro Marcatti 164717

2025-01-27

Analisi del tabellone di gioco di “Lettere da Whitechapel”

Figure 1: Il tabellone di gioco.

Prefazione

L’obiettivo di questo documento è riportare e presentare il processo di analisi condotto sul grafo contenuto nel tabellone del gioco da tavolo “Lettere da Whitechapel”. L’analisi vuole utilizzare un approccio analitico per cercare di rispondere quantitativamente a domande di ricerca relativamente alle strategie di gioco che le

parti possono utilizzare. Data che l'analisi si basa su e utilizza la conoscenza delle regole di gioco, verranno qui brevemente riassunte le regole fondamentali la cui conoscenza è necessaria per apprezzare l'analisi svolta.

Regole

Il gioco vede due parti: "Jack lo Squartatore" e la "Polizia", scontrarsi in un gioco investigativo a turni suddiviso in 4 macro momementi che corrispondono a quattro notti nella Londra del 1888. Una volta sola per partita, all'inizio della prima notte, Jack deve scegliere una nodo bianco come rifugio: ogni notte dovrà farvi ritorno per farla terminare e passare alla fase di gioco successiva. Alcune regole di gioco sono state rilassate e quindi ciascuna delle quattro notti vede le seguenti fasi:

- Jack decide in quale dei nodi rossi (luoghi del crimine) vuole commettere il prossimo omicidio, senza rivelarlo.
- La Polizia posiziona le proprie (5) pedine di gioco su altrettanti nodi neri del tabellone.
- Jack rivela in quale nodo ha commesso l'omicidio e dal quale inizierà la sua fuga.
- Jack, segretamente, sceglie in quale nodo bianco adiacente alla sua posizione muoversi.
- La Polizia muove ognuna delle sue pedine in nodi neri a distanza massima 2 da quello corrente e sceglie, per ciascuna, se:
  - Chiedere indizi: interpella Jack per chiedere se, nella sua fuga, è passato per qualcuno dei nodi bianchi adiacenti a quello dove si trova la pedina
  - Effettuare un arrest: interpella Jack proclamando un singolo nodo bianco adiacente a quello dove si trova la pedina, se Jack si trova lì è arrestato e la Polizia vince il gioco, altrimenti Jack non deve rivelare nessuna informazione.

Jack ha a sua disposizione due metodi di movimento speciali ma limitati:

- Carrozza: consente a Jack di compiere due passi nello stesso turno. Inoltre, gli consente di oltrepassare una pedina della Polizia che si frappone tra due dei nodi bianchi coinvolti in questo movimento.
- Vicolo: consente a Jack di trasportare la sua pedina in uno qualsiasi dei nodi bianchi che si trova sul perimetro dello stesso isolato a cui appartiene il nodo in cui si trova.

Il numero di movimenti speciali a disposizione di Jack varia con ogni notte e sono qui riportati:

- Notte 1: 3 carrozze e 2 vicoli
- Notte 2: 2 carrozze e 2 vicoli
- Notte 3: 2 carrozze e 1 vicolo
- Notte 4: 1 carrozza e 1 vicolo

Analisi Esplorativa

Prima di cercare di rispondere a domande relative al dominio di gioco è importante studiare le caratteristiche del dataset. Questo per comprendere con che tipo di dati si lavora, come sono distribuiti e come si relazionano. L'unica fonte dei dati è il tabellone di gioco dal quale sono stati manualmente estratti i seguenti dataset:

- nodes.csv, con colonne:

  - name, identificativo univoco intero: da 1 a 195 inclusi per i nodi bianchi, da 201 a 434 per i nodi neri
  - type, codifica ridondante della tipologia di nodo: 0 per nodi bianchi, 1 per nodi neri
  - crime_scene: 1 per i nodi bianchi che sono scena del crimine, 0 altrimenti
  - hideout: 1 per i nodi bianchi che sono candidati rifugio per Jack, 0 altrimenti
  - yellow: 1 per i nodi neri che nel gioco senza rilassamento delle regole corrispondono al punto di partenza delle pedine Polizia, 0 altrimenti

- base_edges.csv, con colonne:
  - from, identificativo del primo estremo dell'arco
  - to, identificativo del secondo estremo dell'arco
  - type, codifica della tipologia di arco:
    * 0: arco bianco-bianco semplice
    * 1: arco nero-nero semplice
    * 2: arco nero-bianco
    * 3: arco bianco-bianco per i vicoli
  - step, per gli archi in questo dataset è sempre 0, diverrà rilevante con l'aggiunta di altri archi in seguito
  - id, identificativo univoco intero crescente per gli archi

```
library(igraph)
library(tidyverse)
library(tidygraph)
library(ggraph)
library(gridExtra)
library(tcltk)
library(abind)
library(corrplot)
# Lettura dei file CSV
nodes <- read_csv("datasets/nodes.csv", show_col_types = FALSE)
head(nodes)
```

```
## # A tibble: 6 x 5
##    name  type crime_scene hideout yellow
##   <dbl> <dbl>       <dbl>   <dbl>  <dbl>
## 1     1     0           0       1      0
## 2     2     0           0       1      0
## 3     3     0           1       0      0
## 4     4     0           0       1      0
## 5     5     0           0       1      0
## 6     6     0           0       1      0
```

```
base_edges <- read_csv("datasets/base_edges.csv",show_col_types = FALSE)
head(base_edges)
```

```
## # A tibble: 6 x 5
##    from    to  type  step    id
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1    24     0     0     1
```

```
## 2     1     6     0     0     2
## 3     1     7     0     0     3
## 4     1    26     0     0     4
## 5     1     2     0     0     5
## 6     1     8     0     0     6
```

```r
# Creazione del grafo
g <- graph_from_data_frame(vertices= nodes, d = base_edges, directed = FALSE)
```

Topologia del grafo

Il grafo del tabellone di gioco può essere suddiviso semanticamente in 3 sottografi sulla base di quale tipo di archi si considera:

- Archi di tipo 0: mostra il grafo nel quale si muove Jack con soli movimenti semplici.
- Archi di tipo 1: è il grafo nel quale si muove la Polizia.
- Archi di tipo 2: è il grafo dell'interazione tra i nodi bianchi e quelli neri per la ricerca di indizi e l'emanazione di arresti.

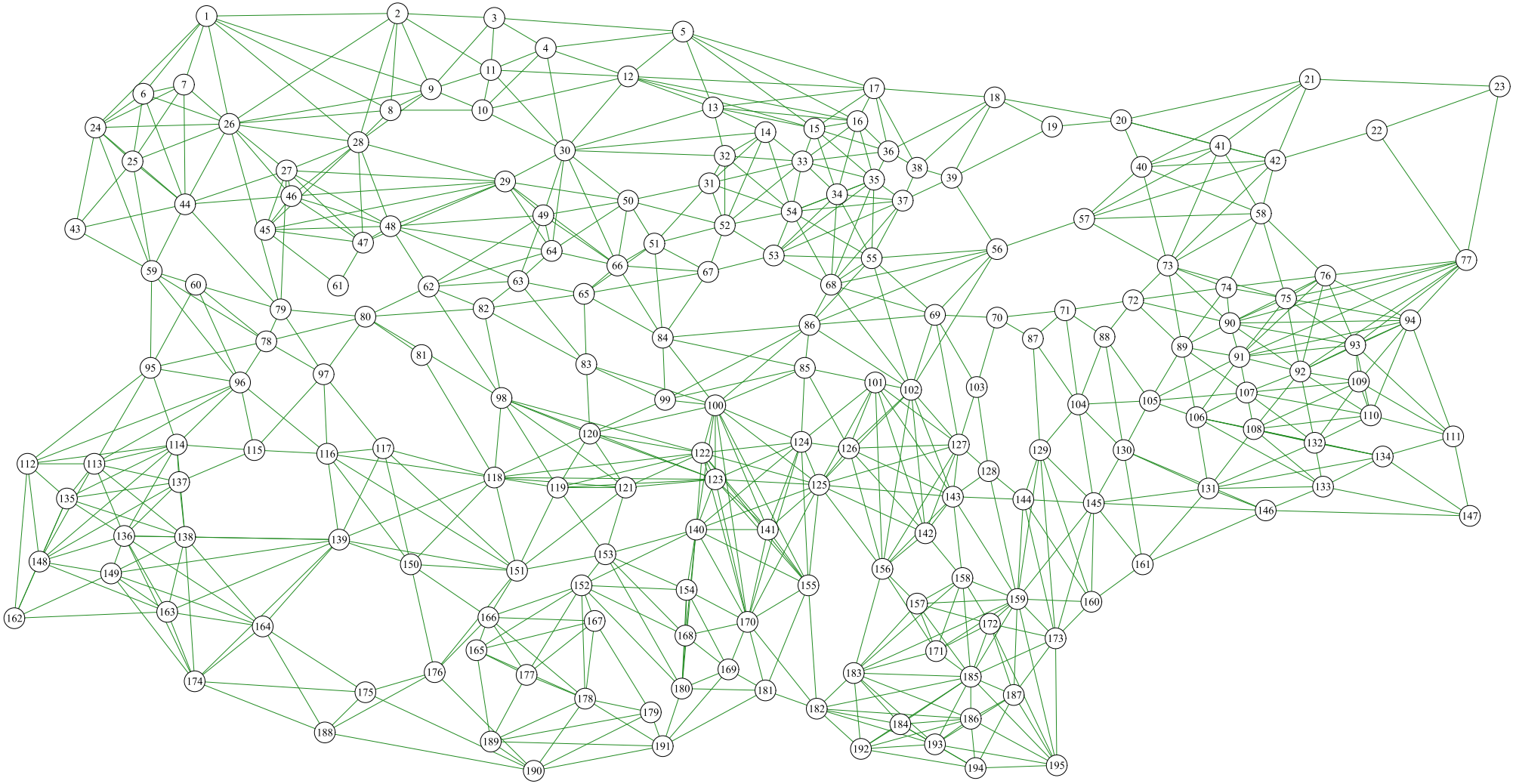


Figure 2: Il grafo dei nodi bianchi con archi di tipo 0

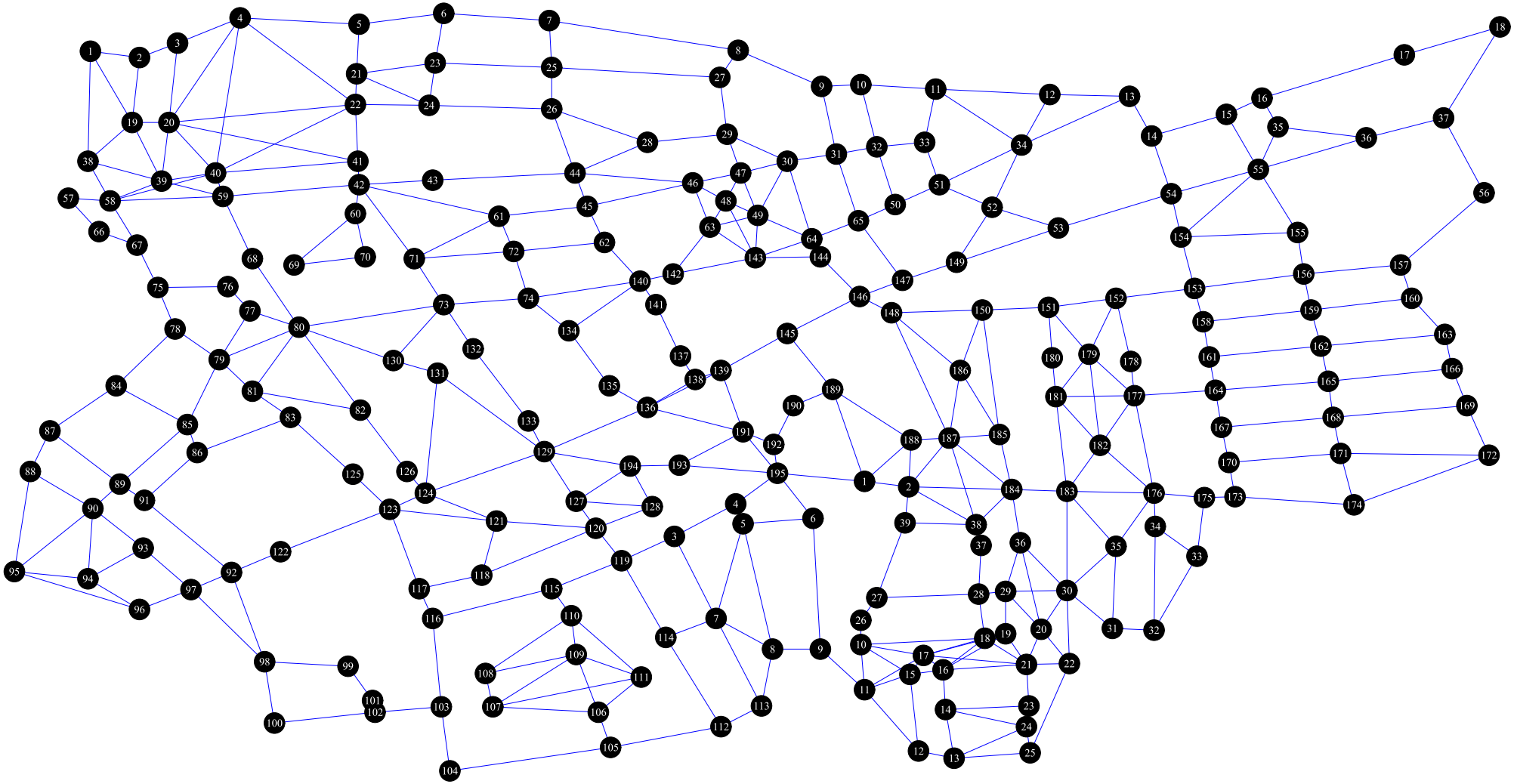


Figure 3: Il grafo dei nodi neri con archi di tipo 1

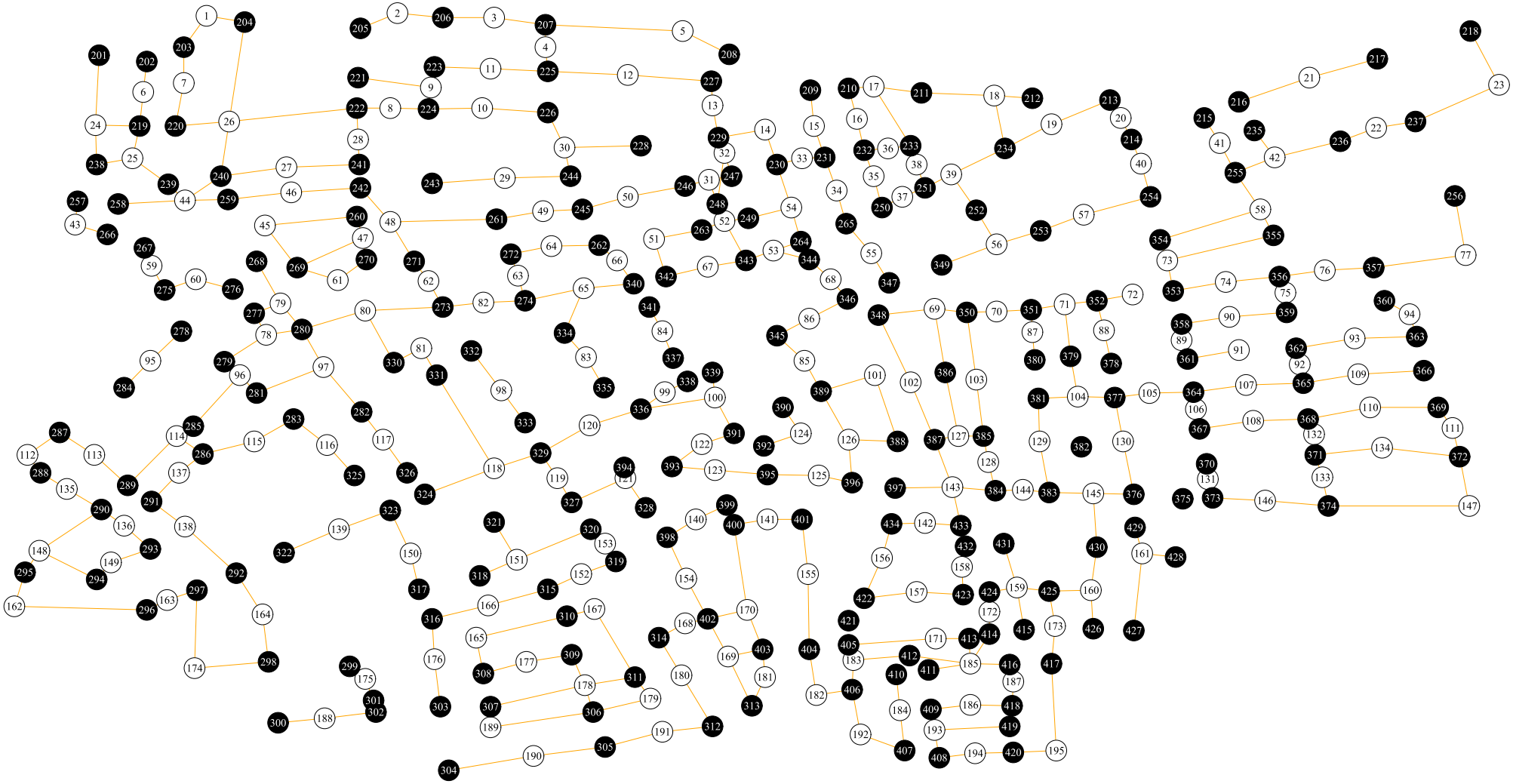


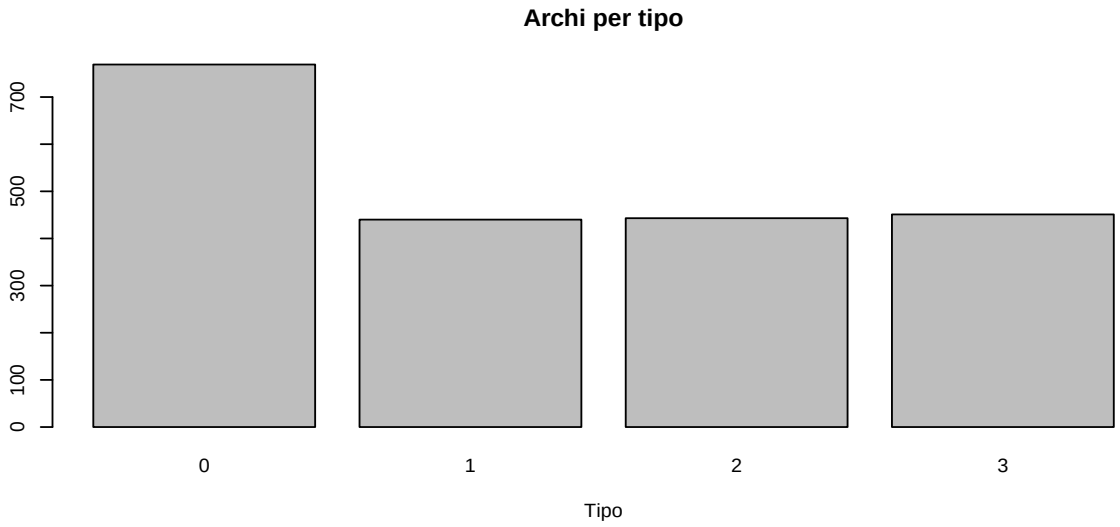
Figure 4: Il grafo di pattugliamento con archi di tipo 2

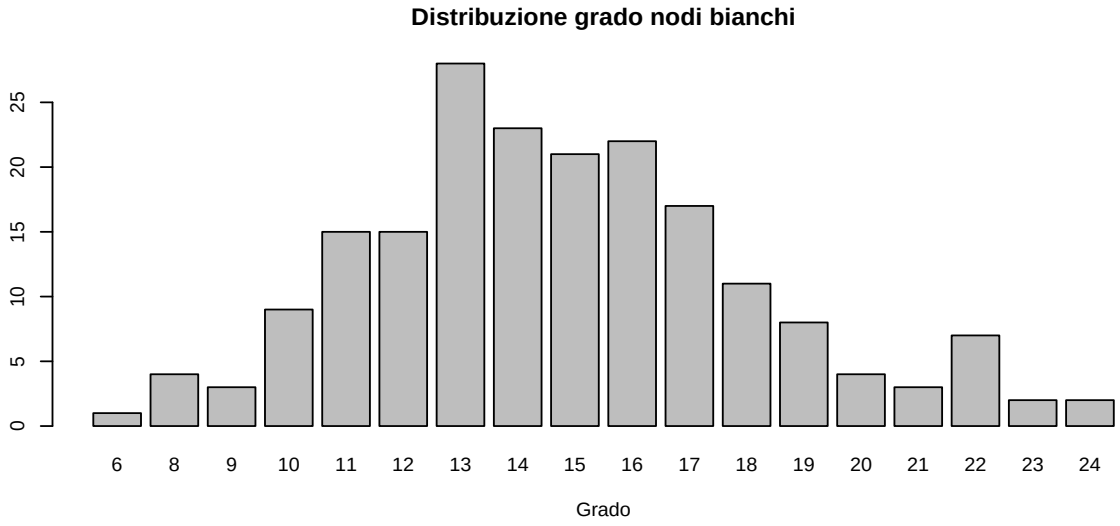
Centralità di base

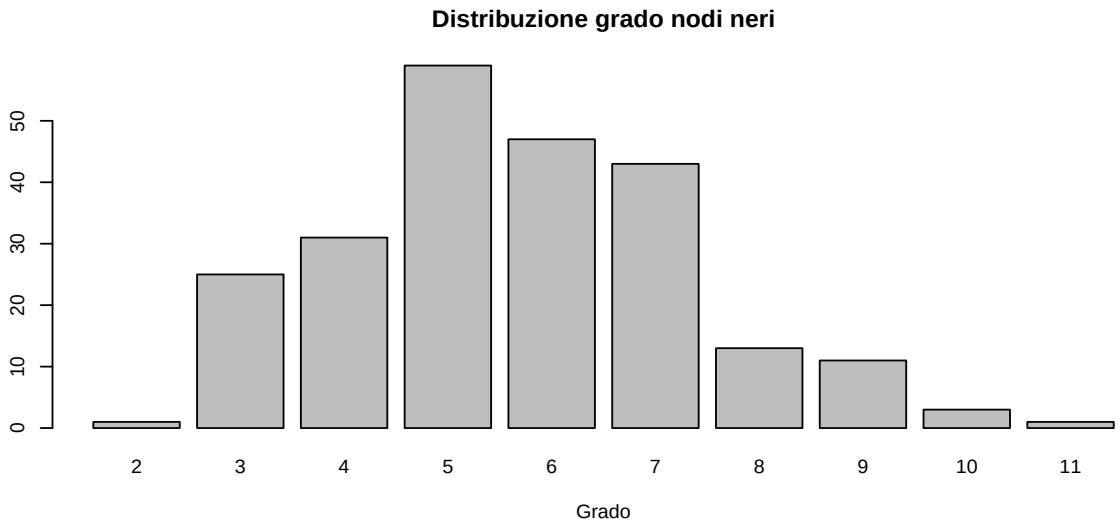
Separiamo i nodi bianchi da quelli neri in quanto la loro natura è abbastanza diversa. Infatti, è possibile vedere che il grado medio e in generale la distribuzione dei gradi nelle reti individuate dai due colori è molto differente.

```
nodi_b = V(g)[type == 1]
nodi_w = V(g)[type == 0]
b_degrees <- degree(g, v = nodi_b)
w_degrees <- degree(g, v = nodi_w)
b_degree_counts <-table(b_degrees)
w_degree_counts <-table(w_degrees)
```

```
## $`Numero di nodi bianchi e neri`
##
##   0   1
## 195 234
##
## $`Numero di archi per tipo`
##
##   0   1   2   3
## 769 440 443 451
##
## $`Distribuzione dei gradi: nodi bianchi`
## w_degrees
##  6  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  1  4  3  9 15 15 28 23 21 22 17 11  8  4  3  7  2  2
##
## $`Distribuzione dei gradi: nodi neri`
## b_degrees
##  2  3  4  5  6  7  8  9 10 11
##  1 25 31 59 47 43 13 11  3  1
```

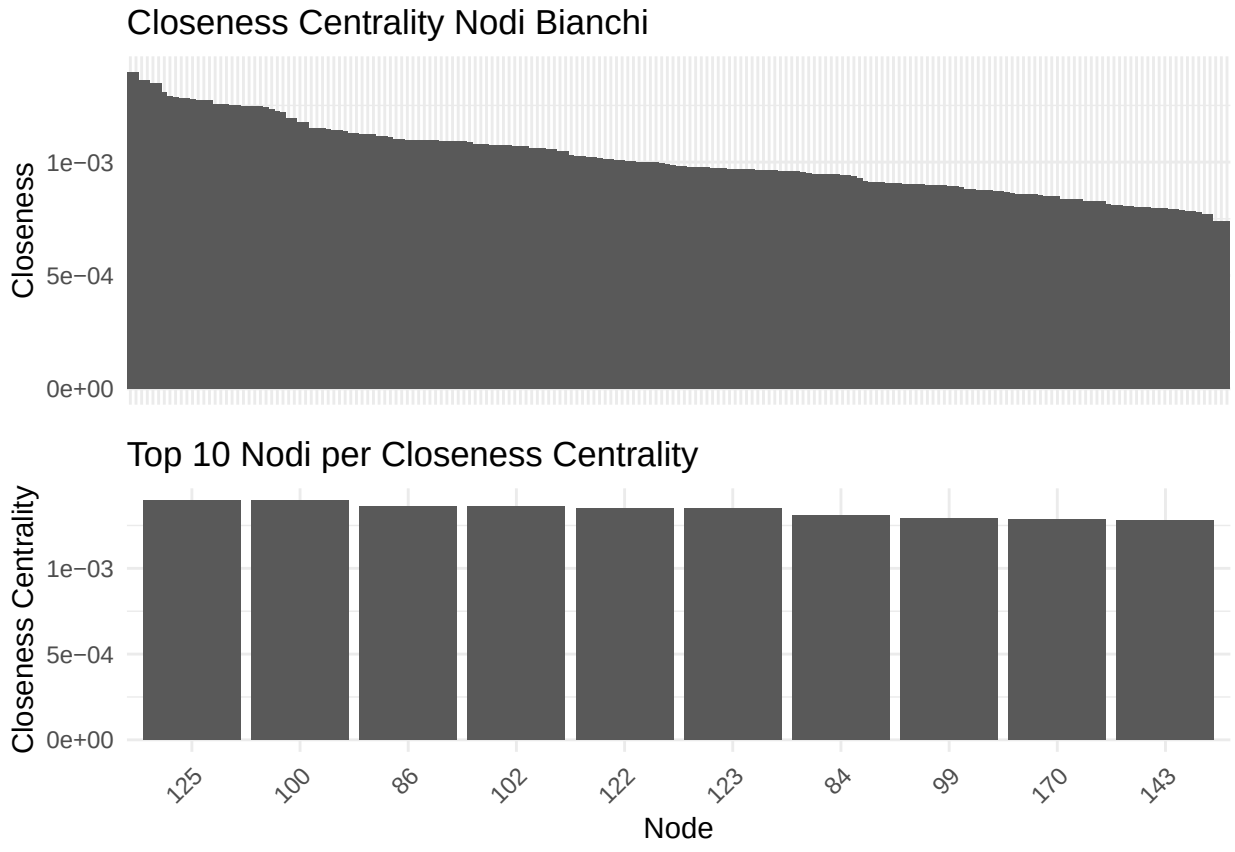

Archi per tipo
0
100
300
500
700
0
1
2
3
Tipo


Distribuzione grado nodi bianchi
0
5
10
15
20
25
6
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Grado

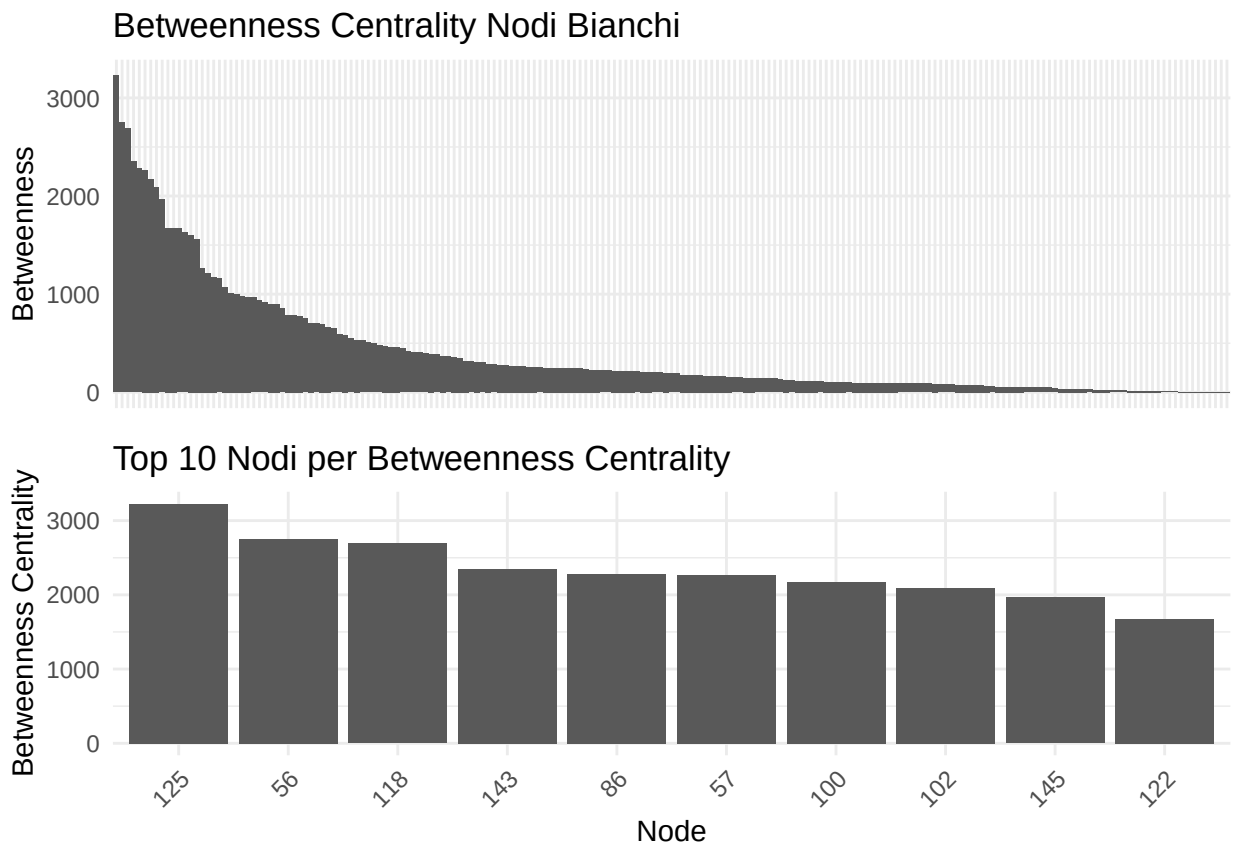

Distribuzione grado nodi neri
0
10
20
30
40
50
2
3
4
5
6
7
8
9
10
11
Grado

La distribuzione del grado per i nodi bianchi si avvicina maggiormente ad una gaussiana. Ha una coda destra leggermente pesante ma tutto sommato la distribuzione è normale. La distribuzione dei nodi neri invece è più sghemba. Proseguendo l'analisi con il calcolo delle centralità più comuni misuriamo la closeness e la betweenness per i grafi indotti dagli archi di tipo 0 e 1 rispettivamente. Questi grafi rappresentano la rete di movimenti di base per entrambe le parti.

```
# Creo il grafo dei nodi bianchi, escludendo gli archi di tipo vicolo
g_w <- subgraph_from_edges(g, which(E(g)$type == 0), delete.vertices = TRUE)
closeness_w <- closeness(g_w)
betweenness_w <- betweenness(g_w)
clo_w <- data.frame(
  Node = names(closeness_w),
  Closeness = closeness_w
)
bet_w <- data.frame(
  Node = names(betweenness_w),
  Betweenness = betweenness_w
)
# Do attenzione particolare ai nodi più centrali per
clo_w <- clo_w[order(-clo_w$Closeness), ]
clo_w_top10 <- clo_w[1:10, ]
bet_w <- bet_w[order(-bet_w$Betweenness), ]
bet_w_top10 <- bet_w[1:10, ]
```



Com'è possibile notare, non sorprende che i nodi topologicamente più centrali secondo la disposizione dei nodi indotta dal tabellone, rappresentino anche i nodi con closeness più alta.



Per quanto riguarda la betweenness, di nuovo, non sorprende come i nodi collocati sulle arterie principali del quartiere che definisce il grafo sono quelli che presentano un valore massimo per la betweenness. Questi due risultati non soprendono particolarmente poichè il grafo del tabellone è artificiale. Esso è pensato per bilanciare le forze in gioco e consentire ai giocatori di trovare conferma in quella che è la loro comprensione della navigabilità di una rete per come rappresentata in mappa.

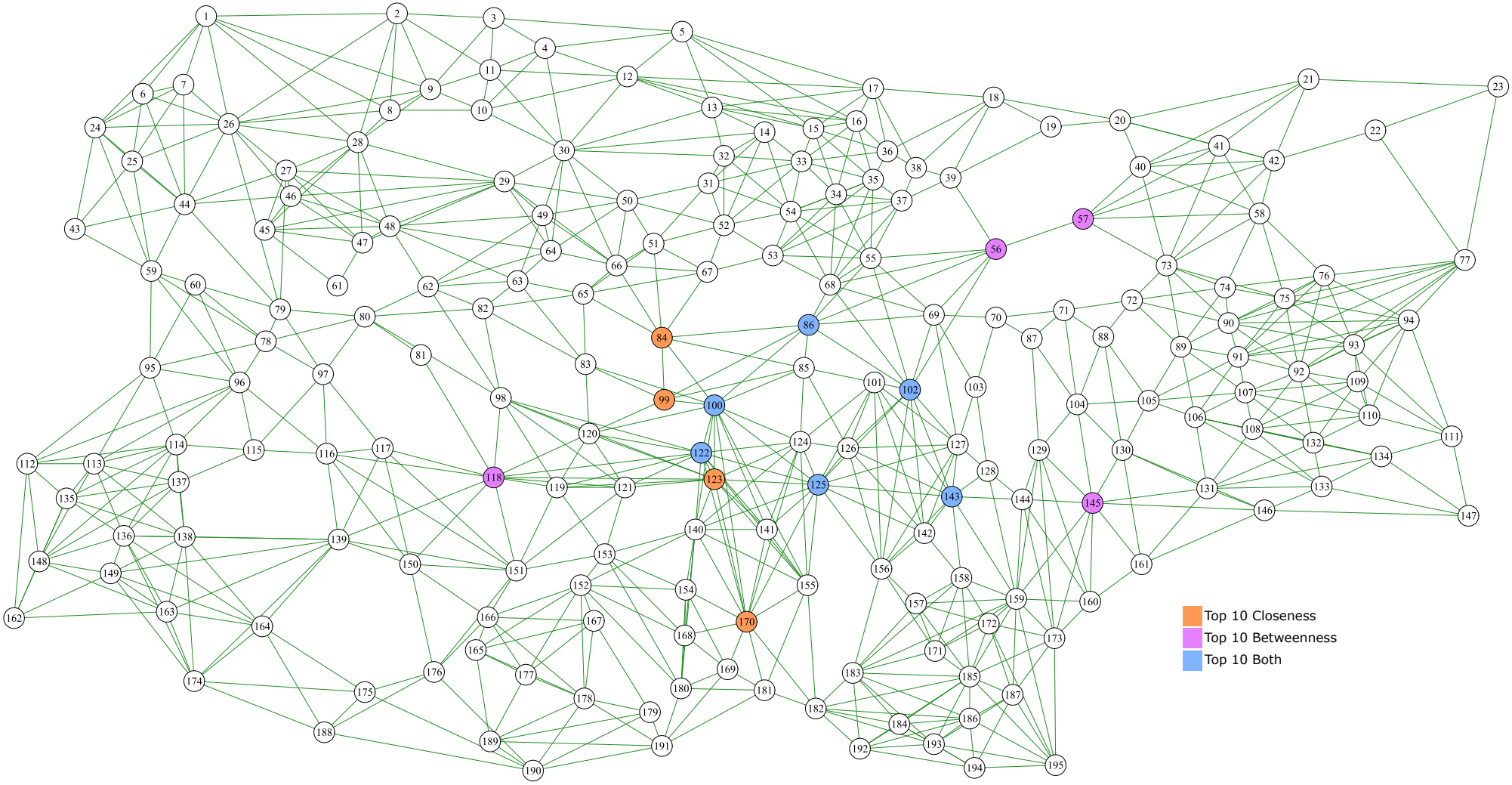
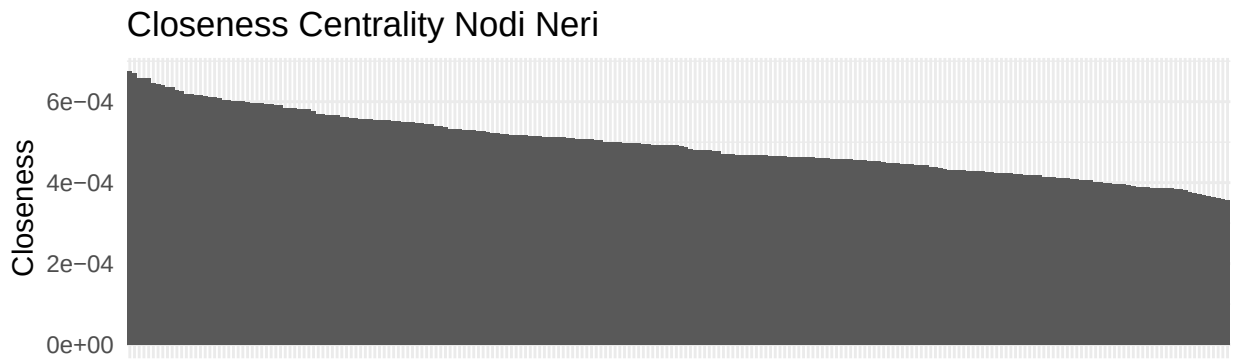
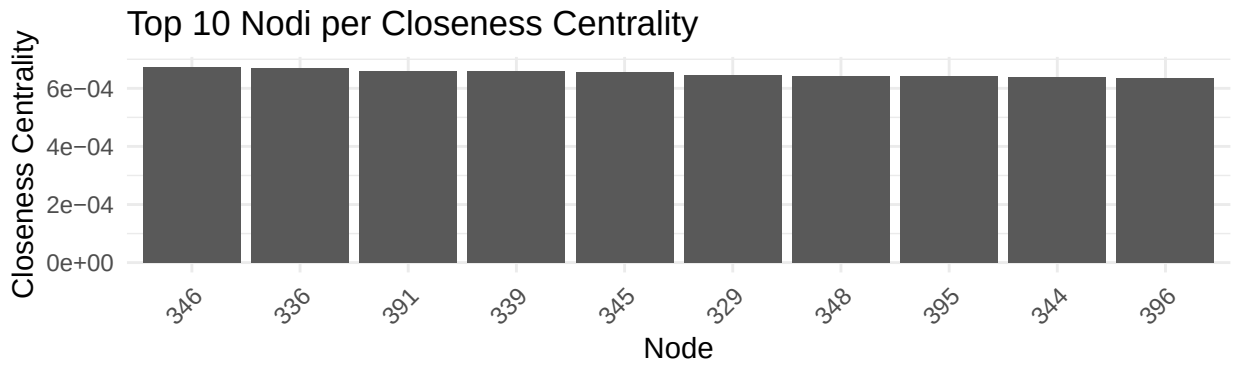
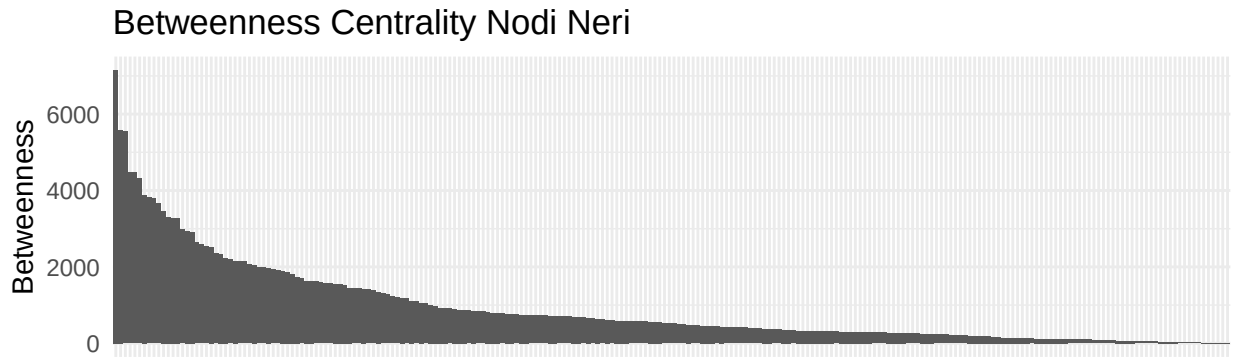


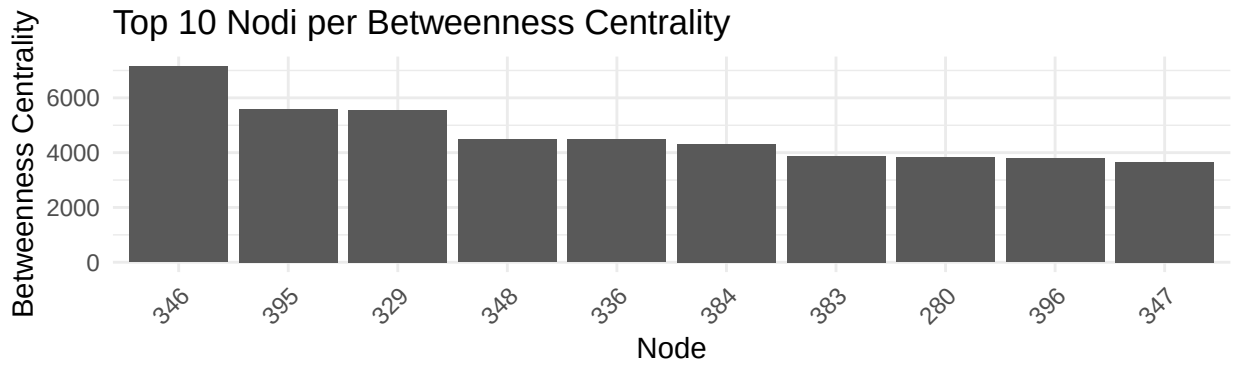
Figure 5: Centralità visualizzate sul grafo bianco.

In maniera simile è possibile calcolare le centralità nella rete definita dai nodi neri con i soli archi semplici di tipo 1









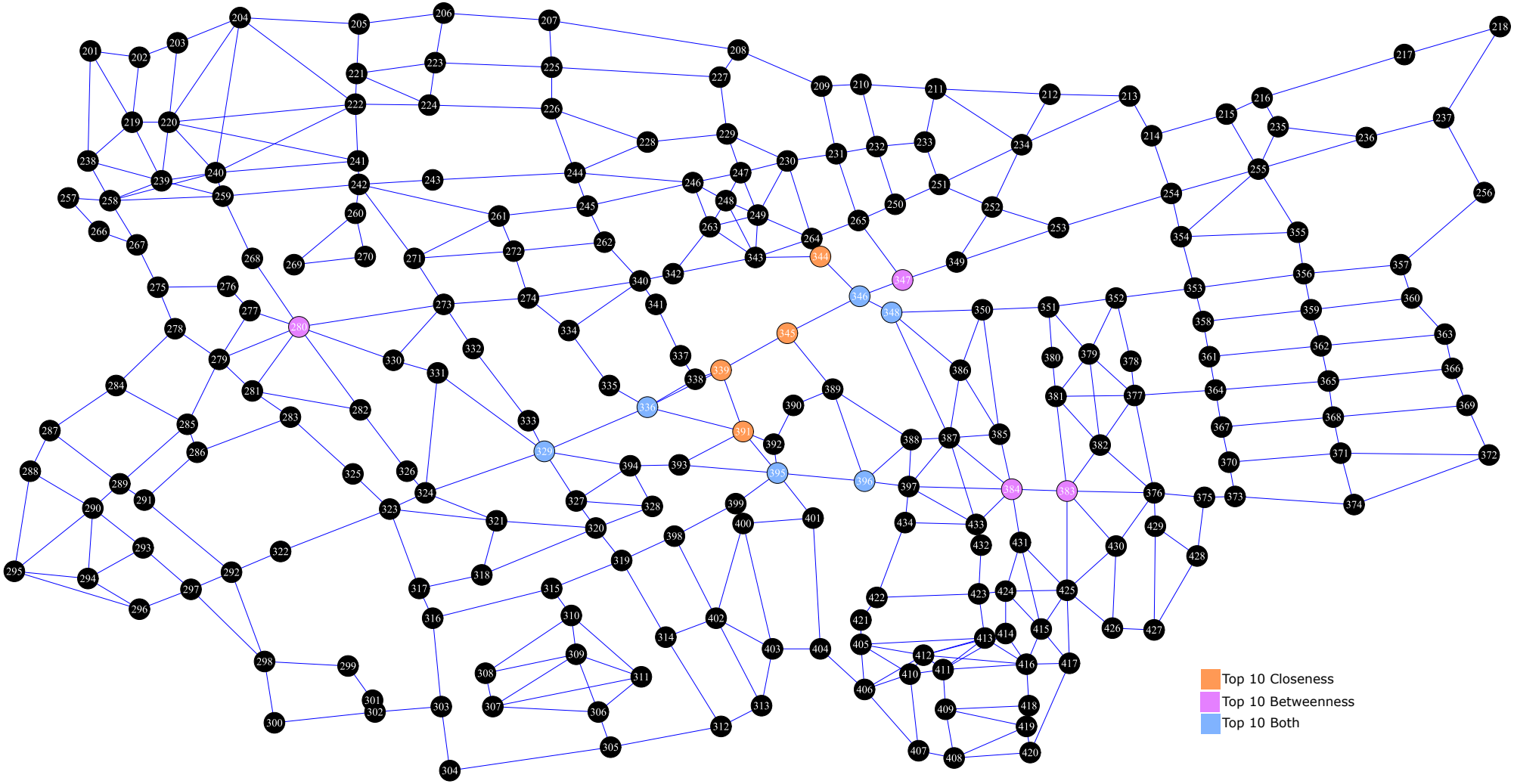


Figure 6: Centralità visualizzate sul grafo nero.

Ricerca delle comunità

```
# Trovo le comunità secondo diversi algoritmi
comm_louvain <- cluster_louvain(g_w)
comm_edge_bet <- cluster_edge_betweenness(g_w)
```

Le comunità trovate dagli algoritmi, in tutti e 3 i casi, rispecchiano molto le comunità che ci saltano all'occhio guardando il tabellone di gioco.

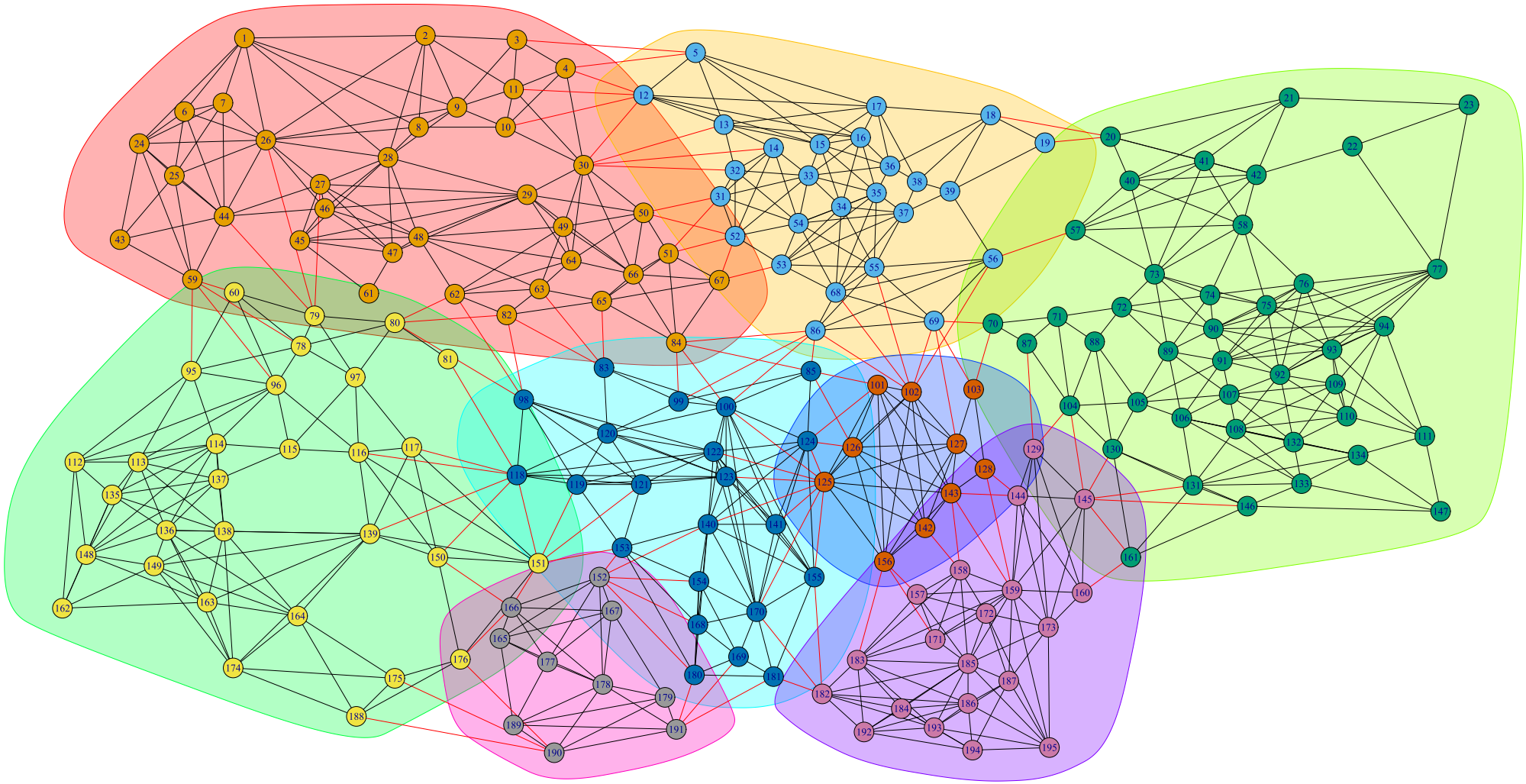


Figure 7: Community detection con algoritmo di Louvain.

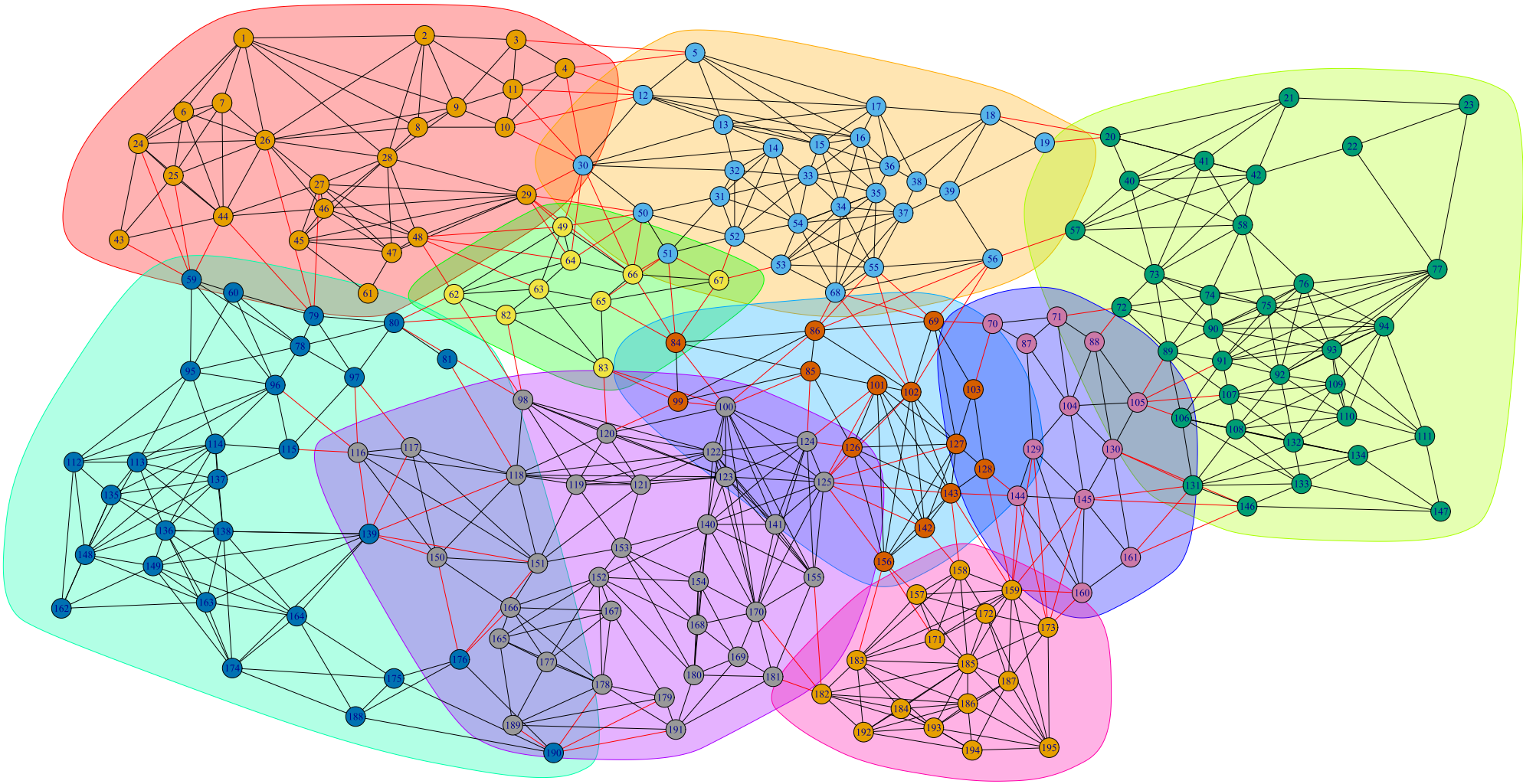


Figure 8: Community detection con algoritmo di Edge Betweenness.

Analisi: qual è il miglior nascondiglio per Jack?

La domanda di ricerca più saliente che suscita il dataset è: quale dei nodi bianchi è il nascondiglio migliore per Jack? Per provare a rispondere a questa domanda è possibile definire delle caratteristiche desiderate e misurarle su ogni nodo a disposizione. I risultati possono poi essere sommati in una combinazione lineare che consente di dare maggior peso ad alcune di esse.

Nota la dinamica dei turni di gioco risulta immediato che i percorsi minimi siano di fondamentale importanza. Questo perchè Jack muove un'unica pedina mentre la polizia 5: più tempo (passi) Jack ci mette per giungere al suo nascondiglio più sono i nodi che la Polizia può raggiungere e controllare per restringere il campo di ricerca.

Un secondo commento che segue spontaneo riguarda quali di questi percorsi siano rilevanti. Nel rilassamento imposto alle regole, i soli nodi in cui è possibile commettere un omicidio sono quelli colorati di rosso, perciò la fuga di Jack comincerà sempre da uno di essi. Mentre la destinazione può essere un qualsiasi nodo candidato nascondiglio.

A questo scopo, integrando la conoscenza del dominio di gioco vengono proposti i seguenti fattori per quantificare la desiderabilità di un nascondiglio.

- Fattore 1 - contributo delle centralità:
  - Per ogni nodo vengono calcolate le centralità
    - Degree
    - Closeness (media per ogni notte di gioco)
    - Eigenvector
  - Per ciascuna viene calcolato
    - Valore normalizzato (min-max)
    - Rango percentile
  - Il punteggio del nodo è la media di questi due (max 100)
- Fattore 2 - contributo della varietà dei percorsi:
  - Per ogni coppia di percorsi minimi da ogni scena del crimine a ogni nascondiglio viene calcolato
    - $1 - |$Similarità di Jaccard tra il vettore dei nodi$|$
    - $1 - |$Similarità di Jaccard tra il vettore degl'archi$|$
  - Questi valori vengono sommati per favorire i nodi con molti percorsi minimi, anche se simili, e normalizzati
  - Il punteggio del nodo è la media del valore ottenuto per nodi e archi (max 100)
- Fattore 3 - contributo della pattugliabilità:
  - Viene definita una metrica ricorsiva per quantificare l'importanza di nodi bianchi e neri nella rete di interazione
    - Un nodo bianco è importante se è vicino a nodi bianchi importanti ed è pattugliabile da nodi neri importanti
    - Un nodo nero è importante se è vicino a nodi neri importanti e può pattugliare nodi bianchi importanti
  - Per ogni nodo viene sommata l'importanza dei nodi neri che lo pattugliano
  - Il vettore viene normalizzato in ordine decrescente (max 100)
- Fattore 4 - contributo rischio dei percorsi:

- Vengono sommati i punteggi di importanza dei nodi neri che pattugliano i nodi bianchi dei percorsi minimi verso ogni nodo,
- La somma ottenuta per ogni nodo viene divisa per il numero di percorsi considerati
- La media ottenuta viene normalizzata (min-max e rango-percentile)
- La somma di questi due valori è il punteggio del nodo (max 100)

Calcolo degli archi derivati

```
compute_all_edges <- function(edges, edge_type) {
  max_id <- max(edges$id, na.rm = TRUE)

  directed_base_edges <- edges %>% bind_rows(edges %>%
              mutate(from = edges$to, to = edges$from))

  two_step_edges <- directed_base_edges %>%
    inner_join(directed_base_edges, by = c("to" = "from"), suffix =  c("_1","_2")) %>%
    filter(from != to_2) %>%  # Avoid cycles like 2->3->2
    transmute(
      from = from,
      step = to,
      to = to_2,
      type = edge_type  # The pivot node
    ) %>%
    arrange(from, to, step) %>%
    distinct(.keep_all = TRUE) %>%  # Remove duplicate paths
    mutate(id = row_number() + max_id)

  return(two_step_edges)
}
```

```
if (!file.exists("datasets/all_edges.csv")){
  carriage_edges = compute_all_edges(base_edges%>%filter(type==0),4)
  black_double_edges = compute_all_edges(base_edges%>%filter(type==1),5)
  edges <- bind_rows(base_edges, carriage_edges, black_double_edges)
  write.csv(edges, "datasets/all_edges.csv", sep=",", row.names = FALSE)
}else{
  edges <- read.csv("datasets/all_edges.csv")
}
```

Calcolo dei percorsi minimi

I percorsi minimi tra tutte le scene del crimine e i nascondigli candidati vengono trovati utilizzando una variazione di BFS. Questo è necessario poichè le regole di gioco escludono alcuni percorsi i quali devono:

- Terminare con un arco semplice di tipo 0
- Contenere al massimo un numero di archi carrozza e vicolo compatibile con la notte di gioco

Anche il computo dei percorsi minimi è particolarmente dispendioso in termini di tempo perciò se nella cartella di lavoro sono presenti i file che li contengono, divisi per notte ti gioco, vengono utilizzati quest'ultimi.

Il codice C++ utilizzato per il calcolo dei percorsi minimi è presente nella cartella cppScripts: "calcoloCammini.exe". Utilizza il parallelismo con OpenMP e si aspetta di trovare nella working directory la cartella "datasets" da quale legge il file "all_edges.csv"

```
if (!file.exists("datasets/n1.csv")){
  exe_path <- "./cppScripts/FastPathComputing.exe"
  system2(exe_path, wait = TRUE)
}
n1 <- read.csv("datasets/n1.csv")
n2 <- read.csv("datasets/n2.csv")
n3 <- read.csv("datasets/n3.csv")
n4 <- read.csv("datasets/n4.csv")
```

```
murder_sites <- c(3,21,27,65,84,147,149,158)
all_paths <- bind_rows(n1, n2, n3, n4)
# Creazione del grafo completo
complete_g <- graph_from_data_frame(vertices = nodes,d = edges, directed = FALSE)
```

Fattore 1: contributo delle centralità

Vengono definite le tre funzioni utilizzate per la costruzione del punteggio e il calcolo della closeness nel caso in cui i cammini minimi siano noti. Quest'ultima è necessaria poichè la distanza tra un nodo candidato e un altro non è rilevante, contano solo quelle tra nodi candidati e scene del crimine.

```
norm_min_max <- function(x) {
  ((x)/ max(x))*100
}

rank_percentiles <- function(x) {
  ecdf(x)(x) * 100
}

clos_with_paths <-function(paths){
  paths %>% group_by(to, from,cost)%>%summarize(count = n(),.groups = "drop")%>%
   group_by(to)%>%summarize(clos = 1/sum(cost), .groups = "drop")
}
```

Viene costruito il grafo dei nodi bianchi, questa volta al completo, includendo anche gli archi di tipo 3 e 4.

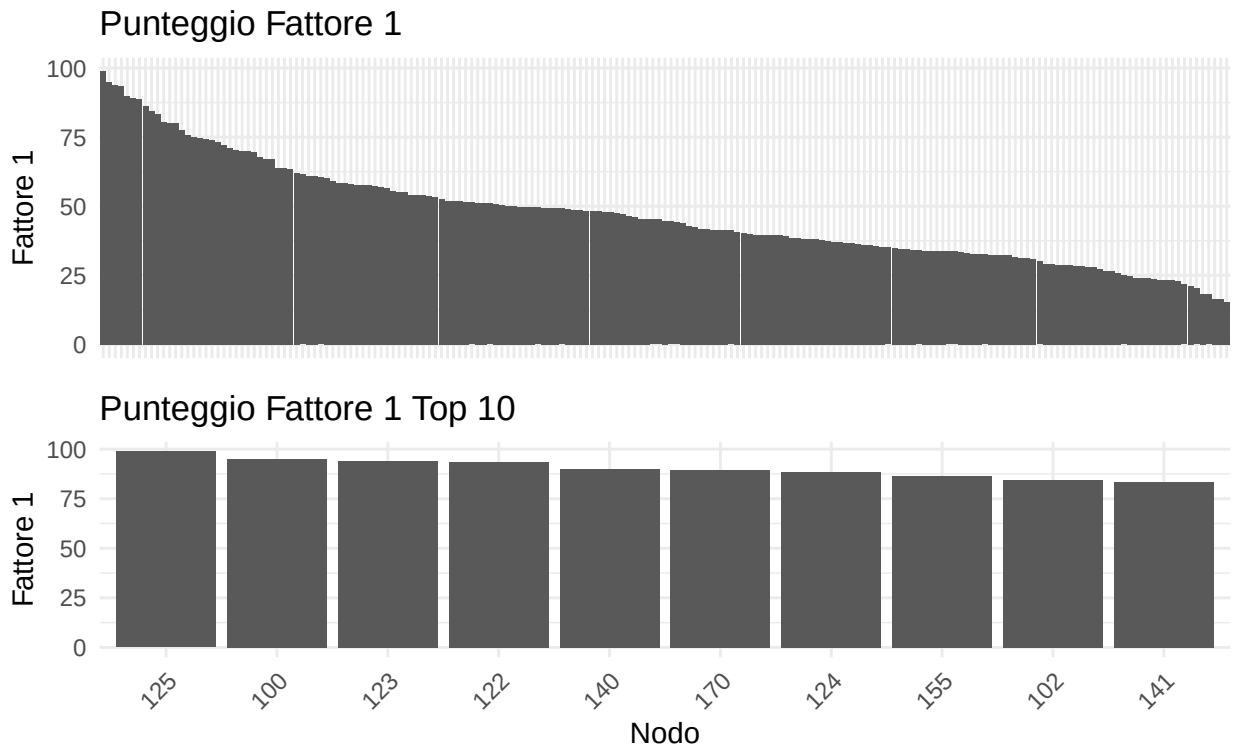
```
g_w <- subgraph_from_edges(complete_g, which(E(complete_g)$type %in% c(0,3,4)),
                           delete.vertices = TRUE)
hideouts <- V(g_w)[!V(g_w) %in% murder_sites]
```

E quindi vengono calcolate le centralità

```
degree_centrality <- degree(g_w)
degree_centrality = degree_centrality[!names(degree_centrality) %in% murder_sites]
degree_norm <- norm_min_max(degree_centrality)
degree_rank <- rank_percentiles(rank(degree_centrality, ties.method = "average"))

c_centrality <- data.frame(
  node = V(g_w),
```

```
  clo_n1 = clos_with_paths(n1)$clos,
  clo_n2 = clos_with_paths(n2)$clos,
  clo_n3 = clos_with_paths(n3)$clos,
  clo_n4 = clos_with_paths(n4)$clos
)
c_centrality$closeness <- rowMeans(c_centrality[,2:5])
c_centrality = c_centrality %>% filter(!node %in% murder_sites)
close_norm <- norm_min_max(c_centrality$closeness)
close_rank <- rank_percentiles(rank(c_centrality$closeness, ties.method = "average"))

eigen_centrality <- eigen_centrality(g_w)$vector
eigen_centrality = eigen_centrality[!names(eigen_centrality) %in% murder_sites]
eigen_norm <- norm_min_max(eigen_centrality)
eigen_rank <- rank_percentiles(rank(eigen_centrality, ties.method = "average"))

factor_1_raw <- data.frame(
  node = hideouts,
  degree_norm,
  degree_rank,
  close_norm,
  close_rank,
  eigen_norm,
  eigen_rank
)

factor_1_score <- data.frame(
  node = hideouts,
  score = rowMeans(factor_1_raw[,2:7])
)
partial_score = data.frame(
  node = factor_1_score$node,
  Fattore1 = factor_1_score$score
)
```

Visualizziamo i punteggi relativi al primo fattore per i nodi bianchi.



Non sorprende che i nodi che avevamo evidenziato come centrali nella fase esplorativa del dataset siano gli stessi che risultano particolarmente centrali quando aggiungiamo tutti gli archi legali del grafo. % Inserire immagine della topologia del grafo per il confronto

Fattore 2:

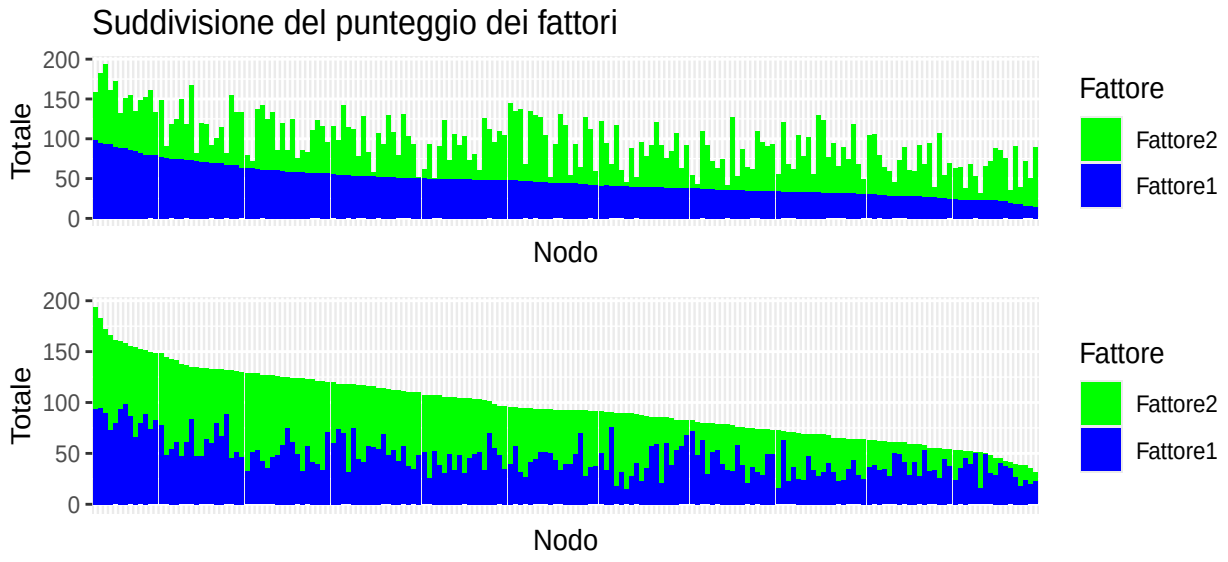
Viene definita la funzione che calcola la similarità di Jaccard tra i vettori dei nodi e degl'archi di percorsi minimi a partire dalla stessa scena del crimine verso lo stesso nodo candidato.

Dato che la computazione della similarità è una molto dispendiosa in termini di tempo (decine di minuti), se è già presente nella cartella di lavoro il dataset relativo, la computazione viene saltata. Il codice C++ utilizzato per il calcolo della similarità di Jaccard è presente nella cartella cppScripts: "calcoloSimilarita.exe". Utilizza il parallelismo con OpenMP e si aspetta di trovare nella working directory la cartella "datasets" da quale legge il file "all_edges.csv"

```
if (!file.exists("datasets/path_similarity.csv")){
  exe_path <- "./cppScripts/pathSimilarity.exe"
  system2(exe_path, wait = TRUE)
}
```

```
path_similarity <- read.csv("datasets/path_similarity.csv")
path_similarity = path_similarity[-murder_sites,]

node_similarity_norm <- norm_min_max(100-norm_min_max(path_similarity$avg_node_iou))
node_similarity_rank <- 100-rank_percentiles(path_similarity$avg_node_iou)
edge_similarity_norm <- norm_min_max(100-norm_min_max(path_similarity$avg_edge_iou))
edge_similarity_rank <- 100-rank_percentiles(path_similarity$avg_edge_iou)
factor_2_raw <- data.frame(
  node = hideouts,
```

```
  node_similarity_norm,
  node_similarity_rank,
  edge_similarity_norm,
  edge_similarity_rank
)

factor_2_score <- data.frame(
  node = factor_2_raw$node,
  score = rowMeans(factor_2_raw[,2:5])
)
```




```
cor_matrix <- cor(partial_score[, c("Fattore1", "Fattore2")])
print(cor_matrix)
```

```
##           Fattore1  Fattore2
## Fattore1 1.0000000 0.1686715
## Fattore2 0.1686715 1.0000000
```

Fattore 3: contributo della pattugliabilità

Le regole del gioco "Lettere da Whitechapel" prevedono che nella sua fuga, Jack, debba raggiungere il suo nascondiglio con un percorso il cui ultimo arco è un semplice arco ti tipo 0. Questo vuol dire che non è possibile avere un percorso valido che termina con l'utilizzo di un vicolo o di una carrozza. Dato che questi movimenti alternativi sono l'unico modo che Jack ha a disposizione per evitare una pattuglia della Polizia che si frappone tra di lui e il nodo che vuole raggiungere, diventa importante capire quanto sia conveniente e strategico per la Polizia avere una delle sue pedine in un dato nodo. Per misurare questa quantità è possibile definire una metrica ricorsiva che bilanci l'importanza dei nodi neri e dei nodi bianchi in base all'interazione che hannno con gli altri nodi del proprio tipo o di quello opposto.

In questa analisi viene proposta una metrica che definisce in questo modo l'importanza di un nodo:

- Importanza di un nodo bianco:
  - Un nodo bianco è importante se è adiacente a nodi bianchi importanti ($\beta = 0.9$)

  - Un nodo bianco è importante se è pattugliato da nodi neri importanti ($\alpha = 0.1$)

- Importanza di un nodo nero:
  - Un nodo nero è importante se pattuglia nodi bianchi importanti ($\gamma = 0.6$)
  - Un nodo nero è importante se è adiacente a nodi nero importanti ($\delta = 0.4$)

Al fine di calcolare l'importanza di un nodo per il Fattore 3 vengono usate due matrici di adiacenza differenti:

- Eliminazione degli archi multipli: $if \quad A_{ij} > 1 \rightarrow A_{ij} = 1$
- Matrice di adiacenza con archi multipli

```
policing_graph = subgraph_from_edges(complete_g, which(E(complete_g)$type == 2),
                                     delete.vertices=FALSE)
adj_bw <- as.matrix(as_adjacency_matrix(policing_graph))[196:429,1:195]
adj_bw[adj_bw>0]<-1
adj_wb <- as.matrix(as_adjacency_matrix(policing_graph))[1:195,196:429]
adj_wb[adj_wb>0]<-1
adj_bb <- as.matrix(as_adjacency_matrix(g_b))
adj_bb[adj_bb>0]<-1
adj_ww <- as.matrix(as_adjacency_matrix(g_w))
adj_ww[adj_ww>0]<-1

# Number of nodes
num_black <- 234
num_white <- 195

# Initialize power vectors
black_power <- rep(1, num_black)
white_power <- rep(1, num_white)

# Set stopping criterion
epsilon <- 1e-6
max_iter <- 100
diff <- Inf
iter <- 0
alpha <- 0.9
beta <- 0.1
gamma <- 0.6
delta <- 0.4
```

Definita l'importanza di un nodo nero, viene definita la pattugliabilità di un nodo bianco come la somma dell'importanza dei nodi neri che lo pattugliano. La pattugliabilità non viene normalizzato per il numero di nodi neri che pattugliano lo stesso nodo.

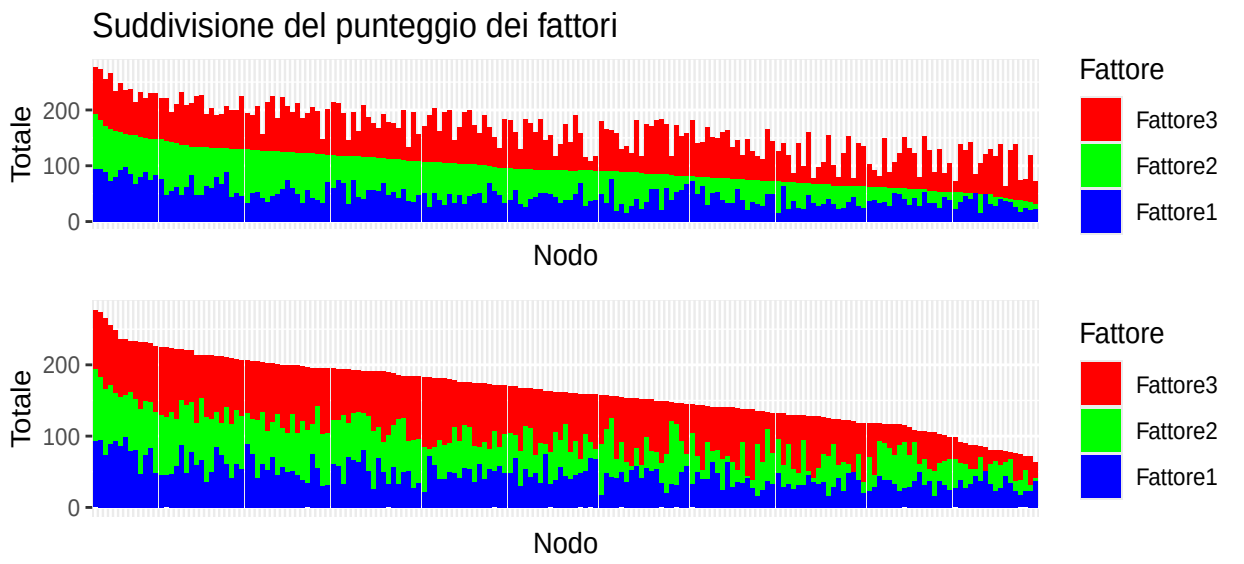
```
while (diff > epsilon && iter < max_iter) {
  iter <- iter + 1
  black_power_new <- norm_min_max(gamma*adj_bw%*%white_power + delta*adj_bb%*%black_power)
  white_power_new <- norm_min_max(alpha*adj_wb%*%black_power + beta*adj_ww%*%white_power)

  diff <- max(abs(black_power_new - black_power),abs(white_power_new - white_power))
  # Update power vectors
  black_power <- black_power_new
```

```
  white_power <- white_power_new
}
single_edge = 100-(adj_wb%*% black_power/rowSums(adj_wb))
```

```
adj_bw <- as.matrix(as_adjacency_matrix(policing_graph))[196:429,1:195]
adj_wb <- as.matrix(as_adjacency_matrix(policing_graph))[1:195,196:429]
adj_bb <- as.matrix(as_adjacency_matrix(g_b))
adj_ww <- as.matrix(as_adjacency_matrix(g_w))

# Initialize power vectors
black_power <- rep(1, num_black)
white_power <- rep(1, num_white)

# Set stopping criterion
diff <- Inf
iter <- 0
```

```
while (diff > epsilon && iter < max_iter) {
  iter <- iter + 1
  black_power_new <- norm_min_max(gamma*adj_bw%*%white_power + delta*adj_bb%*%black_power)
  white_power_new <- norm_min_max(alpha*adj_wb%*%black_power + beta*adj_ww%*%white_power)

  diff <- max(abs(black_power_new - black_power),abs(white_power_new - white_power))
  # Update power vectors
  black_power <- black_power_new
  white_power <- white_power_new
}
multi_edge = 100-(adj_wb%*% black_power/rowSums(adj_wb))
```

```
factor_3_raw <- data.frame(
  node <- nodi_w,
  norm = (norm_min_max(single_edge)+norm_min_max(multi_edge))/2,
  rank = (rank_percentiles(single_edge)+rank_percentiles(multi_edge))/2
)
factor_3_complete <- data.frame(
  node <- nodi_w,
  score = (factor_3_raw$norm+factor_3_raw$rank)/2
)
factor_3_score <- data.frame(
  node = hideouts,
  score = (factor_3_raw[-murder_sites,]$norm+factor_3_raw[-murder_sites,]$rank)/2
)
```




```
cor_matrix <- cor(partial_score[, c("Fattore1", "Fattore2", "Fattore3")])
print(cor_matrix)
```

```
##           Fattore1  Fattore2  Fattore3
## Fattore1 1.0000000 0.1686715 0.3508092
## Fattore2 0.1686715 1.0000000 0.1171943
## Fattore3 0.3508092 0.1171943 1.0000000
```

Fattore 4: rischio dei percorsi

Un'ulteriore caratteristica desiderabile per un nascondiglio è quella di avere a disposizione percorsi minimi che percorrono nodi con un punteggio di pattugliabilità basso.

Questo vuol dire che cercare di seguire Jack richiede alla Polizia di piazzare le proprie pedine i nodi neri meno potenti riducendone l'utilità. A tale scopo, per misurare quanto i percorsi minimi verso un nodo siano sicuri o rischiosi, è possibile utilizzare il punteggio di pattugliabilità definito poc'anzi nel Fattore 3.

E' possibile sommare il punteggio di ogni nodo bianco in ciascuno dei percorsi Possiamo utilizzare il punteggio del Fattore 3 relativo alla pattugliabilità di un nodo

Prima di poter calcolare il rischio dovuto alla pattugliabilità dei percorsi viene creata una matrice con dimensioni $num\_path \times num\_nodi$ che contiene un 1 nella colonna associata ad un nodo se il percorso a quella riga attraversa quel nodo.

Creare questa matrice non è banale poichè i percorsi calcolati contengono anche archi di tipo 4. La lista di nodi visitati contenuta nelle colonne node1…node11 non tengono conto dei nodi intermedi attraversati nell'utilizzo di archi carrozza. Detto questo è necessario considerarli poichè la Polizia, qualora decidesse di pattugliare tali nodi intermedi, verrebbe informata del passaggio di Jack.

Il codice C++ utilizzato per il calcolo della matrice binaria dei percorsi è presente nella cartella cppScripts: "calcoloPathMatrix.exe". Utilizza il parallelismo con OpenMP e si aspetta di trovare nella working directory la cartella "datasets" da quale legge il file i file "n1.csv", "n2.csv", "n3.csv" e "n4.csv".

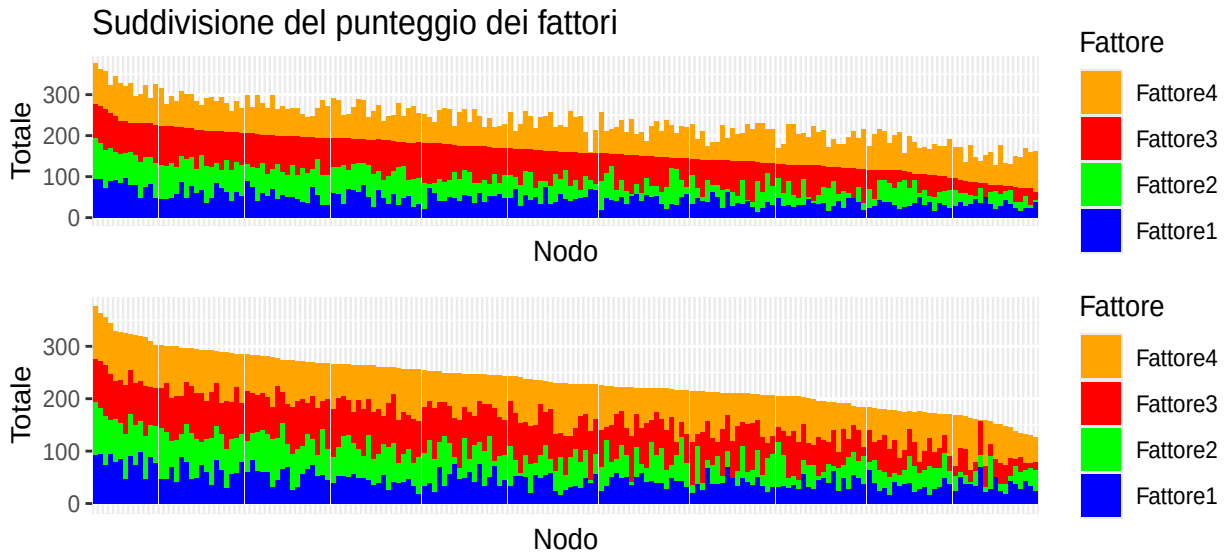
```
if (!exists("path_matrix_data")) {
  if(!file.exists("datasets/path_matrix.csv")){
    print("entro")
```

```
    exe_path <- "./cppScripts/path_matrix.exe"
    system2(exe_path, wait = TRUE)
  }
  path_matrix_data <- read.csv("datasets/path_matrix.csv", header = FALSE,
                               col.names = c("node",1:195))
}
```

```
path_matrix_data <- path_matrix_data %>% arrange(node)
path_matrix <- as.matrix(path_matrix_data)
path_policing_matrix <- path_matrix[,2:196] %*% (100-factor_3_complete$score)
norm_path_pol_matrix <- path_policing_matrix / all_paths$cost
norm_path_pol_cost <- rowSums(norm_path_pol_matrix)
path_policing_frame <- data.frame(
  to = all_paths$to,
  cost = all_paths$cost,
  policing = norm_path_pol_cost
)
path_policing_score <- path_policing_frame %>% group_by(to) %>%
  summarize(
    length = sum(cost),
    count = n(),
    policing_score = sum(policing)/length/count,
    .groups = "drop"
  )
path_policing_score = path_policing_score[-murder_sites,]
norm_4 = norm_min_max(path_policing_score$policing_score)
rank_4 = rank_percentiles(path_policing_score$policing_score)
factor_4_score = data.frame(
  node = hideouts,
  score = 100-(norm_4+rank_4)/2
)
```

```
##   [1] 182.32270 180.92370 151.44958 189.69501 166.53999 183.46432 161.83099
##   [8] 172.55252 119.90704 144.59438 158.32702 172.34972 141.91383 153.43461
##  [15] 156.63977 124.22095 138.15759 162.57284 146.22401 148.53815 161.08734
##  [22] 189.97021 154.42071 146.69518 151.63021 137.25856 165.16271 132.23529
##  [29] 135.27409 187.03287 119.39774 116.01855 137.82293  89.38269 164.94087
##  [36] 150.98780 121.02272 160.51530 144.19833 168.64170 171.10455 155.59362
##  [43] 163.17406 177.12749 131.85517 104.02083 179.63921 164.78711 174.69907
##  [50] 133.50916  87.30115 128.45126 103.23708 142.14027 133.03158 149.22316
##  [57] 145.58906 159.69830 110.07053 121.01673 134.57262 159.79242 104.26030
##  [64] 133.98976 101.15164 113.16537 152.20672 159.38983 112.28952 166.14153
##  [71] 159.37679 123.40653 152.35694 183.39083 162.81812 124.61267 105.19183
##  [78] 144.32028 110.14819  87.99786  73.84395 158.27639 165.84658 131.55554
##  [85] 111.20625 178.66615 162.80904 150.51983 174.41008 129.42072 143.74091
##  [92] 123.35984 112.09697 127.33142  83.02615 142.54507  66.30139  65.80359
##  [99]  92.03483 120.74315 172.94230 153.76792 158.12336 165.29219 138.60199
## [106] 140.35783 101.75029 178.58183 119.25386 176.42274 137.25863 145.98293
## [113] 167.32854 161.08252  97.75142 160.01735 120.93927 130.56880 162.64194
## [120] 103.37280 103.38794 148.14948 144.44073 156.15378 180.41891 178.43910
## [127] 138.59614 144.83014 117.32789 161.36360 138.35185 126.96204 124.44125
## [134] 145.59835 118.63213  99.67539 121.12860 103.91031  66.54420 156.14272
## [141] 123.13629 178.88353  61.33631 103.79567 102.69098 152.61434 109.34876
```

```
## [148]  61.06301 115.98424 143.59138  68.25218 155.38652 166.77766 112.25533
## [155] 107.61677  98.77123 148.67633  72.71591 107.36756 134.20418 148.62596
## [162] 120.46697 119.24967 115.93629 182.53429 152.09071 121.35261 174.58069
## [169]  91.74927 112.99522 115.84170 151.51615 143.29431 159.26102  99.41246
## [176] 128.62404 160.84848 155.79597 158.57899 182.53318 120.10946 180.13445
## [183] 175.06512 108.83735 112.49729 179.46316 126.49136
```




```r
cor_matrix <- cor(partial_score[, c("Fattore1", "Fattore2", "Fattore3", "Fattore4")])
print(cor_matrix)
```

```
##            Fattore1  Fattore2   Fattore3    Fattore4
## Fattore1  1.0000000 0.1686715 0.35080920 -0.01421620
## Fattore2  0.1686715 1.0000000 0.11719432  0.10124862
## Fattore3  0.3508092 0.1171943 1.00000000  0.05155958
## Fattore4 -0.0142162 0.1012486 0.05155958  1.00000000
```

Conclusioni

Dopo aver calcolato il punteggio di ogni fattore per ogni nodo del grafo è possibile sommare i punteggi per avere un quadro globale su quali dei nodi risultino dei canditati più validi come nascondigli.

```r
best_hideout = data.frame(
  node = factor_1_score$node,
  score = factor_1_score$score + factor_2_score$score +
    0.5*factor_3_score$score + 0.5*factor_4_score$score
)

best_hideout <- best_hideout[order(-best_hideout$score), ]
best_hideout_top10 <- best_hideout[1:10,]
```

```r
plot1<-ggplot(best_hideout, aes(x = reorder(node, -score), y = score)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
```

```
  labs(title = "Totale punteggio",x = NULL, y = "Punteggio")+
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank())
plot2<-ggplot(best_hideout_top10, aes(x = reorder(node, -score), y = score)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  labs(title = "Totale punteggio Top 10",x = NULL, y = "Punteggio")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
grid.arrange(plot1, plot2, ncol = 1)
```