

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 momenti 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: interella Jack per chiedere se, nella sua fuga, è passato per qualcuno dei nodi bianchi adiacenti a quello dove si trova la pedina
 - Effettuare un arresto: interella 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)
# 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
```

```
# 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'emissione di arresti.

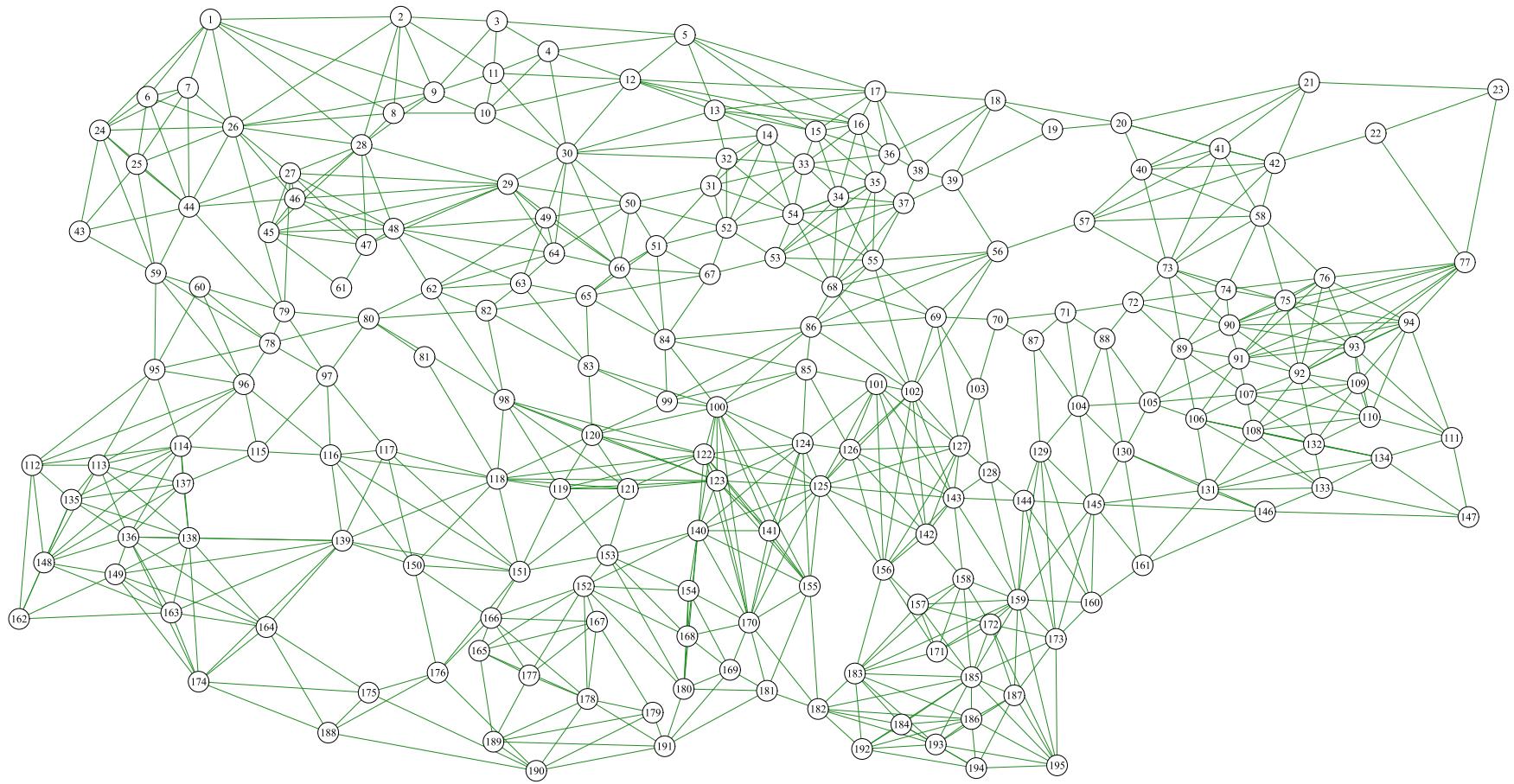


Figure 2: Il grafo dei nodi bianchi con archi di tipo 0

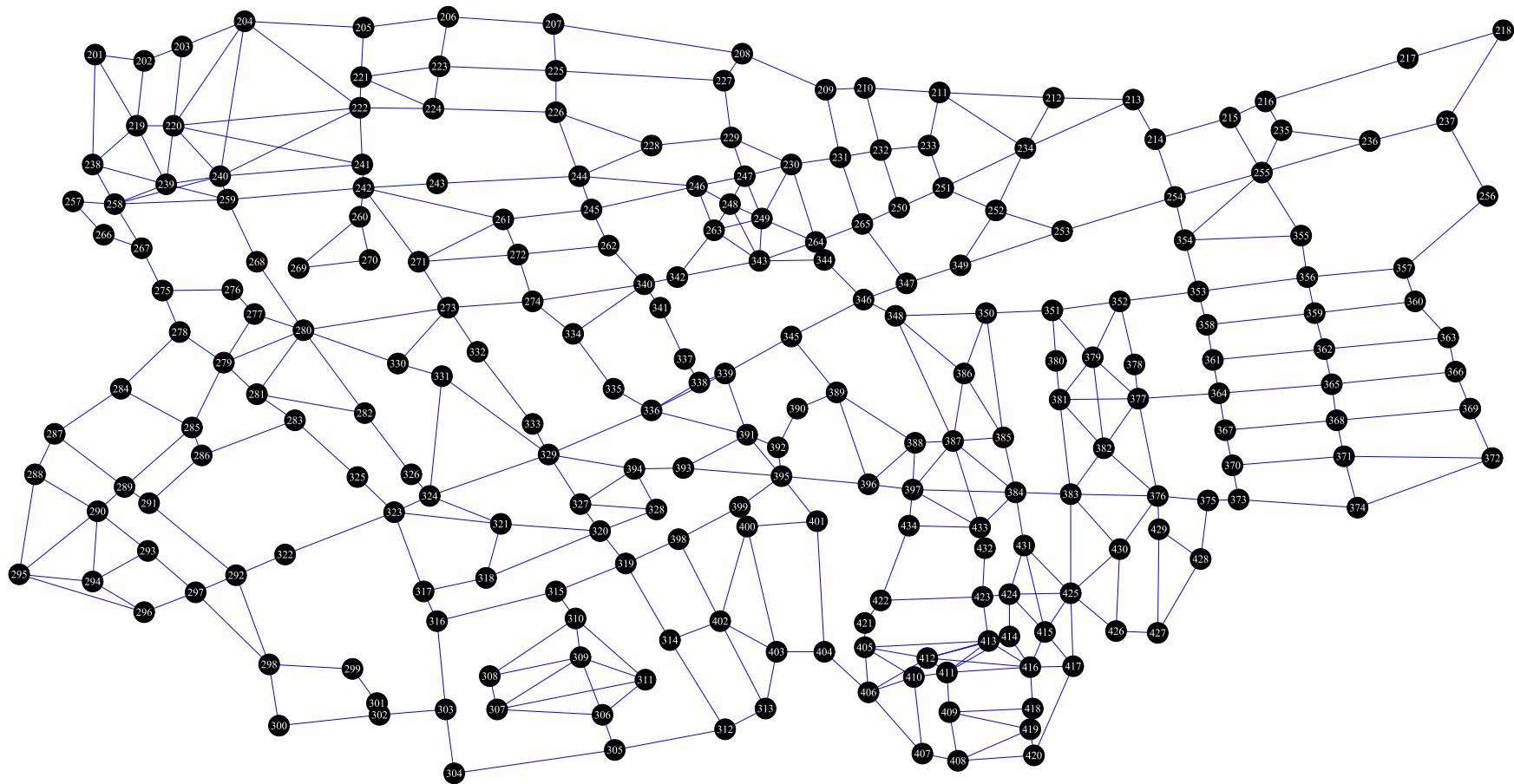


Figure 3: Il grafo dei nodi neri con archi di tipo 1

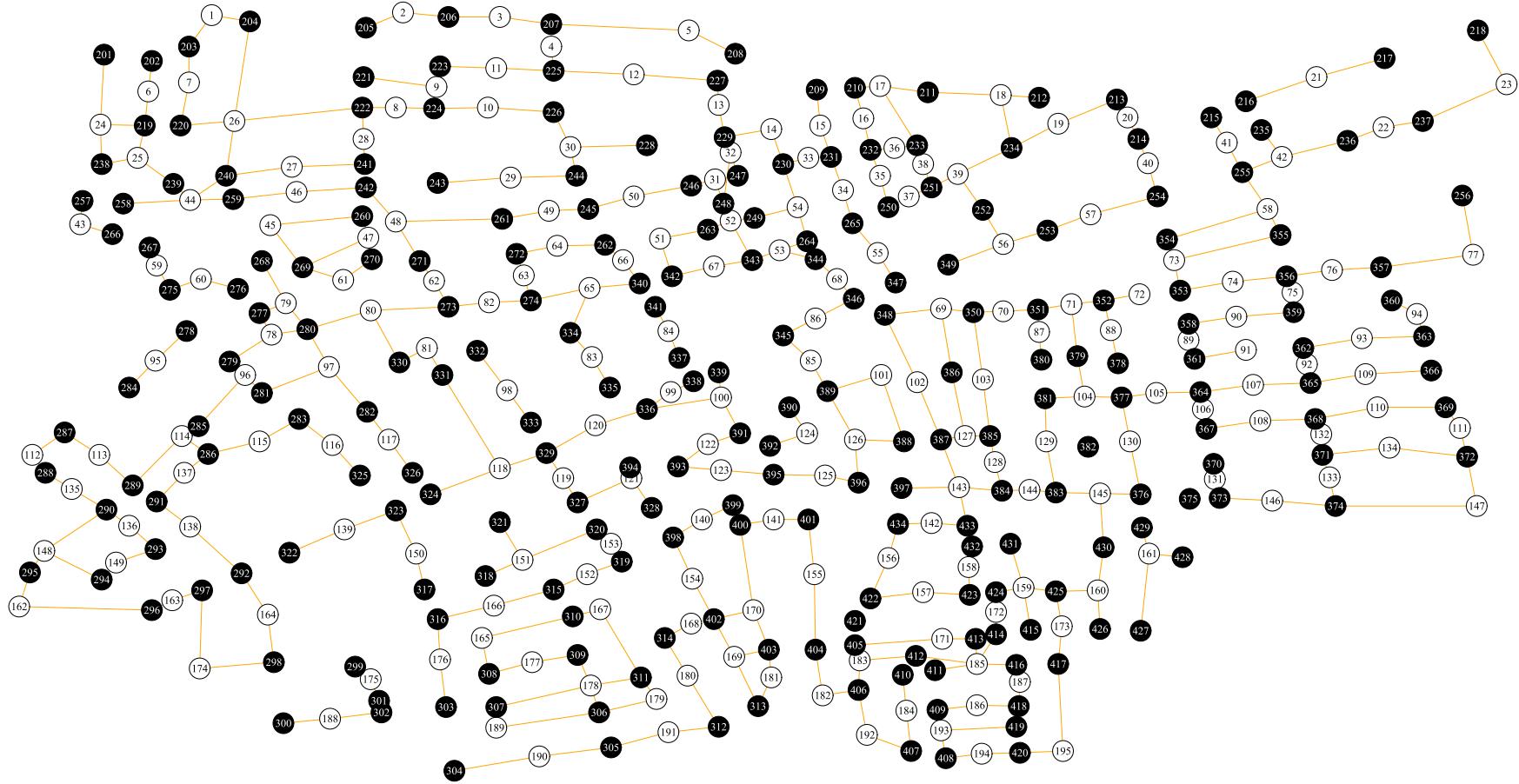


Figure 4: Il grafo di pattugliamento con archi di tipo 2

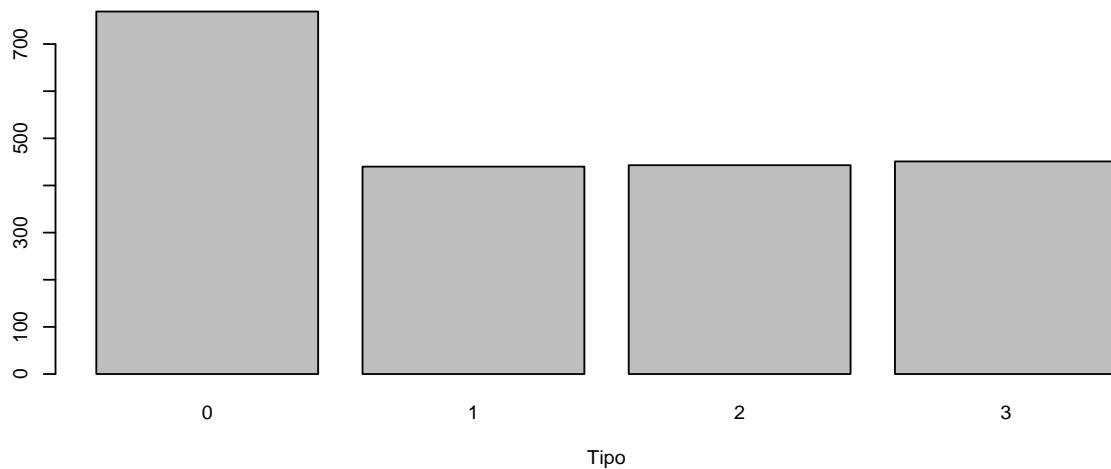
Centralità di base

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.

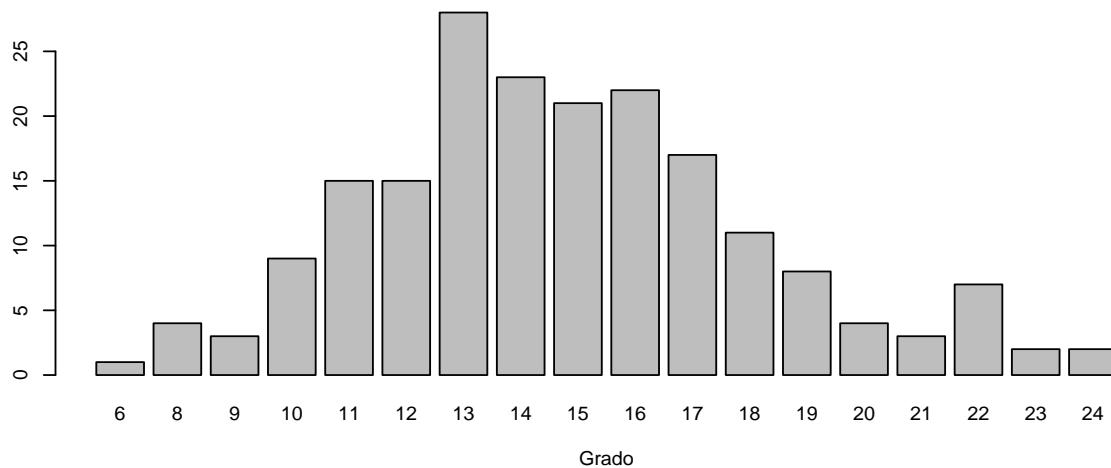
```
nodi_b = V(g)[type == 1]
nodi_w = V(g)[type == 0]
b_degrees <- degree(g, v = nodi_b)
w_degrees <- degree(g, v = nodi_w)
b_degree_counts <- table(b_degrees)
w_degree_counts <- table(w_degrees)
```

```
## $`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
```

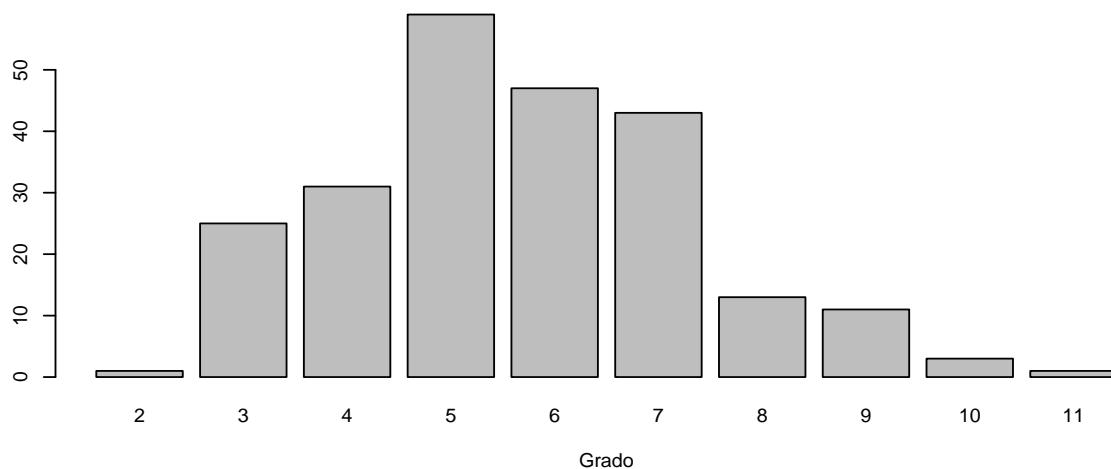
Archi per tipo



Distribuzione grado nodi bianchi

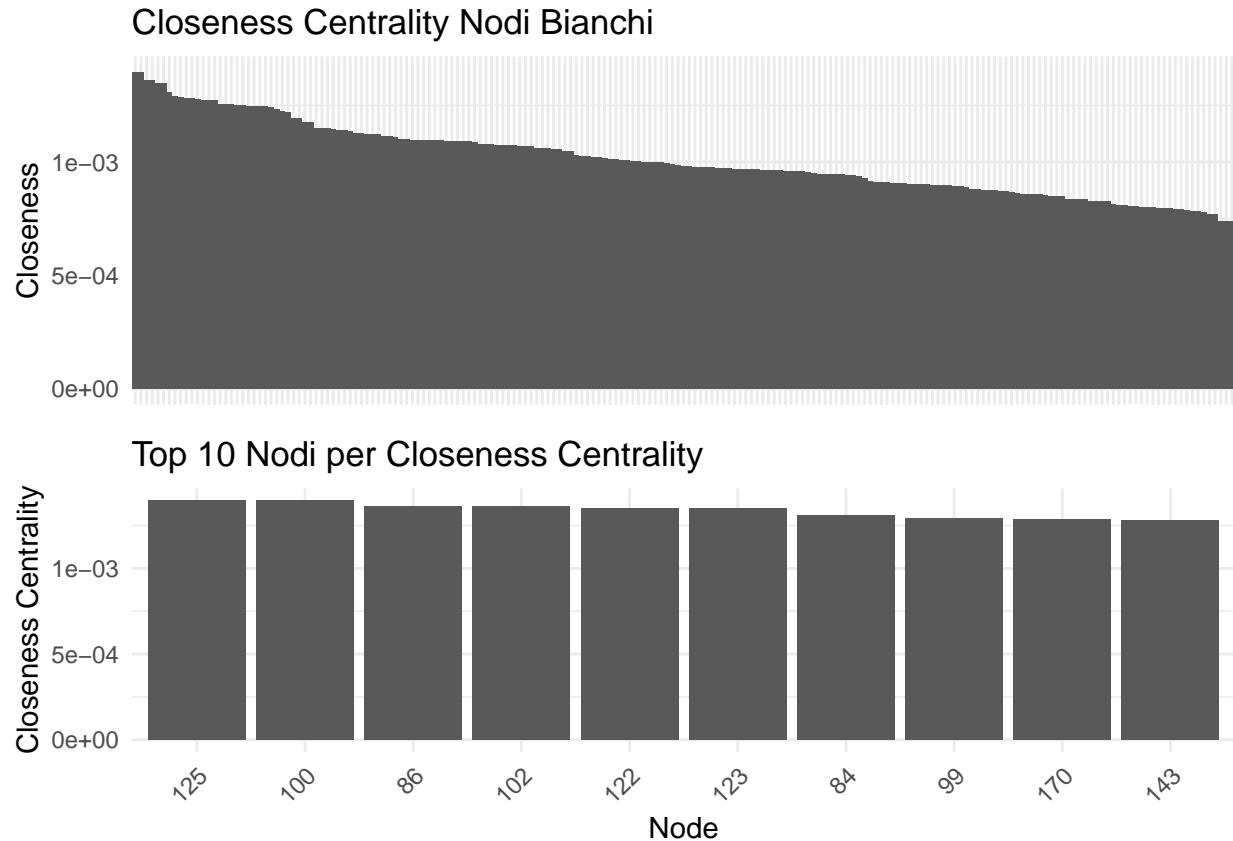


Distribuzione grado nodi neri



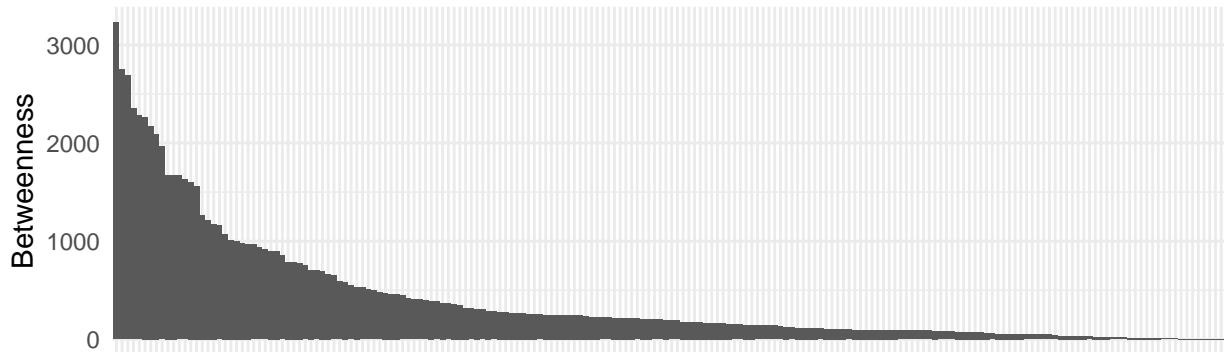
La distribuzione del grado per i nodi bianchi si avvicina maggiormente ad una gaussiana. Ha una coda destra leggermente pesante ma tutto sommato la distribuzione è normale. La distribuzione dei nodi neri invece è più sghemba. Proseguendo l'analisi con il calcolo delle centralità più comuni misureremo 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)$type == 0), delete.vertices = TRUE)
closeness_w <- closeness(g_w)
betweenness_w <- betweenness(g_w)
clo_w <- data.frame(
  Node = names(closeness_w),
  Closeness = closeness_w
)
bet_w <- data.frame(
  Node = names(betweenness_w),
  Betweenness = betweenness_w
)
# Do attenzione particolare ai nodi più centrali
clo_w <- clo_w[order(-clo_w$Closeness), ]
clo_w_top10 <- clo_w[1:10, ]
bet_w <- bet_w[order(-bet_w$Betweenness), ]
bet_w_top10 <- bet_w[1:10, ]
```

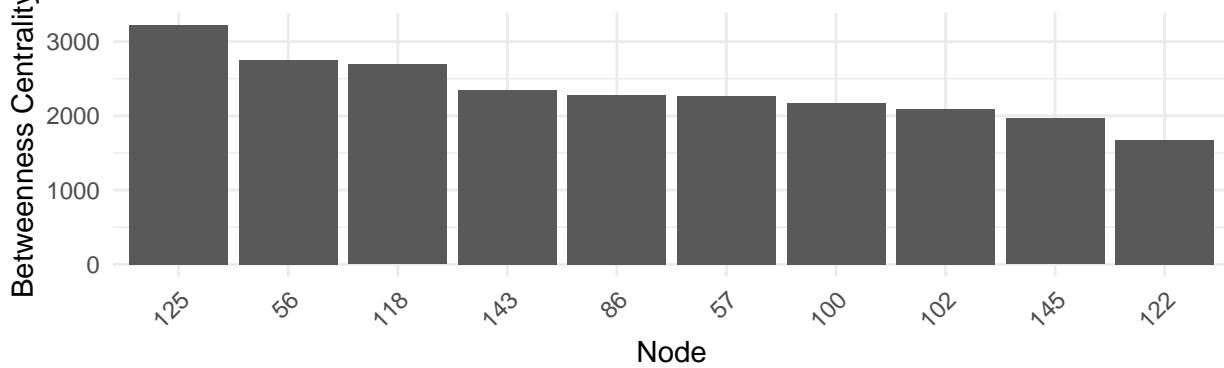


Com'è possibile notare, non sorprende che i nodi topologicamente più centrali secondo la disposizione dei nodi indotta dal tabellone, rappresentino anche i nodi con closeness più alta.

Betweenness Centrality Nodi Bianchi



Top 10 Nodi per Betweenness Centrality



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 poiché 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.

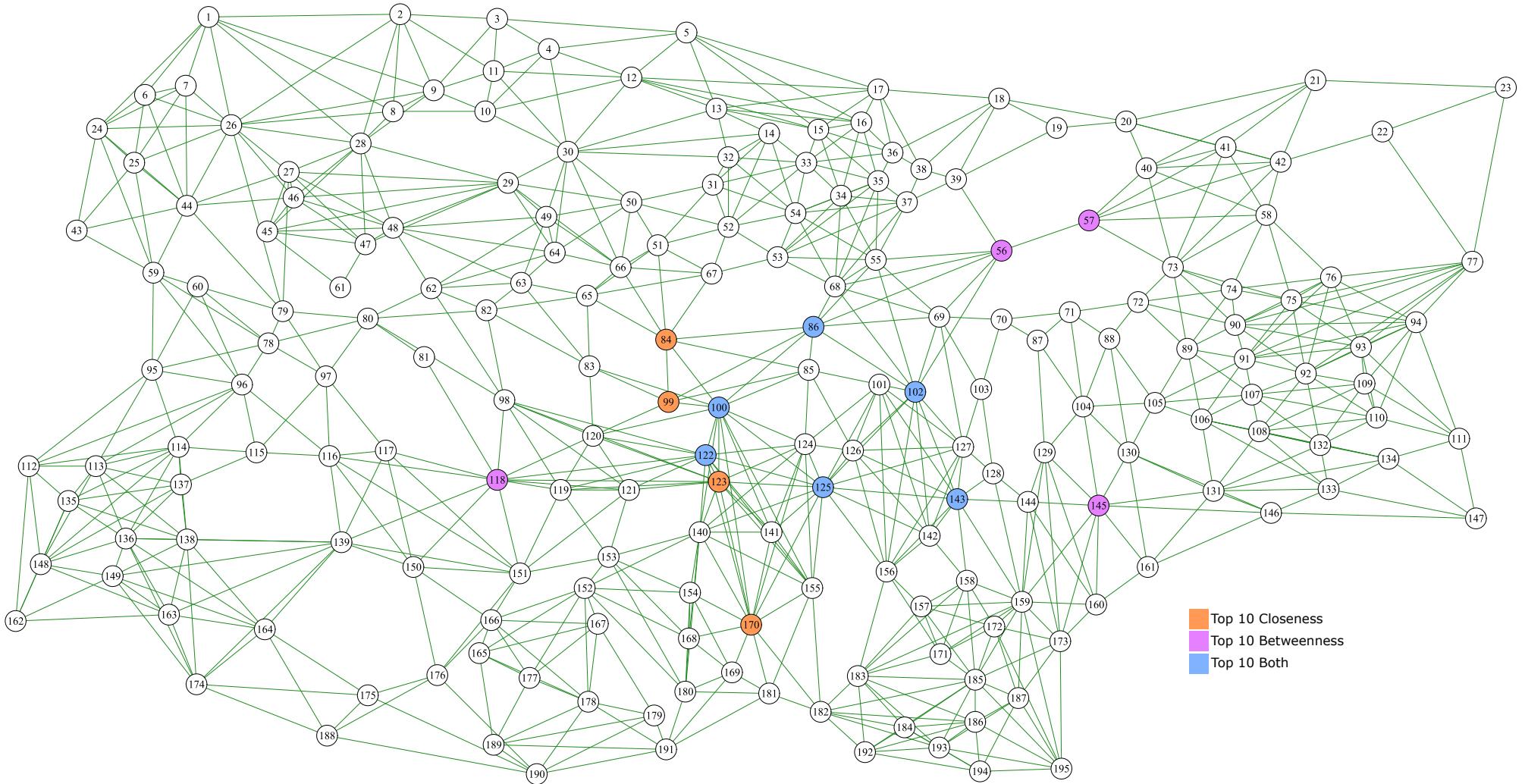
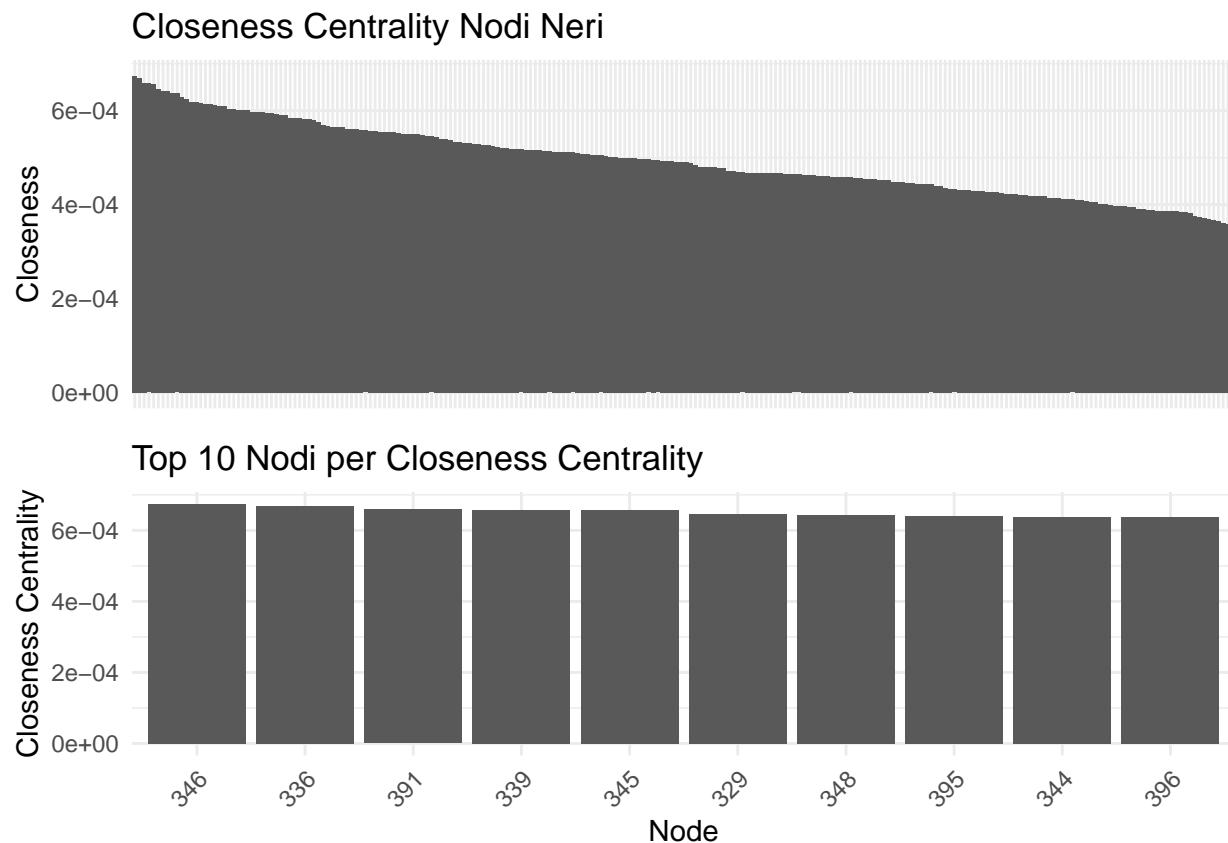
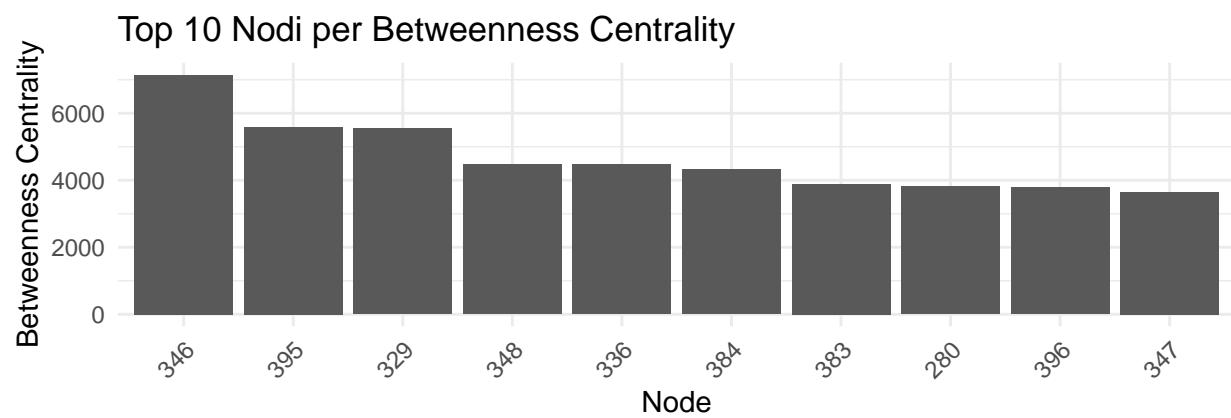
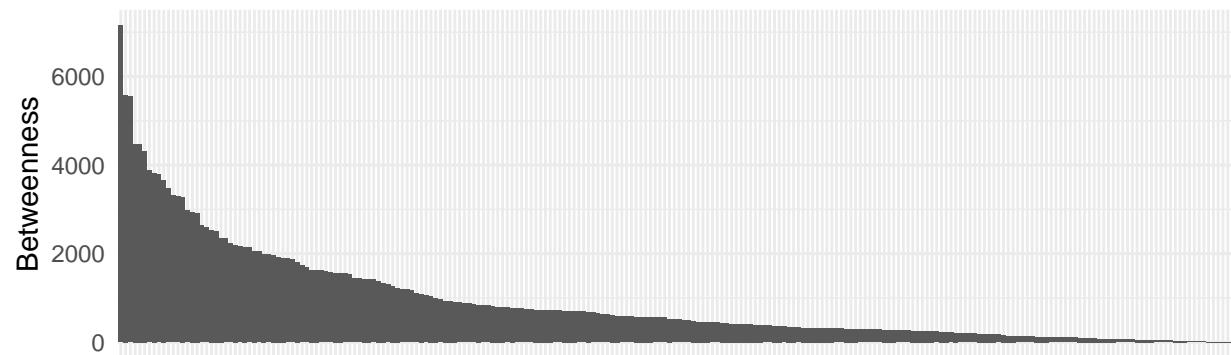


Figure 5: Centralità visualizzate sul grafo bianco.

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



Betweenness Centrality Nodi Neri



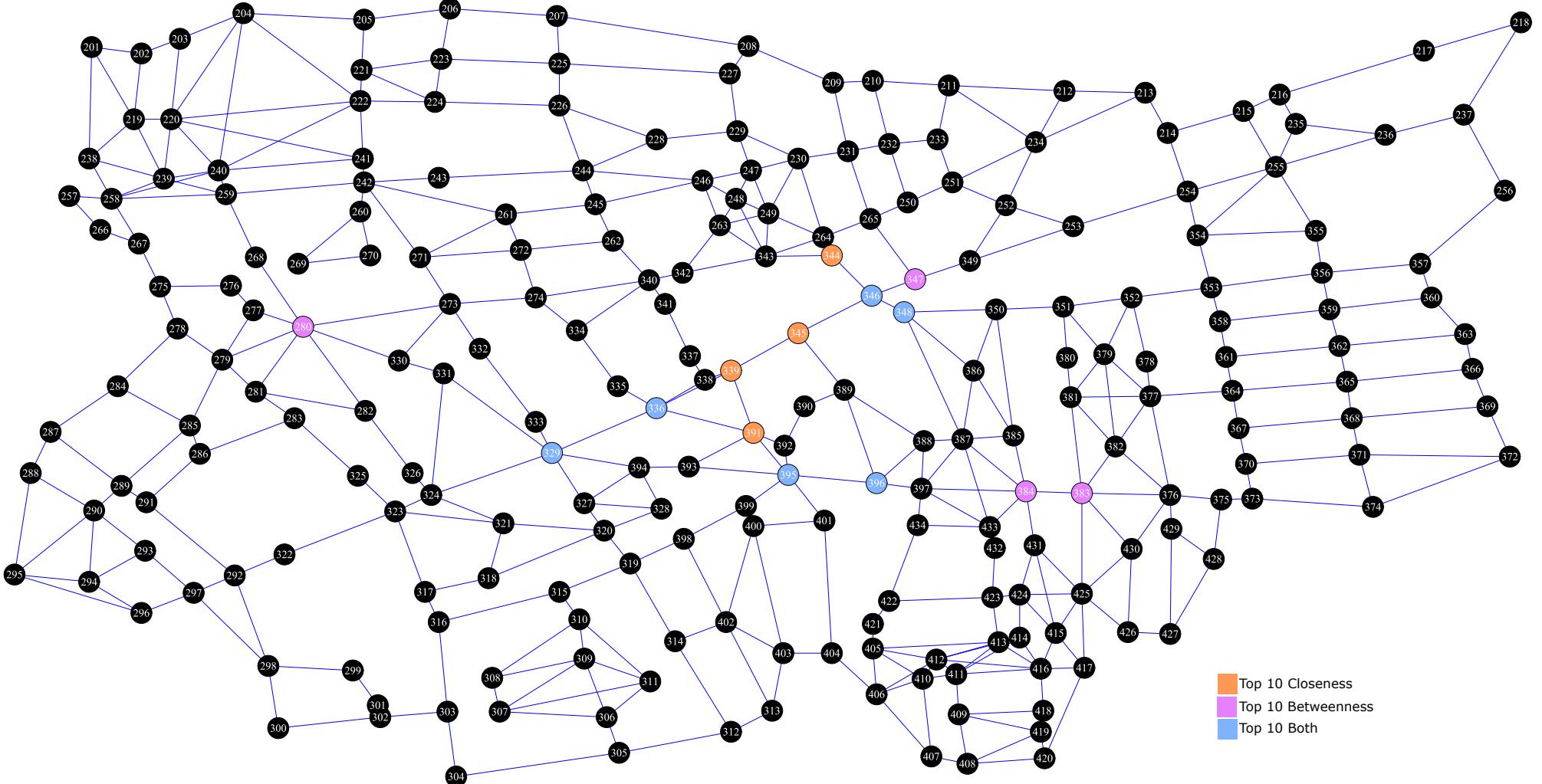


Figure 6: Centralità visualizzate sul grafo nero.

Differenza tra le due reti

Facendo una partita a “Lettere da Whitechapel” ci si rende conto presto che la velocità di movimento di Jack sembra molto più alta di quella della Polizia, anche se quest’ultima si muove con 5 pedine. E' possibile avere una riprova di questa sensazione guardando al diametro della rete bianca e quella nera:

```
#Diametro rete bianca  
diameter(g_w)
```

```
## [1] 12
```

```
#Diametro rete nera  
diameter(g_b)
```

```
## [1] 21
```

Il diametro della rete nera è quasi il doppio di quella bianca. Questo fatto contribuisce alla sensazione di lentezza delle pedine della Polizia anche se queste possono avanzare di due nodi alla volta.

Ricerca delle comunità

Dato che il tabellone di gioco suggerisce implicitamente un raggruppamento dei nodi in comunità, è interessante vedere se c’è riscontro con quelle che emergono dall’analisi degl’archi tramite due algoritmi per la ricerca di comunità come Louvain e Edge Betweenness. Risulta che le comunità trovate dagli algoritmi, in tutti e 2 i casi, rispecchiano molto le comunità che ci saltano all’occhio guardando il tabellone di gioco. Le differenze sono più che altro additabili alla granularità con cui queste sono state individuate.

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

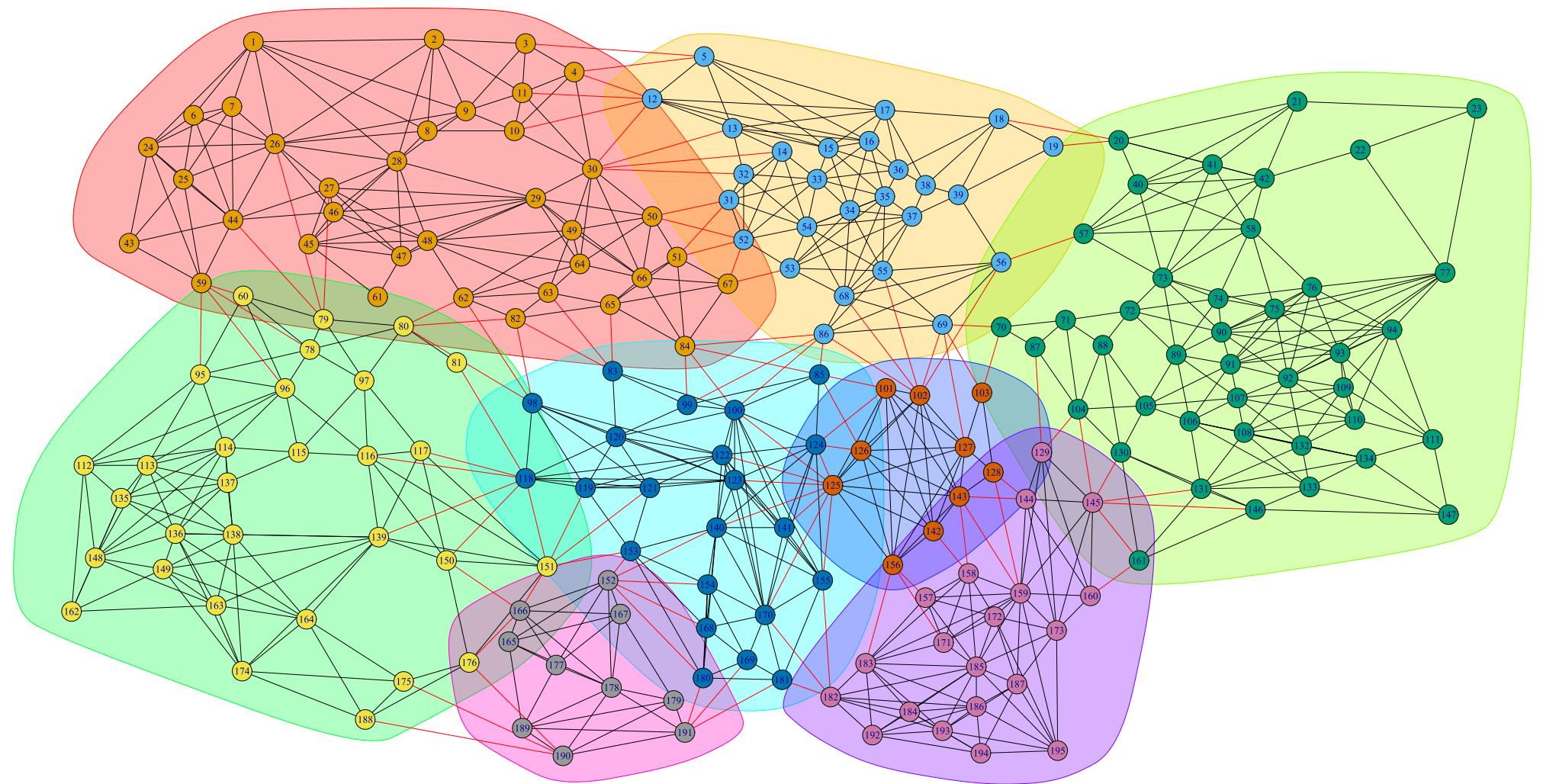


Figure 7: Community detection con algoritmo di Louvain.

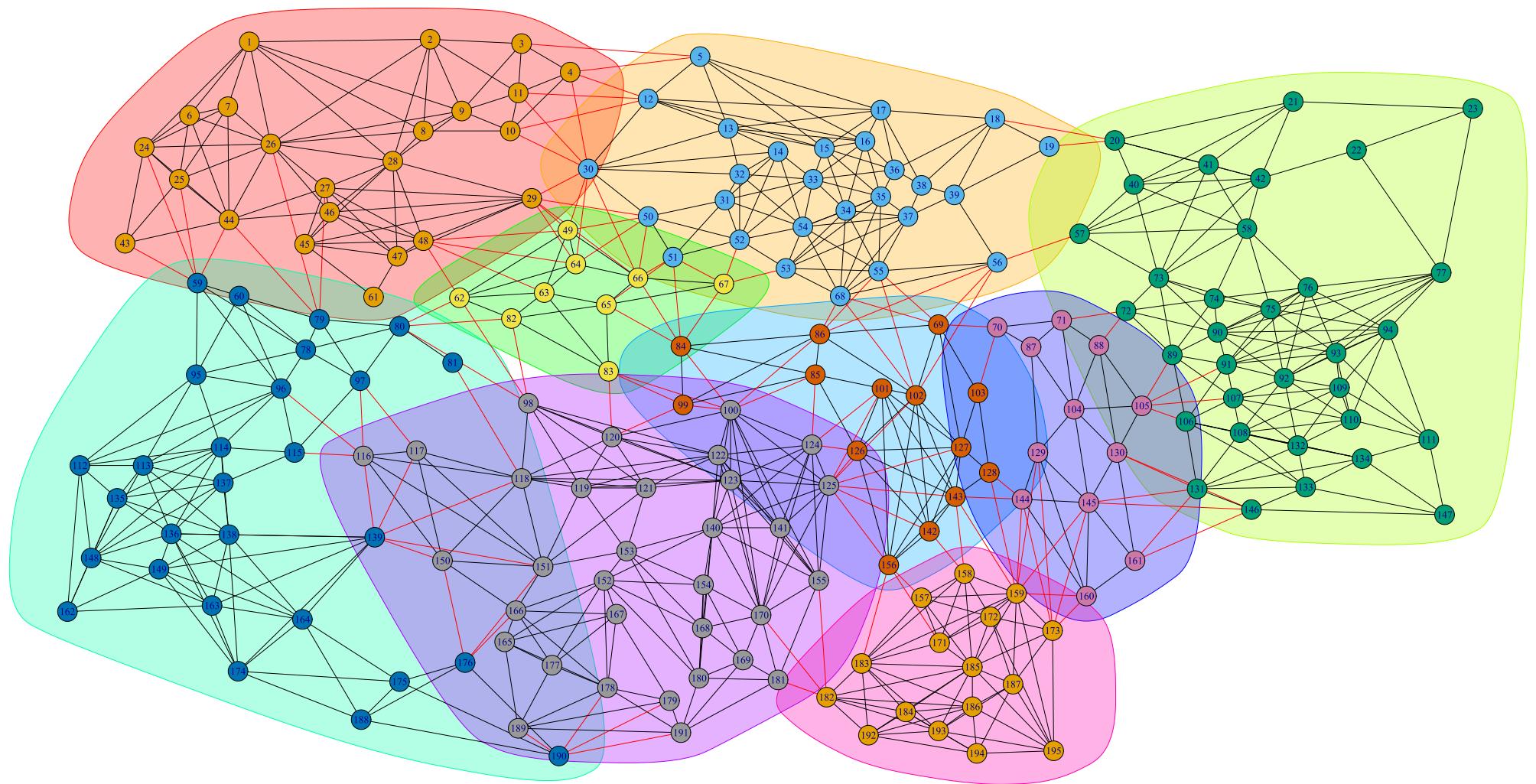


Figure 8: Community detection con algoritmo di Edge Betweenness.

Analisi: qual è il miglior nascondiglio per Jack?

Una domanda di ricerca molto saliente che suscita il dataset è: tramite la definizione di metriche quantitative, è possibile trovare 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 uniti in maniera adeguata per integrare i loro contributi.

Nota la dinamica dei turni di gioco, risulta immediato che i percorsi minimi siano di fondamentale importanza per trovare il nascondiglio ideale. Questi saranno al centro di diverse delle metriche utilizzate. Dato che 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. La destinazione, invece, 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: premiare i nodi centrali con un valore alto per le centralità classiche
- Fattore 2 - contributo della varietà dei percorsi: premiare i nodi che hanno percorsi più vari dalle scene del crimine, sia sui nodi che sugli archi.
- Fattore 3 - contributo della pattugliabilità: premiare i nodi che sono pattugliati da nodi neri poco importanti
- Fattore 4 - contributo rischio dei percorsi: premiare i nodi che hanno a disposizione percorsi minimi mediamente più difficili da pattugliare
- Fattore 5 - contributo del numero di percorsi: premiare i nodi che hanno più percorsi generici per raggiungere il nascondiglio dalla scena del crimine

Calcolo degli archi derivati

Fin ora abbiamo mostrato la rete bianca solamente con gli archi di tipo 0 associati ai movimenti generici non limitati. Detto questo, per il calcolo dei percorsi minimi è necessario avere a disposizione tutti gli archi che Jack può attraversare e questi includono anche quelli descritti dall'utilizzo di una carrozza. All'atto pratico, l'uso di una carrozza rappresenta l'esecuzione di due movimenti semplici in un unico turno di gioco ma è importante distinguerli in base al nodo intermedio attraversato. Per trovare tutti gli archi carrozza possiamo effettuare un inner join degli archi di tipo 0 con se stessi.

```
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) %>% # Scarta gli archi ciclici  
    transmute(  
      from = ifelse(from == to_1, to_2, to),  
      to = ifelse(to == to_1, from, to_2),  
      edge_type = ifelse(from == to_1, "carrozza", "pedata")  
    )  
}  
  
# Calcolo degli archi derivati  
edges %>% compute_all_edges(edge_type = "pedata") %>%  
  filter(edge_type == "pedata") %>%  
  select(-edge_type)
```

```

    from = from,
    step = to,
    to = to_2,
    type = edge_type
) %>%
arrange(from, to, step) %>%
distinct(.keep_all = TRUE) %>%
mutate(id = row_number() + max_id)

return(two_step_edges)
}

```

Il dataset contenente gli archi viene aggiornato per includere tutti gli archi corrispondenti a due movimenti semplici di fila. La stessa cosa viene fatta anche per la rete nera, poichè questi archi saranno necessari per il calcolo della pattugliabilità di un nodo.

```

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

Il computo dei percorsi minimi è particolarmente dispendioso in termini di tempo e la mole di dati generati è non indifferente. In particolare R non è adatto per svolgere questo tipo di computazioni perciò è stato necessario sviluppare uno script C++ apposito. Se nella cartella di lavoro sono presenti i file che contengono i percorsi, divisi per notte ti gioco, vengono utilizzati quest'ultimi. Altrimenti il codice C++ utilizzato per il calcolo dei percorsi minimi è presente nella cartella cppScripts: “pathComputing.exe”. Questo 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/pathComputing.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")

```

Viene finalmente creato il grafo completo della rete bianca che sarà utilizzato per il resto dell’analisi.

```

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

```

Funzioni di supporto

Seguono alcune funzioni di normalizzazione utilizzate nel calcolo delle metriche e anche una funzione per il calcolo della closeness. Quest'ultima è necessaria poichè in possesso di tutti i percorsi minimi rilevanti i quali sono stati calcolati secondo le regole di gioco. Inoltre la distanza tra un nodo candidato e un altro non è rilevante, contano solo quelle tra nodi candidati e scene del crimine, perciò solo queste distanze vengono utilizzate.

```

norm_max <- function(x) {
  ((x - mean(x)) / sd(x))
}

z_scale <- function(x) {
  z_normalized_data <- (x - mean(x)) / sd(x)
  scaled_data <- ((z_normalized_data - min(z_normalized_data)) /
    (max(z_normalized_data) - min(z_normalized_data))) * 100
}
}

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

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

```

Fattore 1: contributo delle centralità

Nello specifico, il punteggio di un nodo per il Fattore 1 è calcolato nel seguente modo:

- Vengono calcolate le centralità
 - Degree
 - Closeness (media per ogni notte di gioco)
 - Eigenvector
- Per ciascuna centralità viene calcolato
 - Valore normalizzato
 - Rango percentile
- Il punteggio del nodo è la media dei sei valori ottenuti (max 100)

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 <- as.numeric(V(g_w)[!V(g_w) %in% murder_sites])

```

E quindi vengono calcolate le centralità

```

degree_centrality <- degree(g_w)
degree_centrality = unname(degree_centrality[!names(degree_centrality) %in% murder_sites])
degree_norm <- z_scale(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$closeness <- rowMeans(c_centrality[,2:5])
c_centrality = c_centrality %>% filter(!node %in% murder_sites)
close_norm <- z_scale(c_centrality$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 <- z_scale(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 = as.numeric(hideouts),
  score = as.numeric(rowMeans(factor_1_raw[,2:7])))
)
partial_score = data.frame(
  node = factor_1_score$node,
  score = factor_1_score$score,
  Fattore1 = factor_1_score$score
)

```

Visualizziamo i punteggi relativi al primo fattore per i nodi bianchi. Non sorprende che i nodi che sono stati evidenziati come centrali nella fase esplorativa del dataset siano gli stessi che risultano particolarmente centrali quando aggiungiamo tutti gli archi legali del grafo.

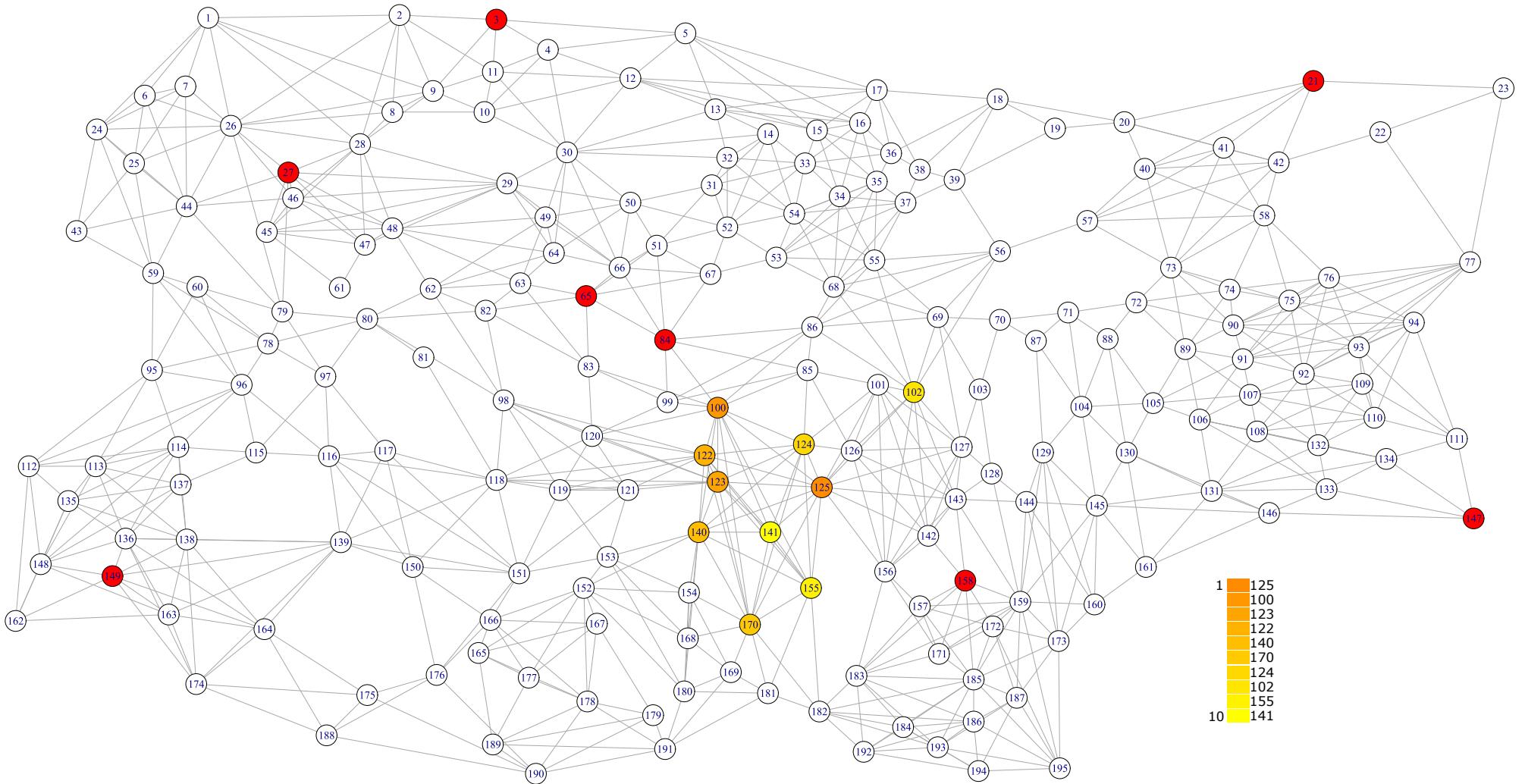


Figure 9: Il punteggio per il Fattore 1

Fattore 2:

Nello specifico il punteggio del Fattore 2 di un nodo è calcolato come di seguito:

- Per ogni coppia di percorsi minimi verso lo stesso nascondiglio vengono calcolate la *IoU* tra il vettore dei nodi e degli archi
 - $|IoU(v_n^i, v_n^j)|$
 - $|IoU(v_a^i, v_a^j)|$

- 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)

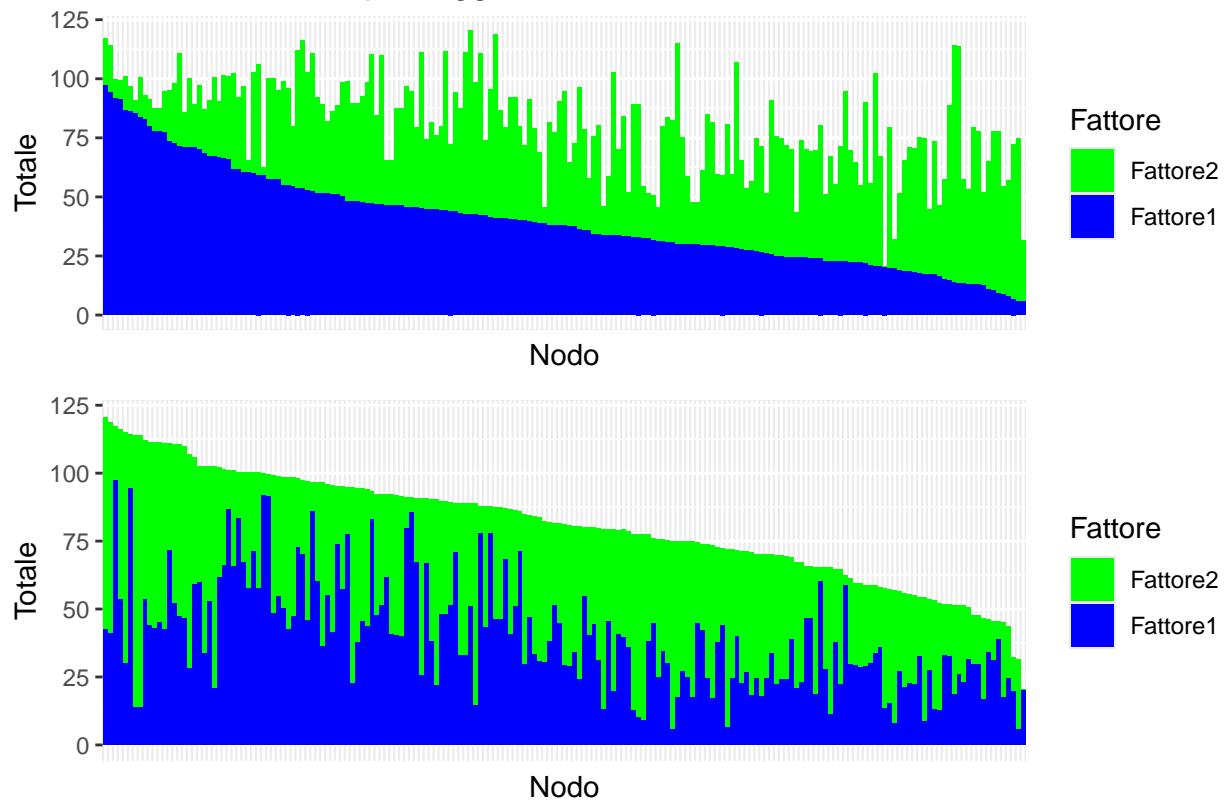
Dato che la computazione della similarità è una molto dispendiosa in termini di tempo, se è già presente nella cartella di lavoro il dataset relativo, la computazione viene saltata. Il codice C++ utilizzato per il calcolo della similarità di Jaccard è presente nella cartella cppScripts: “calcoloSimilarita.exe”.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

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

node_similarity_norm <- (100-z_scale(path_similarity$avg_node_iou))*(path_similarity$paths^(1/4))
edge_similarity_norm <- (100-z_scale(path_similarity$avg_edge_iou))*(path_similarity$paths^(1/4))
factor_2_raw <- data.frame(
  node = path_similarity$to,
  node_similarity_norm,
  edge_similarity_norm,
  z_scale(path_similarity$paths^(1/2)))
)

factor_2_score <- data.frame(
  node = path_similarity$to,
  score = z_scale(rowSums(factor_2_raw[, 2:4])))
)
```

Suddivisione del punteggio dei fattori



Risulta che i nodi sulla sinistra del tabellone presentano la maggior varietà di percorsi minimi.

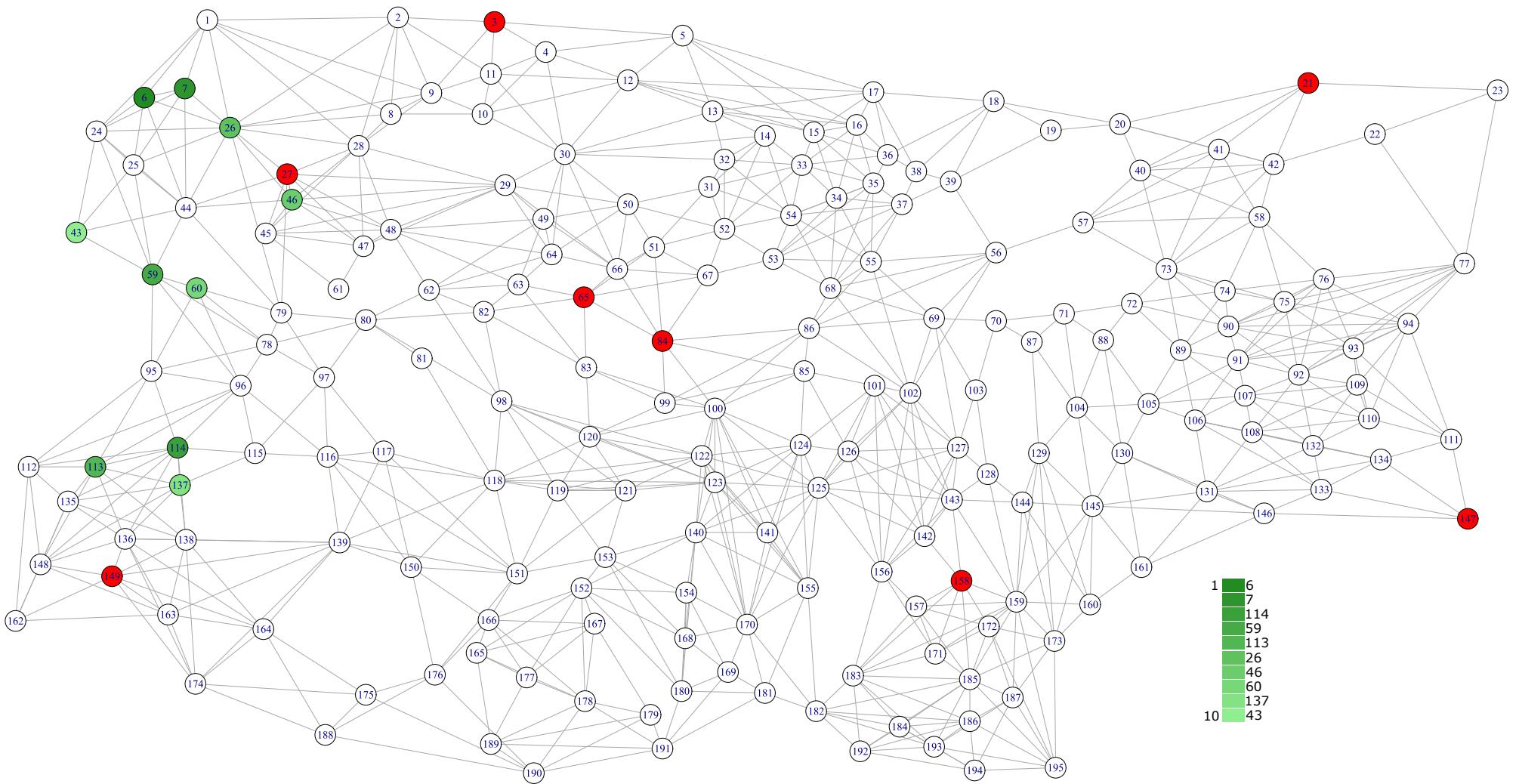


Figure 10: Il punteggio per il Fattore 2

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 di 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 hanno con gli altri nodi del proprio tipo e di quello opposto.

- Viene definita una metrica ricorsiva per quantificare l'importanza di nodi bianchi e neri nella rete di interazione
 - 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 neri importanti ($\delta = 0.4$)
- Per ogni nodo viene sommata l'importanza dei nodi neri che lo pattugliano
- Il vettore viene normalizzato in ordine decrescente (max 100)

La stessa metrica ricorsiva viene calcolata due volte utilizzando due matrici di adiacenza differenti:

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

```
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

# Numero dei nodi
num_black <- 234
num_white <- 195

# Inizializzo i vettori dell'importanza
black_power <- rep(1, num_black)
white_power <- rep(1, num_white)

# Criteri di alt e parametri della combinazione lineare
epsilon <- 1e-6
```

```

max_iter <- 100
diff <- Inf
iter <- 0
alpha <- 0.9
beta <- 0.1
gamma <- 0.6
delta <- 0.4

while (diff > epsilon && iter < max_iter) {
  iter <- iter + 1
  black_power_new <- norm_max(gamma*adj_bw%*%white_power + delta*adj_bb%*%black_power)
  white_power_new <- norm_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))
  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))

# Reset dei vettori dell'importanza
black_power <- rep(1, num_black)
white_power <- rep(1, num_white)

# Reset dei criteri di alt
diff <- Inf
iter <- 0

while (diff > epsilon && iter < max_iter) {
  iter <- iter + 1
  black_power_new <- norm_max(gamma*adj_bw%*%white_power + delta*adj_bb%*%black_power)
  white_power_new <- norm_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))
  black_power <- black_power_new
  white_power <- white_power_new
}
multi_edge = 100-(adj_wb%*% black_power/rowSums(adj_wb))

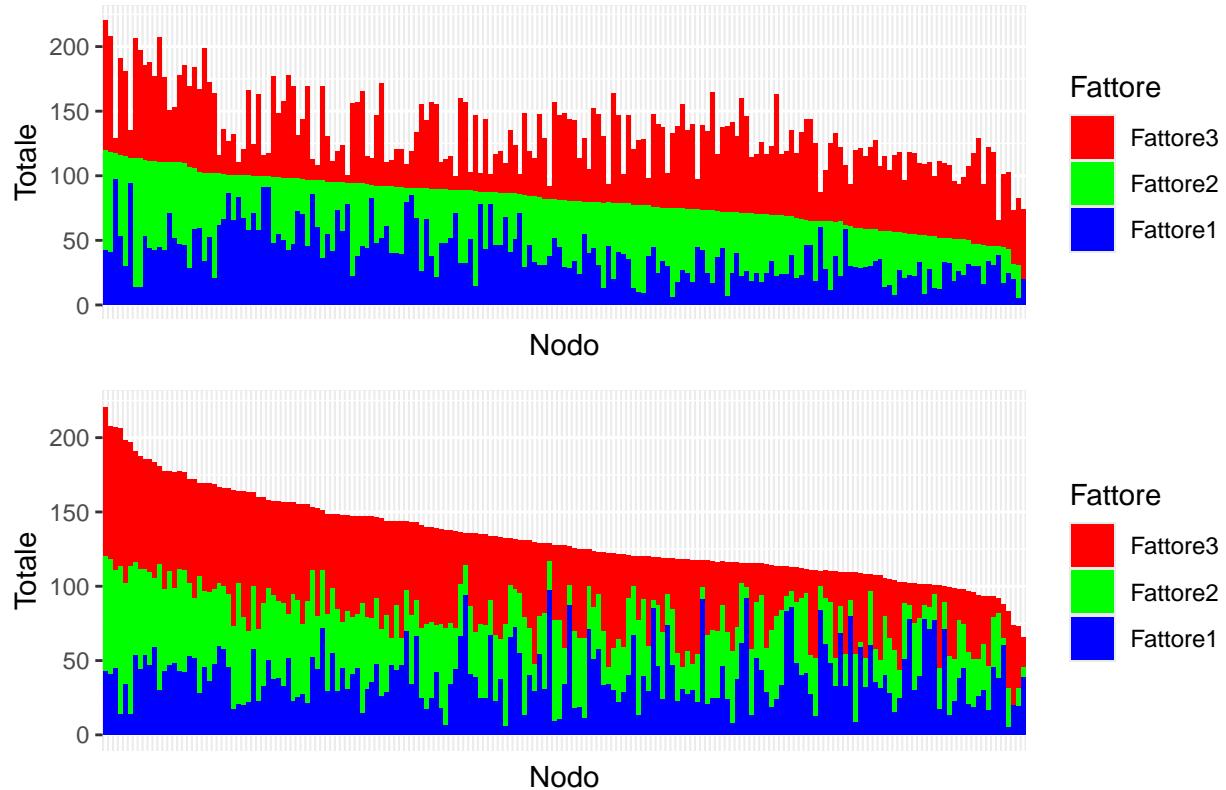
factor_3_raw <- data.frame(
  node <- nodi_w,
  norm = (z_scale(single_edge)+z_scale(multi_edge))/2,
  rank = (rank_percentiles(single_edge)+rank_percentiles(multi_edge))/2
)
factor_3_complete <- data.frame(
  node = as.integer(nodi_w),
  score = (factor_3_raw$norm+factor_3_raw$rank)/2
)

factor_3_score <- data.frame(
  node = hideouts,

```

```
    score = (factor_3_raw[~murder_sites,]$norm+factor_3_raw[~murder_sites,]$rank)/2
)
```

Suddivisione del punteggio dei fattori



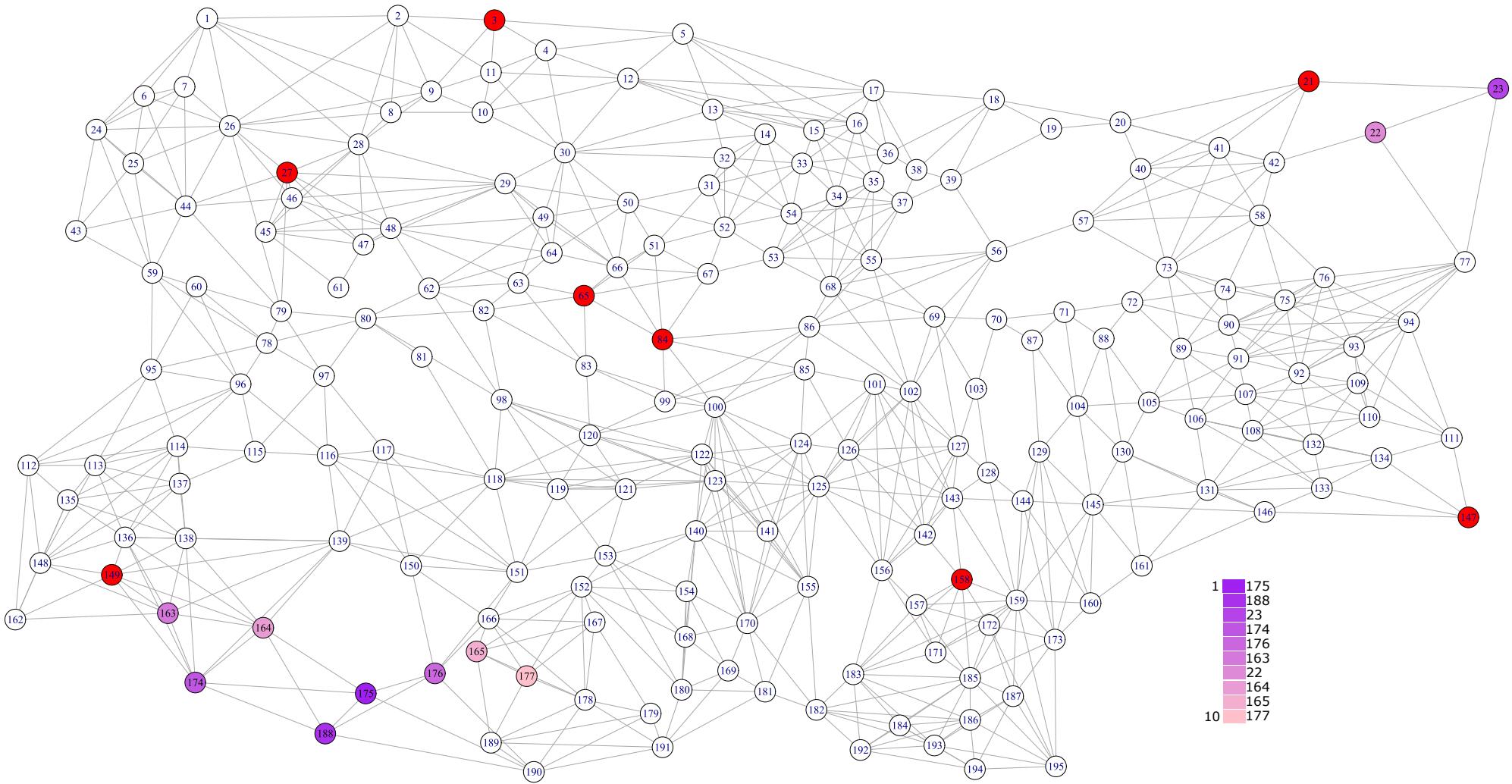


Figure 11: Il punteggio per il Fattore 3

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.

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 che devono essere risolti per includere anche il nodo perno. 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.

La mole di percorsi e la necessità di effettuare la decodifica dei nodi attraversati in maniera iterativa escludono la possibilità di utilizzare R per svolgere questo compito. Il codice C++ utilizzato per il calcolo della matrice binaria dei percorsi è presente nella cartella cppScripts: “pathMatrix.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”. Anche solo il caricamento di questa matrice può richiedere diverso tempo (fino ad alcuni minuti).

In dettaglio il punteggio del Fattore 4 è così costruito:

- 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 (z_norm e rango-percentile)
- La somma di questi due valori è il punteggio del nodo (max 100)

```
## [1] "Inizio il caricamento del dataset"

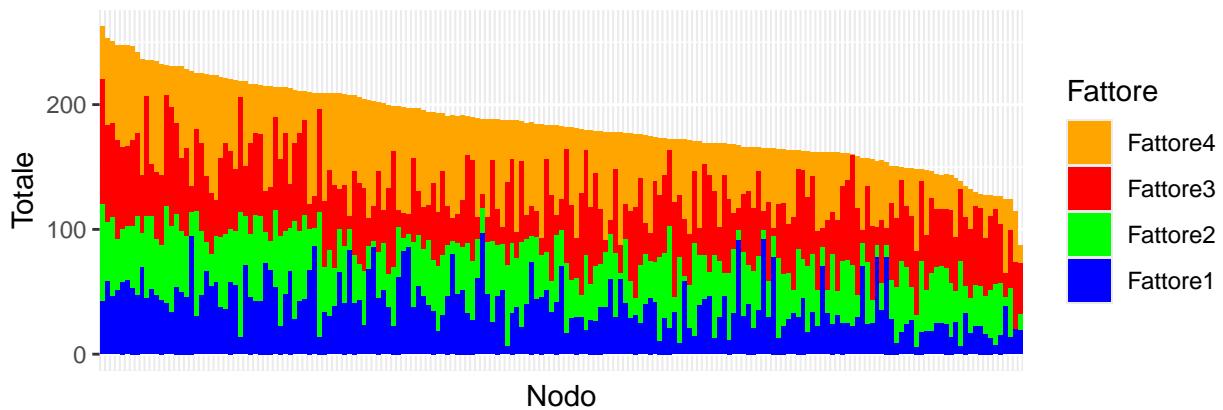
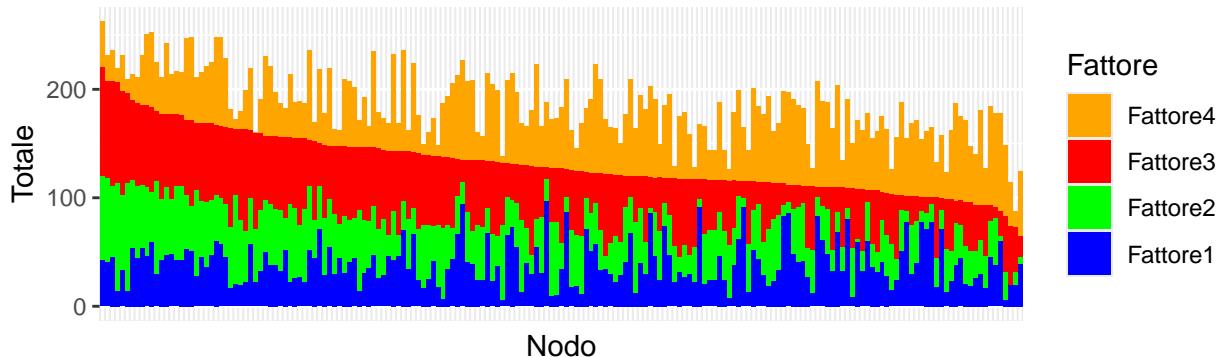
path_matrix <- as.matrix(path_matrix_data %>% arrange(node))
# Somma del punteggio di pattugliabilità dei nodi di ciascun percorso
path_policing_vec <- path_matrix[,3:197] %*% (factor_3_complete$score)
# Nodo target di ciascun percorso
path_nodes <- as.data.frame(path_matrix[,1])
colnames(path_nodes) <- c("node")
# Punteggio di pattugliabilità del nodo target necessario per normalizzare
norm_factor = (path_nodes %>% left_join( factor_3_complete, by="node"))
# Vettore del punteggio di pattugliabilità del percorso normalizzato per nodo target
norm_path_pol_vec <- (path_policing_vec - norm_factor$score)
path_policing_frame <- data.frame(
  to = path_matrix[,1],
  cost = path_matrix[,2],
  policing = norm_path_pol_vec
)
path_policing_score <- path_policing_frame %>% group_by(to) %>%
  summarize(
    length = sum(cost),
```

```

    count = n(),
    policing_score = sum(policing)/count,
    .groups = "drop"
)
path_policing_score = path_policing_score[-murder_sites,]
norm_4 = 100-z_scale(path_policing_score$policing_score)
rank_4 = 100-rank_percentiles(path_policing_score$policing_score)
test = norm_4+rank_4
factor_4_score = data.frame(
  node = hideouts,
  score = z_scale((norm_4+rank_4)/2)
);

```

Suddivisione del punteggio dei fattori



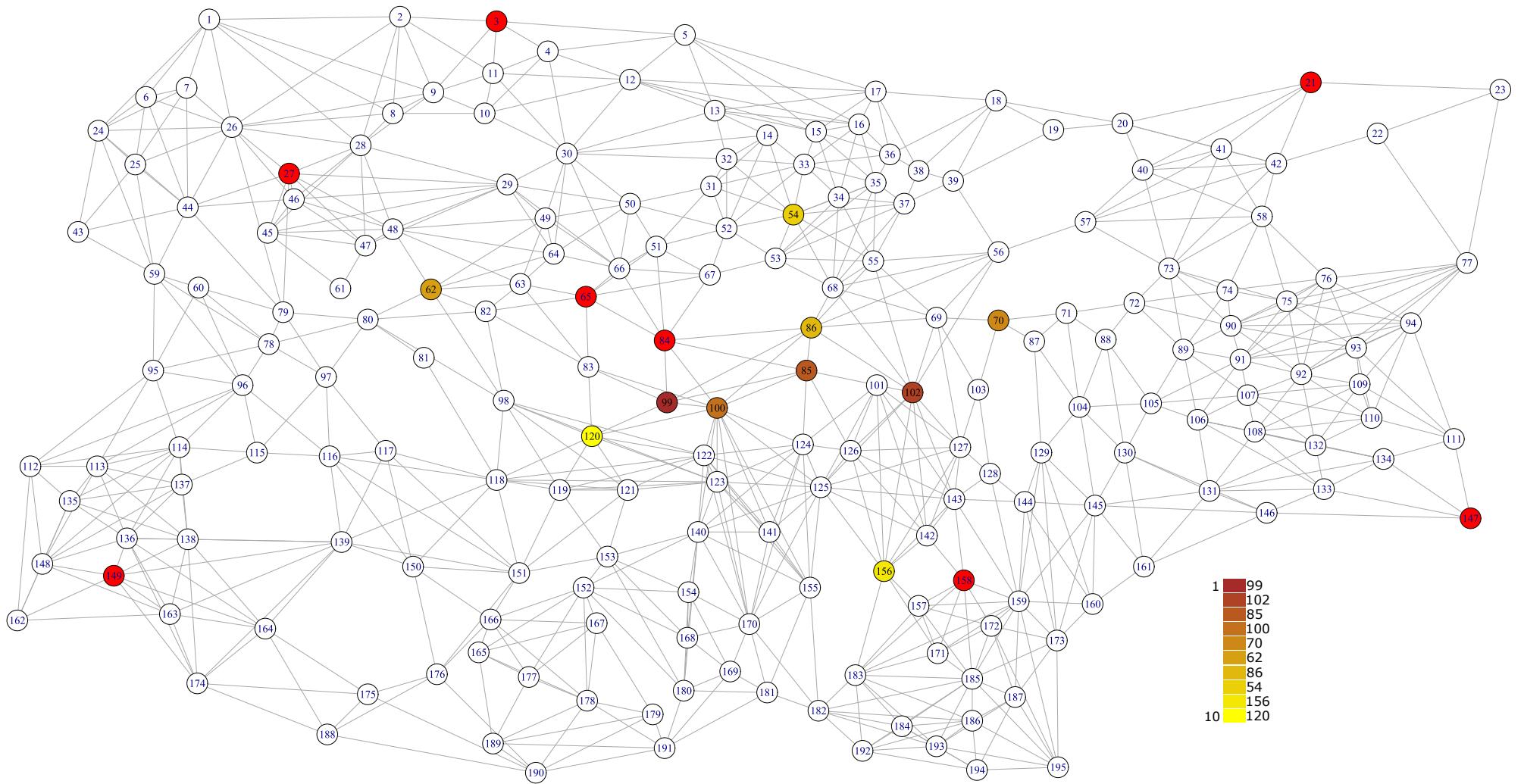


Figure 12: Il punteggio per il Fattore 4

Fattore 5 - Contributo del numero totale di percorsi

Fin'ora sono stati utilizzati primariamente i percorsi minimi per valutare la desiderabilità di un nodo. Detto questo, non è sufficiente considerare solamente questi percorsi poichè determinate configurazioni delle pedine della Polizia potrebbero rendere impossibile il loro attraversamento. Inoltre, soprattutto per le notti di gioco 3 e 4 è fondamentale avere a disposizione percorsi estremamente brevi dalla scena del crimine al nascondiglio. Questo perchè questi ultimi turni di gioco rappresentano l'ultima chance per la Polizia di acchiappare Jack.

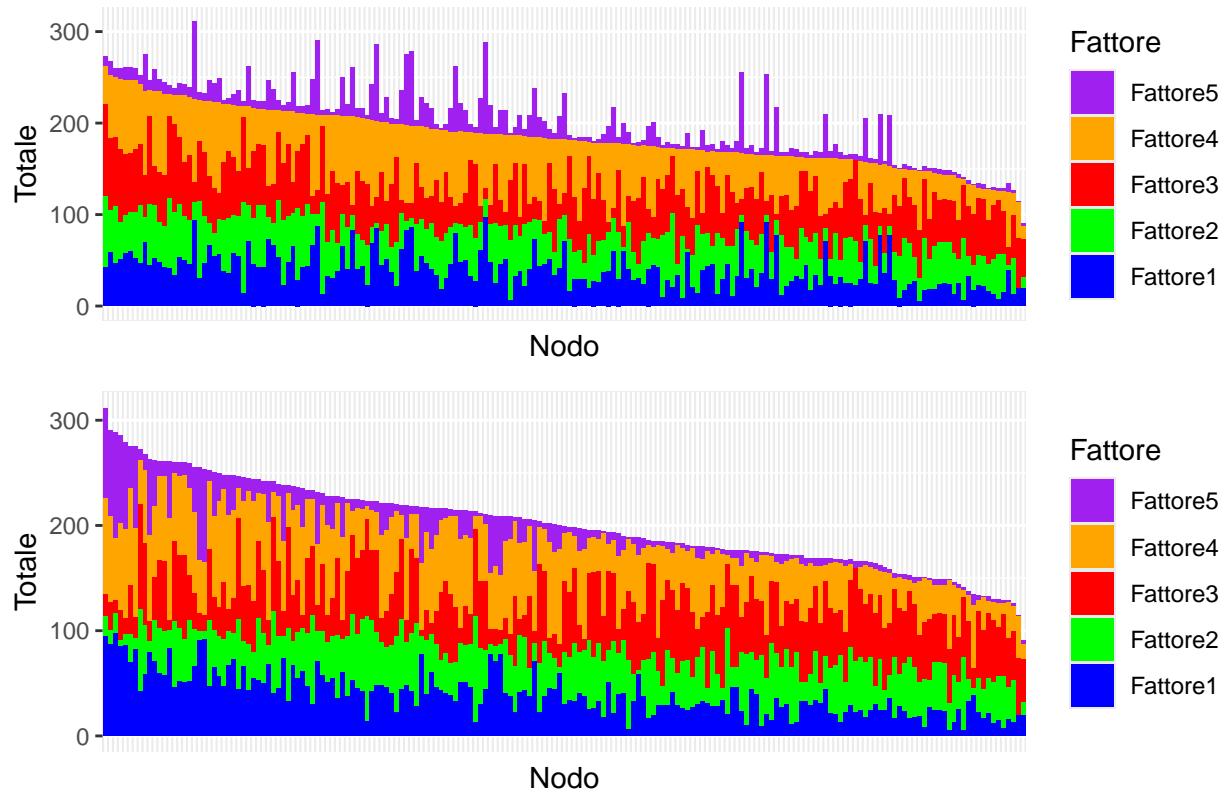
Per questo motivo viene definito un fattore che conta il numero di percorsi (non necessariamente minimi) che possono essere intrapresi dai luoghi del crimine ai nascondigli. Sono stati inclusi quelli di lunghezza uguale o inferiore a 4 poichè l'esperienza di gioco indica che questo è un limite massimo di tempo che si è disposti a circolare nelle ultime notti di gioco per massimizzare le chance di vittoria.

Il punteggio del Fattore 5 è così definito:

- Viene utilizzata la matrice di adiacenza dei soli archi semplici perchè non è facile impostare il limite massimo di movimenti speciali da utilizzare
- Vengono computate le potenze della matrice di adiacenza in maniera iterativa
- Ad ogni step vengono sommati i percorsi da ciascun nascondiglio ai luoghi del crimine
- La media di questi due valori viene normalizzata e costituisce il punteggio del nodo (max 100)

```
paths = matrix(data= NA,nrow = 195, ncol = 4)
adj = as.matrix(as_adjacency_matrix(g_w))
for(i in 1:4){
  paths[,i] = rowSums(adj[,murder_sites])
  adj = adj %*% adj
}
mean_paths = z_scale(rowMeans(paths))
factor_5_score = data.frame(
  node = factor_1_score$node,
  score = mean_paths[-murder_sites]
)
```

Suddivisione del punteggio dei fattori



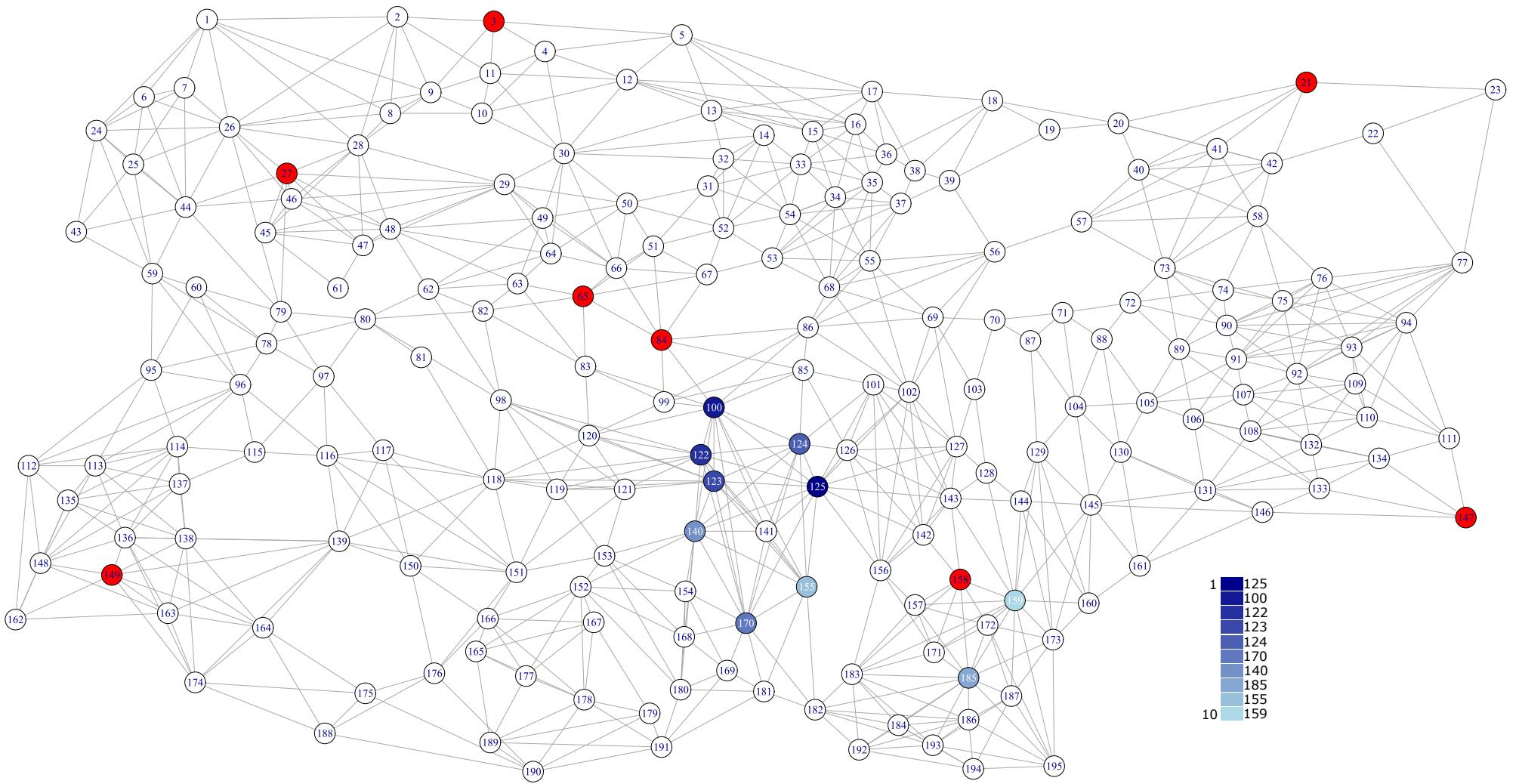


Figure 13: Il punteggio per il Fattore 5

Conclusioni

Dopo aver calcolato il punteggio di ogni fattore per ogni nodo del grafo è possibile combinare i punteggi utilizzando la media geometrica per avere un quadro globale su quali dei nodi risultino dei candidati più validi come nascondigli. La media geometrica è indicata perché particolarmente adatta per il confronto in termini relativi.

In conclusione risulta che tra i nodi più indicati per Jack come nascondiglio ci sono alcuni tra i nodi più centrali ma non i più potenti secondo le metriche classiche. Inoltre risulta che alcuni nodi vicini a nodi potenti ma abbastanza lontani dalle vie di passaggio più importanti per la Polizia si sono rivelati particolarmente validi.

```
geom = 4/(1/partial_score$Fattore1+1/partial_score$Fattore2+
           1/partial_score$Fattore3+1/partial_score$Fattore4+1/partial_score$Fattore5)
best_hideout = data.frame(
  node = partial_score$node,
  score = geom
)
best_hideout <- best_hideout[order(-best_hideout$score), ]
best_hideout_top10 <- best_hideout[1:10,]
best_hideout_bottom10 <- tail(best_hideout,10)
```

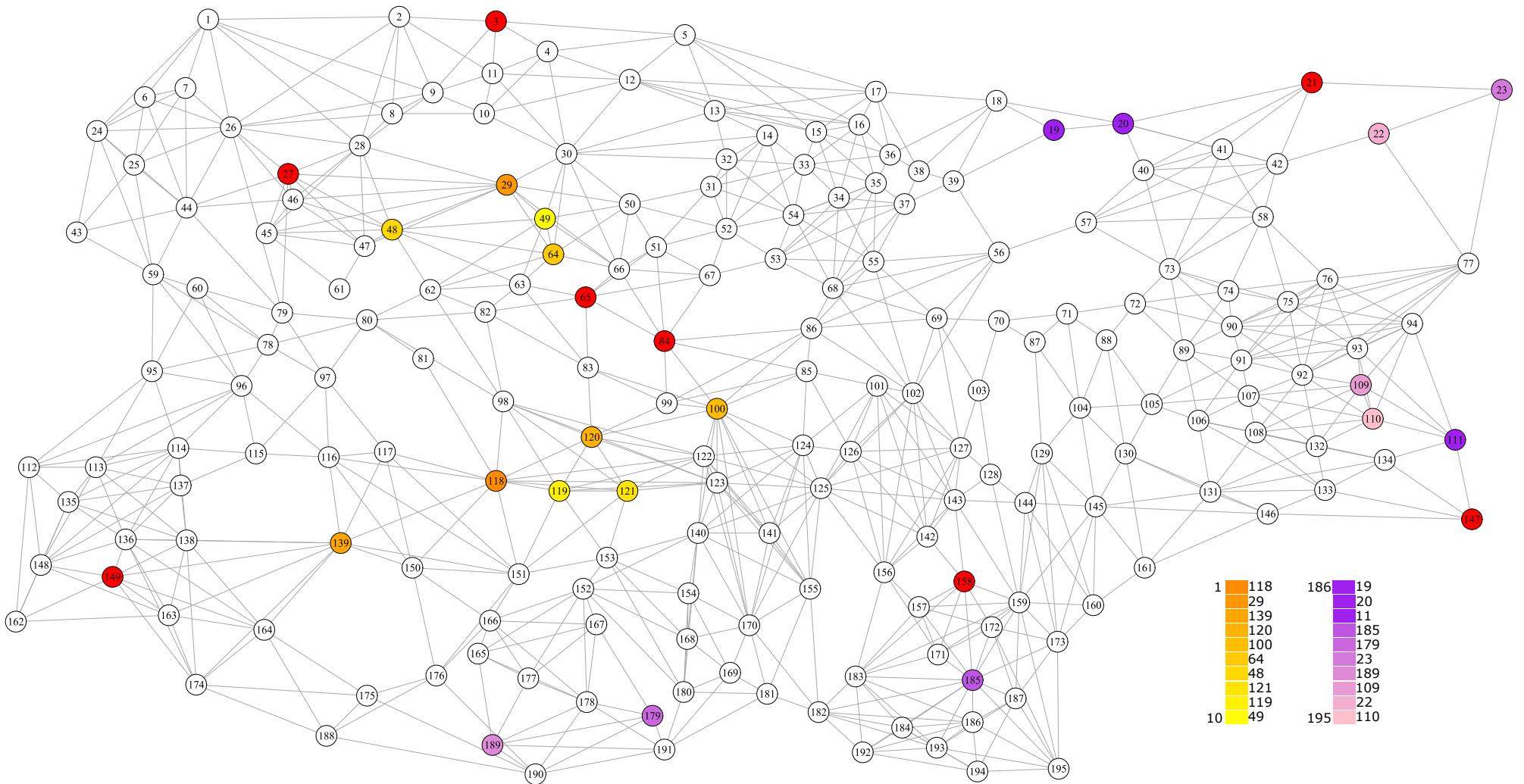


Figure 14: Il risultato dell'analisi