



**UNIVERSITÀ DEGLI STUDI DI CATANIA**  
DIPARTIMENTO DI MATEMATICA E INFORMATICA  
CORSO DI LAUREA TRIENNALE IN INFORMATICA

---

*Salvatore Alfio Sambataro*

FlightAnalyzer: app in R per l'analisi della rete  
aeroporuale mondiale

---

RELAZIONE PROGETTO FINALE

---

Relatore: Prof. Giovanni Micale

---

Anno Accademico 2022 - 2023

*Ai miei genitori, fonte di sostegno e di coraggio nei momenti più  
difficili, che hanno creduto in me sin dal primo momento.  
Senza di loro nulla sarebbe stato possibile*

# **Abstract**

Il presente lavoro di tesi consiste nella realizzazione di un'applicazione interattiva, sviluppata in linguaggio R tramite la libreria Shiny, che ha come obiettivo quello di permettere all'utente di effettuare analisi riguardanti la rete aeroportuale mondiale, la quale può essere vista sotto la veste di un grafo, in cui i nodi rappresentano i singoli aeroporti, mentre gli archi rappresentano le singole tratte esistenti tra i diversi aeroporti. Tra le funzionalità più importanti offerte dall'applicazione abbiamo: il calcolo di diverse misure di centralità del grafo che rappresenta la rete, in maniera tale da individuare quali sono gli aeroporti più importanti; la visualizzazione dei dati relativi alla rete su grafici e mappe interattive; il calcolo del numero di eventuali triangoli. La funzionalità più importante dell'applicazione è la "ricerca di motivi": tale funzionalità permette all'utente di "disegnare" uno o più motivi, ricercare questi ultimi all'interno della rete e stamparli su un globo 3D interattivo. L'obiettivo principale di questa funzionalità è quello di individuare e visualizzare eventuali pattern ricorrenti nella rete.

# Indice

<b>1</b>	<b>Introduzione</b>	<b>5</b>
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Network Science . . . . .	7
2.2	Reti o grafi . . . . .	8
2.2.1	Tipi di grafi . . . . .	8
2.2.1.1	Grafi diretti e indiretti . . . . .	8
2.2.1.2	Grafi etichettati e pesati . . . . .	9
2.2.1.3	Grafi bipartiti e multipartiti . . . . .	9
2.2.1.4	Proiezioni di un grafo bipartito . . . . .	10
2.2.1.5	Grafi completi . . . . .	11
2.2.1.6	Grafi regolari . . . . .	11
2.2.2	Adiacenza tra nodi e grado di un nodo . . . . .	12
2.2.3	Cammini tra due nodi e distanza . . . . .	13
2.2.4	Sottografi orientati e non orientati . . . . .	14
2.3	Graph mining . . . . .	15
2.4	Frequent Subgraph Mining . . . . .	15
2.4.1	Algoritmi di Frequent Subgraph Mining . . . . .	16
2.4.1.1	Generazione dei candidati . . . . .	16
2.4.1.2	Pruning delle ridondanze . . . . .	18
2.4.1.3	Conteggio . . . . .	18
2.4.2	Significatività statistica . . . . .	19
2.5	Caratteristiche delle reti reali . . . . .	20
2.5.1	Proprietà "Small World" . . . . .	20
2.5.2	Reti scale-free e legge "power-law" . . . . .	21
2.5.2.1	Robustezza di una rete scale-free . . . . .	22
<b>3</b>	<b>Dataset "OpenFlights"</b>	<b>23</b>
3.1	Struttura del dataset . . . . .	23
3.1.1	Airport Database . . . . .	24
3.1.2	Airline Database . . . . .	25

3.1.3	Countries Database . . . . .	25
3.1.4	Routes Database . . . . .	26
3.1.5	Planes Database . . . . .	27
<b>4</b>	<b>Metodologie di lavoro</b>	<b>28</b>
4.1	Modellazione dei dati di OpenFlights come un multigrafo . . . . .	28
4.2	Linguaggio R e librerie utilizzate . . . . .	29
4.3	Formato dei dati . . . . .	30
4.4	Caricamento del dataset in R . . . . .	30
4.5	Fase di Pre-processing . . . . .	31
4.6	Creazione del grafo . . . . .	32
4.7	Assegnamento dei pesi agli archi . . . . .	33
4.8	Scrittura del grafo su file . . . . .	34
<b>5</b>	<b>Struttura dell'applicazione</b>	<b>36</b>
5.1	Struttura di un'applicazione Shiny . . . . .	37
5.1.1	User Interface . . . . .	37
5.1.2	Server . . . . .	38
<b>6</b>	<b>Funzionalità dell'applicazione</b>	<b>40</b>
6.1	Dashboard . . . . .	40
6.1.1	Statistiche della rete . . . . .	41
6.1.1.1	Nodi e archi . . . . .	41
6.1.1.2	Assortatività . . . . .	41
6.1.1.3	Triangoli . . . . .	42
6.1.1.4	Coefficiente di clustering . . . . .	42
6.1.1.5	Distanza media . . . . .	43
6.1.1.6	Diametro . . . . .	43
6.1.2	Tipologie di tratte . . . . .	44
6.1.2.1	Suddivisione delle tratte . . . . .	44
6.1.3	Grafici vari . . . . .	45
6.1.4	Distribuzione dei gradi . . . . .	47
6.2	Ricerca di aeroporti . . . . .	49
6.3	Visualizzazione 3D della rete . . . . .	52
6.4	Misure di centralità . . . . .	55
6.4.1	Degree Centrality . . . . .	55
6.4.2	Betweenness Centrality . . . . .	56
6.4.3	Closeness Centrality . . . . .	57
6.4.4	PageRank Centrality . . . . .	58
6.4.5	Implementazione del calcolo delle misure di centralità . . . . .	59
6.5	Ricerca di motivi . . . . .	62

6.5.1	Significatività statistica nel mining di un singolo grafo . . . . .	64
6.5.2	Strategie di ricerca dei motivi . . . . .	64
6.5.3	Algoritmo "MultiMotif" . . . . .	65
6.5.3.1	Modello "Expected Degree Distribution" per multigrafi etichettati . . . . .	65
6.5.3.2	Varianti del subgraph matching: ESM e MSM	66
6.5.3.3	Probabilità di occorrenza esatta del motivo .	67
6.5.3.4	Media e varianza del conteggio di motivi . . .	67
6.5.3.5	Calcolo del p-value . . . . .	67
6.5.4	Implementazione della ricerca di motivi con MultiMotif	69
6.5.4.1	Formato della rete da analizzare . . . . .	69
6.5.4.2	Formato delle query . . . . .	70
6.5.4.3	Creazione delle query . . . . .	72
6.5.4.4	Esecuzione di MultiMotif . . . . .	74
6.5.4.5	Formato dei risultati . . . . .	76
6.5.4.6	Visualizzazione dei motivi . . . . .	76
<b>Conclusione</b>		<b>79</b>
<b>Appendice: screenshot dell'applicazione FlightAnalyzer</b>		<b>80</b>
<b>Bibliografia</b>		<b>88</b>

# Capitolo 1

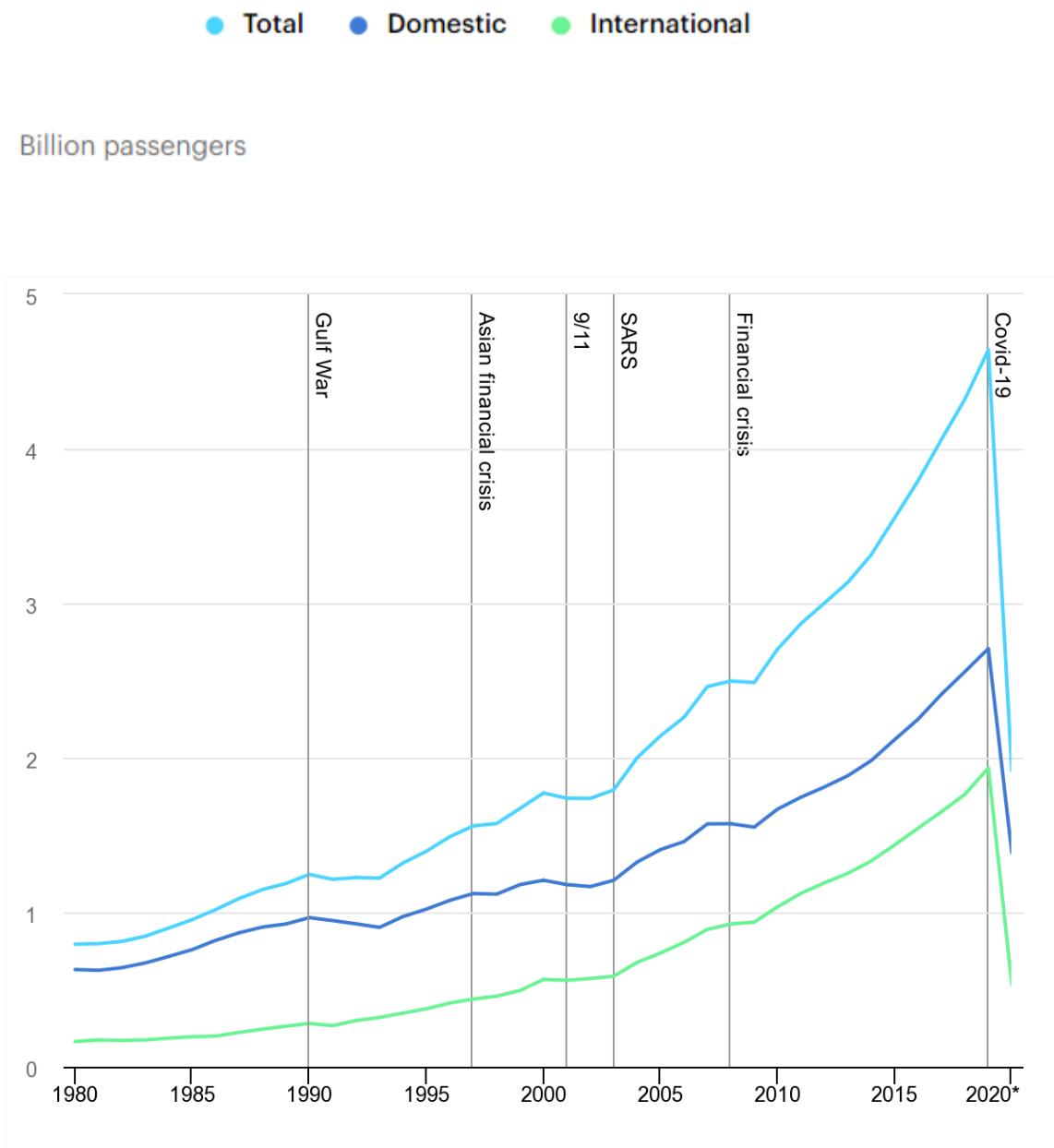
## Introduzione

Nell'ultimo decennio, la crescita del settore dell'aviazione è stata straordinaria. Numerosi fattori, tra cui l'apertura di nuovi mercati, l'aumento delle compagnie aeree a basso costo, l'innovazione tecnologica e il miglioramento dell'efficienza operativa, hanno contribuito a rendere i viaggi aerei più accessibili e convenienti per un numero sempre maggiore di persone. Questo ha comportato un aumento significativo del numero di passeggeri che utilizzano i servizi aeroportuali e ha posto sfide senza precedenti per l'industria.

Analizzare e comprendere la struttura e il funzionamento della rete aeroportuale può quindi fornire informazioni preziose per la pianificazione e lo sviluppo futuro del settore del trasporto aereo.

Al fine di facilitare lo studio e l'analisi della suddetta rete, il presente lavoro di tesi ha come obiettivo la realizzazione di un'applicazione interattiva che permetta all'utente di effettuare diverse operazioni e diverse tipologie di analisi riguardo la rete aeroportuale mondiale.

Ai fini dell'analisi, la rete aeroportuale verrà vista sotto la veste di un grafo, la quale risulterà essere la rappresentazione più adatta a tale tipologia di dati.



**Figura 1.1:** Andamento del numero dei passeggeri di voli civili, periodo 1980-2020. La crescita è stata continua, fatta eccezione per particolari momenti storici, evidenziati in figura.

**Fonte:** [1]

# Capitolo 2

## Background

Questo capitolo offre una panoramica teorica sugli aspetti principali riguardanti la "Network Science", ovvero la scienza che si occupa dello studio e analisi delle reti. Si passa poi ad una spiegazione più dettagliata dei vari aspetti delle reti che sono alla base delle analisi eseguibili tramite il presente progetto.

### 2.1 Network Science

La "**Network Science**" è la scienza che studia e analizza le reti, al fine di comprendere le leggi fondamentali che regolano il funzionamento di sistemi complessi, i quali sono caratterizzati da un grandissimo numero di entità che interagiscono in maniera più o meno complessa tra loro.

Alcuni esempi di sistemi complessi sono:

- Internet, in cui gli host comunicano tra loro attraverso la rete;
- I Social Network, in cui moltissime persone interagiscono tra loro scambiando contenuti, like, e così via;
- Infrastrutture di trasporti

## 2.2 Reti o grafi

Le reti o grafi sono un modello matematico che permette di "rappresentare" le varie componenti di un sistema complesso e le rispettive interazioni.

Formalmente, un grafo è una coppia  $G=(V,E)$  tale che

- $V$  è l'insieme dei **vertici**;
- $E$  è l'insieme degli **archi**, ove un arco è definito come una coppia di vertici  $(u,v)$ .

Più nello specifico:

- i vertici rappresentano le componenti di un sistema complesso
- gli archi rappresentano le interazioni tra le diverse componenti

### 2.2.1 Tipi di grafi

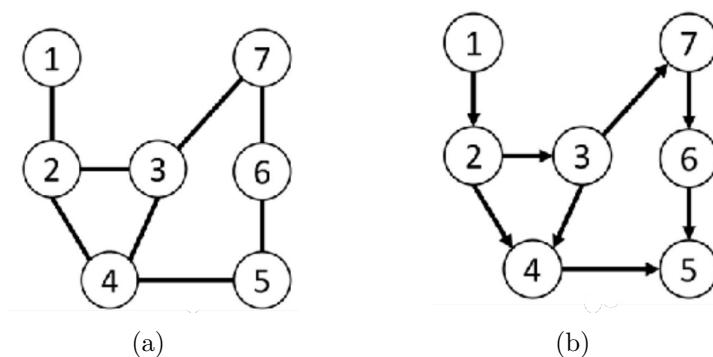
A seconda della loro struttura, è possibile fare una distinzione tra diverse tipologie di grafi.

#### 2.2.1.1 Grafi diretti e indiretti

Possiamo distinguere due principali tipologie di grafi: grafi **indiretti** e grafi **diretti**.

Un grafo è detto "indiretto" o "non orientato" se  $\forall(a,b) \in E$  anche  $(b,a) \in E$ ; altrimenti, il grafo è detto "diretto" o "orientato".

In altre parole, un grafo è indiretto se le interazioni tra due nodi sono in ogni caso reciproche, altrimenti il grafo è diretto.

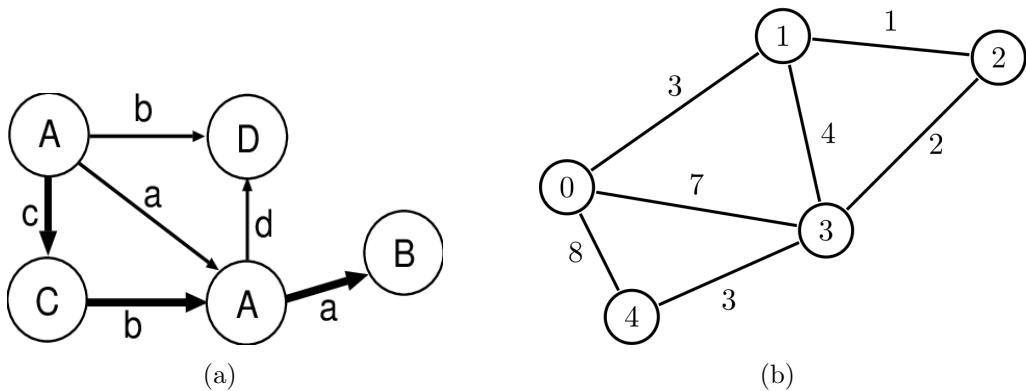


**Figura 2.1:** (a) Grafo indiretto (b) Grafo diretto

### 2.2.1.2 Grafi etichettati e pesati

È possibile associare diverse informazioni aggiuntive agli archi e nodi di una rete, le quali sono solitamente legate alla semantica della rete stessa. A seconda della tipologia di etichetta, possiamo fare una distinzione tra:

- grafi **etichettati**, in cui le informazioni aggiuntive sono etichette o stringhe.
- grafi **pesati**, in cui le informazioni aggiuntive sono valori numerici interi o reali.



**Figura 2.2:** (a) Grafo etichettato (b) Grafo pesato

### 2.2.1.3 Grafi bipartiti e multipartiti

Un **grafo bipartito** è un grafo in cui i nodi costituiscono due insiemi disgiunti  $U$  e  $V$  e ciascun arco del grafo collega un nodo di  $U$  ad un nodo di  $V$ . Inoltre, non esistono archi che collegano tra loro nodi dello stesso insieme.

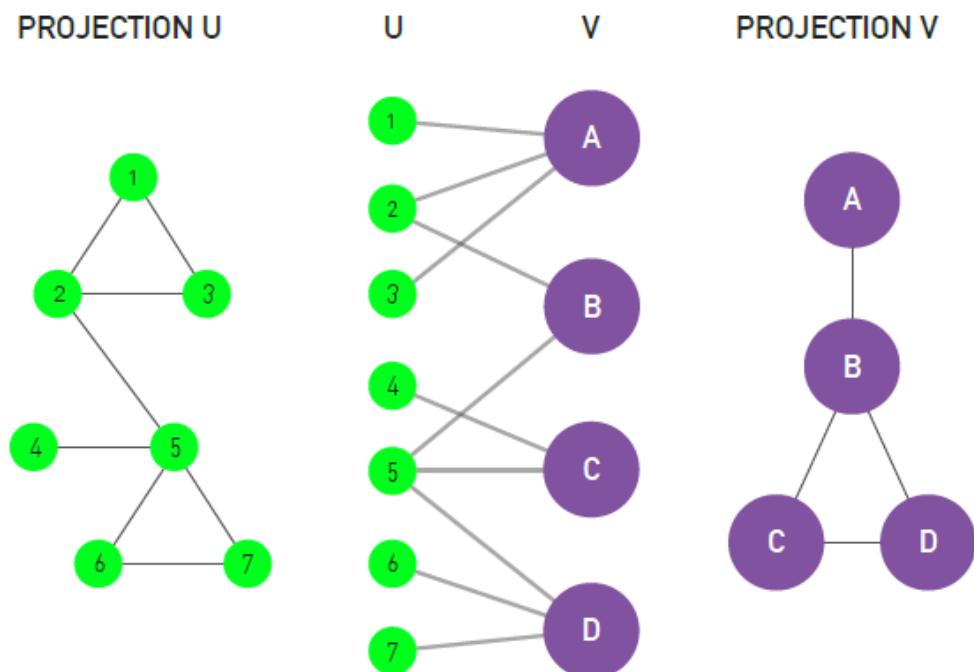
La definizione di grafo bipartito può poi essere estesa al caso di tre ("**grafi tripartiti**") o più ("**grafi multipartiti**") insiemi disgiunti di nodi.

### 2.2.1.4 Proiezioni di un grafo bipartito

A partire da un grafo bipartito è possibile creare due reti, chiamate "proiezioni": una rispetto all'insieme  $U$  e una rispetto all'insieme  $V$ .

In particolare, la **proiezione rispetto a  $U$**  è un grafo i cui nodi sono i nodi di  $U$  e un arco collega due nodi  $u, v \in U$  se e solo se  $u$  e  $v$  hanno almeno un adiacente in comune nel grafo bipartito.

Definizione analoga per la proiezione rispetto a  $V$ .

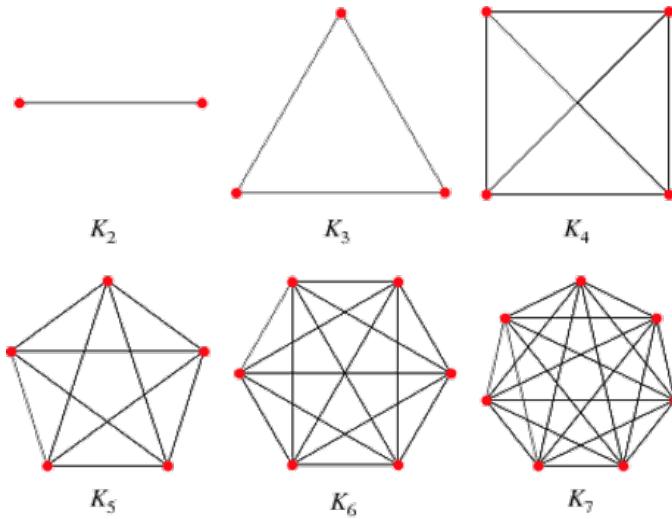


**Figura 2.3:** Esempio di grafo bipartito e sue proiezioni

### 2.2.1.5 Grafi completi

Un **grafo completo**, detto anche **"clique"**, è un grafo in cui *tutte le coppie di vertici distinte tra loro sono collegate da un arco*.

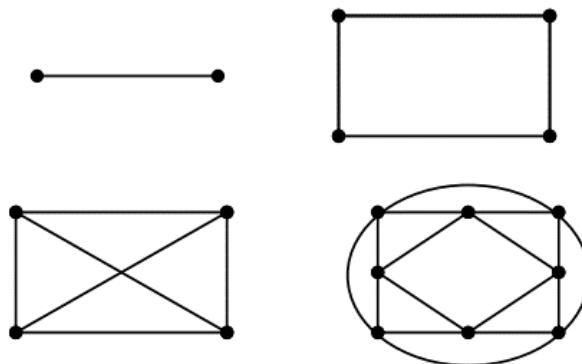
Formalmente, un grafo completo con  $n$  vertici è denotato dal simbolo  $K_n$ .



**Figura 2.4:** Esempi di grafi completi con un diverso numero di nodi

### 2.2.1.6 Grafi regolari

Un **grafo regolare** è un grafo in cui *tutti i nodi hanno lo stesso grado*.



**Figura 2.5:** Esempi di grafi regolari

### 2.2.2 Adiacenza tra nodi e grado di un nodo

Dato un grafo  $G = (V, E)$ , e considerati due generici nodi  $u, v \in V$ , diremo che "***v* è adiacente a *u***" se  $(u, v) \in E$ .

Considerando adesso un nodo  $u \in V$ , definiamo il **grado di *u*** come il **numero di nodi adiacenti a *u*** nel grafo.

Nel caso di un grafo diretto, è possibile fare un'ulteriore distinzione:

- **Grado uscente di *u*:** numero di nodi adiacenti ad *u*, o analogamente il numero di nodi  $x \in V$  tali che  $(u, x) \in E$
- **Grado entrante di *u*:** numero di nodi a cui *u* è adiacente, o analogamente il numero di nodi  $x \in V$  tali che  $(x, u) \in E$
- **Grado totale di *u*:** *grado entrante di *u* + grado uscente di *u**

Una volta definito il grado di un singolo nodo, definiamo il **grado medio della rete**  $\langle k \rangle$  come il rapporto tra il numero di archi e il numero di nodi di quest'ultima:

$$\langle k \rangle = \frac{|E|}{|V|}$$

Definiamo infine il concetto di **distribuzione dei gradi**: la distribuzione dei gradi  $P$  è una distribuzione di probabilità tale che  $P_k$  rappresenta la probabilità che un nodo random della rete abbia grado pari a  $k$ .

Data una rete reale, possiamo calcolare  $P_k$  come il rapporto tra il numero  $N_k$  di nodi di grado  $k$  e il numero di nodi  $N$  della rete:

$$P_k = \frac{N_k}{N} = \frac{N_k}{|V|}$$

### 2.2.3 Cammini tra due nodi e distanza

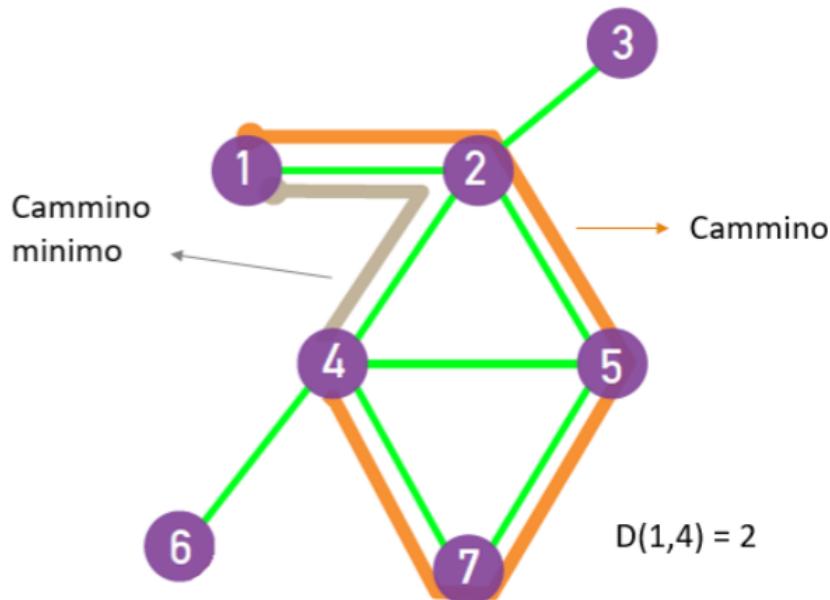
Un cammino tra due nodi  $u, v$  è definito come una **sequenza ordinata di  $n$  archi**  $(i_0, i_1), (i_1, i_2), \dots, (i_{n-1}, i_n)$ , dove  $i_0 = u$  e  $i_n = v$ .

La quantità  $n$  è detta "lunghezza del cammino".

Definiamo poi il **cammino minimo tra due nodi**  $u, v$  come il cammino di lunghezza minima.

In generale, tra due nodi  $u, v$  possono esistere **diversi cammini e diversi cammini minimi**.

Definiamo infine la **distanza tra due nodi**  $u, v$  come la lunghezza del cammino minimo tra  $u$  e  $v$ .



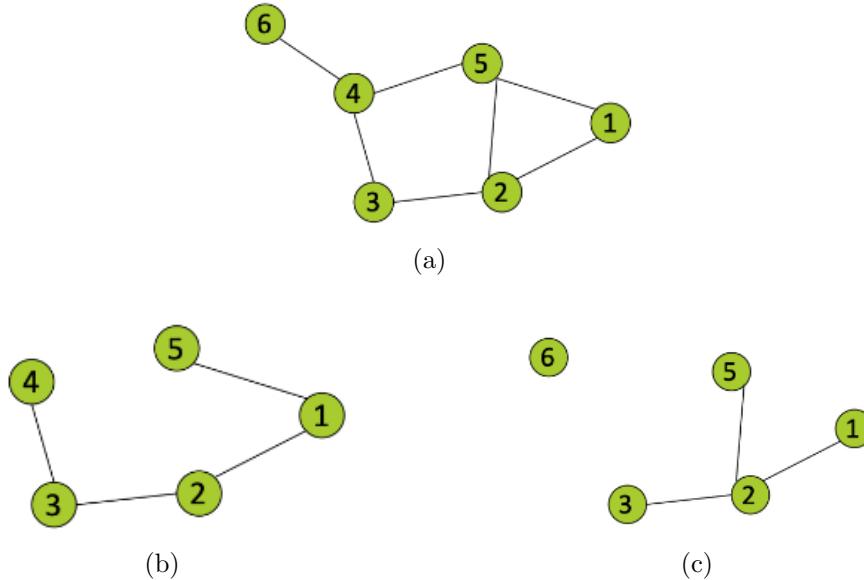
**Figura 2.6:** Cammini fra i nodi 1 e 4. La distanza in questo caso è pari a 2

### 2.2.4 Sottografi orientati e non orientati

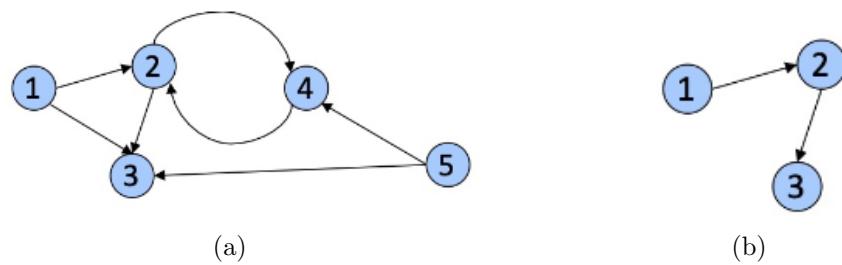
Sia  $G = (V, E)$  un grafo non orientato. Diciamo che un grafo  $G' = (V', E')$  è un **sottografo** di  $G$  se:

- $V' \subseteq V$
- $E' \subseteq E$  ed inoltre per ogni arco  $(u, v) \in E'$  i suoi estremi  $u, v$  appartengono entrambi a  $V'$

La definizione è analoga nel caso di grafi orientati.



**Figura 2.7:** (a) Grafo non orientato (b) Esempio 1 di sottografo (c) Esempio 2 di sottografo



**Figura 2.8:** (a) Grafo orientato (b) Esempio di sottografo

## 2.3 Graph mining

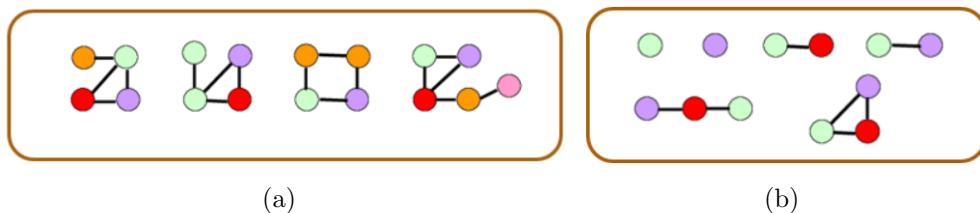
Il Graph Mining è una branca della Network Science che si occupa dell'estrazione di informazioni significative dai dati rappresentati sottoforma di grafi. Il Graph Mining prevede un'ampia varietà di possibili operazioni, ad esempio la scoperta di pattern ricorrenti, la ricerca di comunità, l'individuazione di nodi influenti, la classificazione dei nodi in base alle loro proprietà oppure la previsione di legami mancanti o futuri all'interno del grafo.

## 2.4 Frequent Subgraph Mining

Tra le operazioni tipiche del Graph Mining, una tra quelle di maggior interesse è il cosiddetto **"Frequent Subgraph Mining"** (**FSM**): esso consiste nella ricerca dei cosiddetti **"sottografi frequenti"**, ovvero di sottografi che compaiono frequentemente in un **"database di grafi"** definito in precedenza. Al fine di capire se un sottografo è frequente o meno, quello che si fa inizialmente è contare il numero di grafi del database al cui interno compare il sottografo in questione: tale quantità prende il nome di **"supporto"**. A questo punto, per stabilire se tale sottografo è frequente o meno, deve essere stabilita una **"soglia minima"**  $\sigma$ , chiamata **"soglia di supporto"**, e se il supporto del sottografo è maggiore o uguale di  $\sigma$ , allora il sottografo sarà considerato frequente.

Una soluzione alternativa prevede di fare riferimento al **"conteggio relativo"** delle occorrenze, piuttosto che al **"conteggio assoluto"**, calcolando la **frequenza del sottografo** come il rapporto tra il suo supporto e il numero di grafi del database. In questo caso la frequenza del sottografo e la soglia di supporto  $\sigma$  saranno espressi in termini di percentuale.

Diremo quindi che **un sottografo è frequente se esso è presente almeno nel  $\sigma\%$  dei grafi del database**.



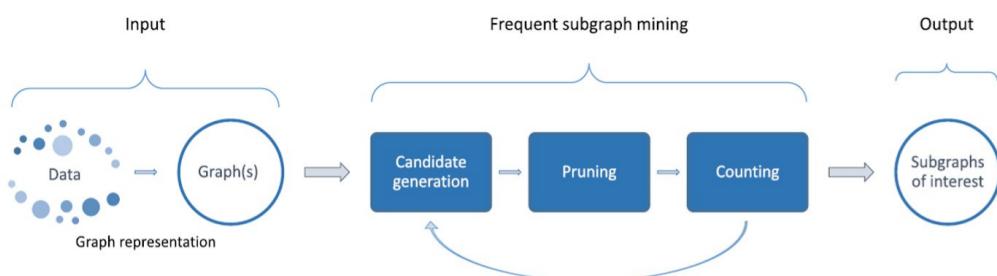
**Figura 2.9:** Esempio di applicazione di un algoritmo di FSM, con  $\sigma = 3$ .

(a) Database di partenza (b) Output dell'algoritmo: vengono restituiti tutti i sottografi che sono presenti in almeno  $\sigma$  grafi del database di partenza

### 2.4.1 Algoritmi di Frequent Subgraph Mining

Esistono numerosi algoritmi che permettono di effettuare la ricerca di sottografi frequenti.

In generale, tali algoritmi sono composti da tre passi fondamentali: **generazione dei candidati, pruning e conteggio**.



**Figura 2.10:** Passi generali di un algoritmo di Frequent Subgraph Mining

#### 2.4.1.1 Generazione dei candidati

Il primo passo di un algoritmo di Frequent Subgraph Mining consiste nella cosiddetta *generazione dei candidati*: essa avviene generando un insieme di "sottografi candidati" che potenzialmente potrebbero essere frequenti.

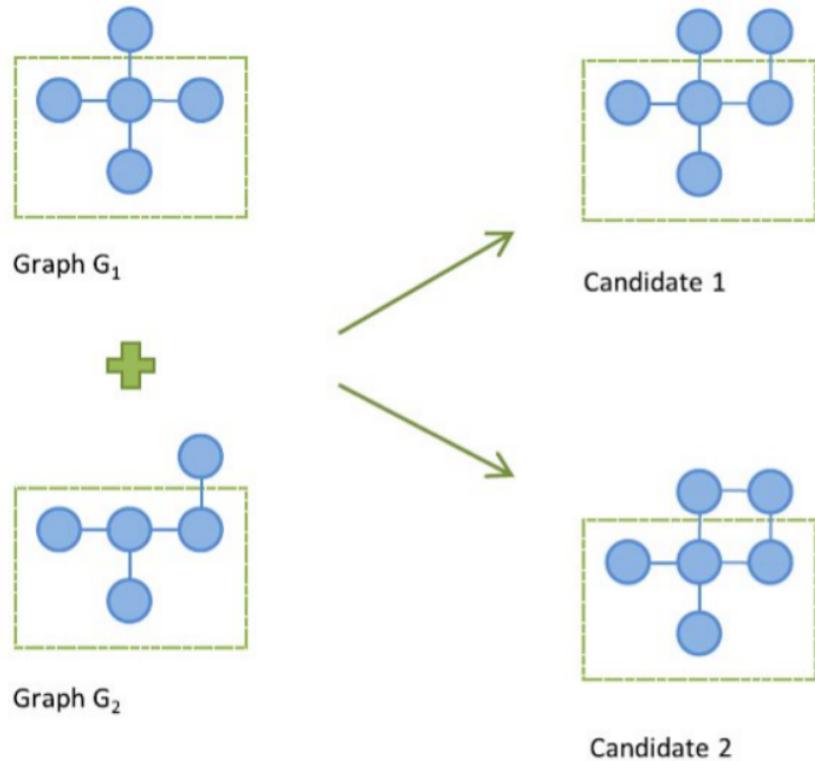
La generazione può avvenire usando due diverse tecniche:

- Metodo **"Join-based"**: un sottografo candidato con  $k + 1$  nodi è ottenuto dall'unione di due sottografi frequenti con  $k$  nodi. L'unione è possibile se e solo se i due sottografi hanno almeno un sottografo con  $k$  nodi in comune, chiamato **"core graph"**.

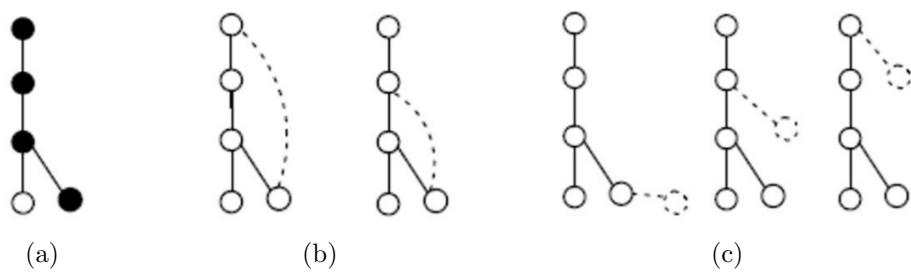
La generazione può anche essere fatta basandosi sugli archi. In tal caso, si aggiunge al grafo frequente un arco di un altro grafo frequente con lo stesso core.

- Metodo **"Extend-based"**: un sottografo candidato con  $k + 1$  nodi è generato estendendo un sottografo frequente con  $k$  nodi, aggiungendo un solo nodo. Una tecnica per evitare la generazione di sottografi candidati ridondanti consiste nell'effettuare l'estensione aggiungendo un nuovo nodo solo a partire da nodi del cosiddetto "cammino right-most", ovvero il cammino più a destra nell'albero di visita DFS del grafo da estendere.

Anche in questo caso, l'estensione può essere fatta aggiungendo un arco anziché un nodo.



**Figura 2.11:** Esempio di generazione di candidati con tecnica "Join-based". Abbiamo due possibili risultati: nel primo candidato viene inserito un arco tra i due nodi esterni al core, mentre nell'altro no



**Figura 2.12:** Esempio di generazione di candidati con tecnica "Extend-based".  
 (a) Albero prodotto da una visita DFS sul sottografo frequente (**nodi in nero: cammino right-most**)  
 (b) Due possibili estensioni tramite un arco  
 (c) Tre possibili estensioni tramite un nodo

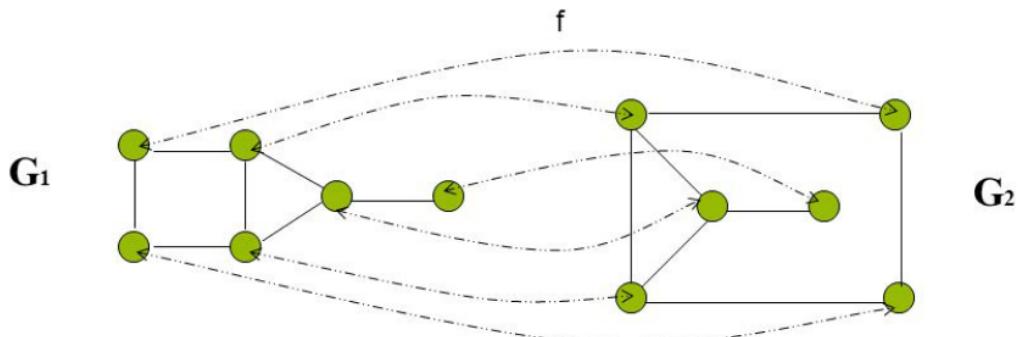
### 2.4.1.2 Pruning delle ridondanze

Il passo successivo è effettuare il cosiddetto **”pruning dei candidati”**: tale operazione consiste nel ridurre il numero di candidati da esaminare attraverso apposite euristiche. Ciò è chiaramente molto utile dal punto di vista dell’efficienza computazionale.

Un possibile criterio per effettuare il pruning è quello di eliminare candidati che sono ***isomorfi***, ovvero sono in qualche modo **uguali**.

Formalmente, due grafi  $G_1 = (E_1, V_1)$  e  $G_2 = (E_2, V_2)$  si dicono **”isomorfi”** se esiste una funzione biunivoca  $f$  dall’insieme dei vertici  $V_1$  nell’insieme dei vertici  $V_2$  tale che  $(u, v) \in E_1$  se e solo se  $(f(u), f(v)) \in E_2$ .

La funzione  $f$  è detta **”isomorfismo”** o **”mapping”**.



**Figura 2.13:** Esempio di isomorfismo tra grafi

### 2.4.1.3 Conteggio

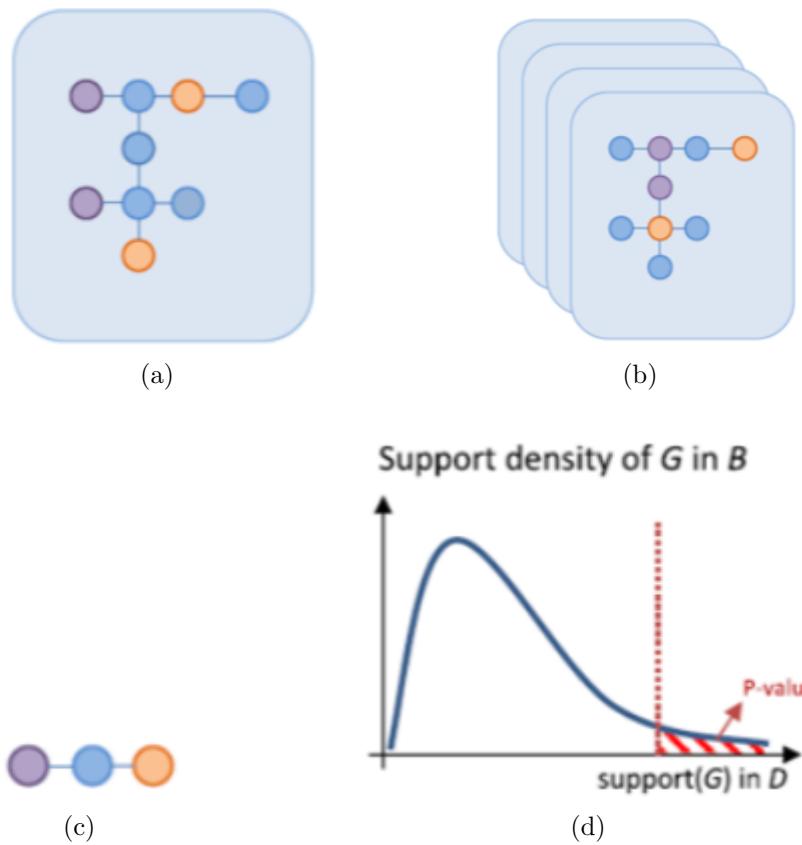
Una volta generati tutti i possibili candidati, si verifica se essi sono frequenti o meno, e in caso affermativo essi vengono aggiunti nell’insieme dei risultati.

Dopo aver verificato se i candidati sono frequenti o meno, il processo viene reiterato, a partire dalla generazione di nuovi candidati di dimensioni sempre maggiori, per poi continuare a identificare sottografi frequenti sempre più grandi e complessi.

### 2.4.2 Significatività statistica

Nel momento in cui si applica un qualunque algoritmo di FSM ad un database di grafi, con molta probabilità l'algoritmo restituirà molti sottografi frequenti. Tuttavia, non è detto che i sottografi che risultano essere frequenti siano "rilevanti" per il problema che si sta affrontando, in quanto potrebbero, per esempio, rappresentare dei pattern banali: un esempio di pattern banale sono i sottografi formati da un unico nodo. È necessario quindi un modo per capire se un sottografo frequente è in qualche modo rilevante.

A tal fine è necessario calcolare la cosiddetta "**significatività statistica**", detta anche "**p-value**", rispetto ad un database di background costruito opportunamente. Una volta calcolato tale valore con appositi metodi, se esso è al di sotto di una certa soglia  $\alpha$ , allora **il sottografo è "statisticamente significativo"**.



**Figura 2.14:** Esempio di calcolo significatività statistica.

- (a) Database di grafi  $D$
- (b) Database di Background  $B$
- (c) Sottografo candidato  $G$
- (d) P-value

## 2.5 Caratteristiche delle reti reali

Uno degli obiettivi della Network Science è quello di costruire modelli in grado di riprodurre proprietà specifiche della rete reale, legate per esempio alla distribuzione dei gradi dei nodi, al modo in cui i nodi tendono a legarsi fra loro, e molto altro.

In generale, le reti reali **non hanno una struttura regolare**, bensì presentano, almeno in apparenza, una certa *randomicità*.

Più nello specifico, una rete reale, nella maggior parte dei casi, presenta le seguenti caratteristiche:

- Elevato numero di nodi;
- Proprietà **”Small World”**: le distanze medie tra i nodi sono piccole.
- Alto coefficiente di clustering;
- Distribuzione dei gradi non omogenea: si hanno molti nodi con grado basso e pochi nodi con grado elevato. Nello specifico, si parla di **”rete scale-free”**;
- Proprietà di **”crescita della rete”**: nelle reti reali il numero di nodi cresce continuamente grazie all’aggiunta di nuovi nodi;
- Proprietà di **”Preferential attachment”**: i nuovi nodi che entrano nella rete tendono a legarsi ai nodi più connessi, cioè a quelli più importanti.

### 2.5.1 Proprietà ”Small World”

La proprietà ”small world” afferma che **la distanza tra due nodi qualsiasi è relativamente piccola rispetto alle dimensioni della rete**.

Questo concetto è legato alla cosiddetta **”teoria dei sei gradi di separazione”**. In tale esperimento, condotto nel 1967 dallo psicologo americano Stanley Milgram, vennero selezionati in maniera casuale un gruppo di studenti americani, ai quali venne chiesto di spedire un pacchetto ad un estraneo del Massachusetts, a migliaia di chilometri di distanza. Insieme al pacchetto, gli studenti ricevettero indicazioni riguardo il nome del destinatario, il suo impiego e la zona di residenza, senza però specificare l’indirizzo esatto. Ad ogni partecipante all’esperimento fu quindi chiesto di spedire il proprio pacchetto ad una persona da egli conosciuta che, a loro giudizio, potesse avere maggiori probabilità di conoscere il destinatario finale. Quella persona avrebbe fatto a sua volta lo stesso, fino a quando il pacchetto non fosse arrivato a

destinazione.

Al termine dell'esperimento, Milgram scoprì che il numero medio di intermediari era 5.2, molto vicino a 6.

Formalmente, una rete soddisfa la proprietà small-world se la distanza media  $\langle d \rangle$  tra i nodi è:

$$\langle d \rangle \approx \frac{\ln N}{\ln \langle K \rangle}$$

NETWORK	$N$	$L$	$\langle k \rangle$	$\langle d \rangle$	$d_{max}$	$\frac{\ln N}{\ln \langle k \rangle}$
Internet	192,244	609,066	6.34	6.98	26	6.58
WWW	325,729	1,497,134	4.60	11.27	93	8.31
Power Grid	4,941	6,594	2.67	18.99	46	8.66
Mobile Phone Calls	36,595	91,826	2.51	11.72	39	11.42
Email	57,194	103,731	1.81	5.88	18	18.4
Science Collaboration	23,133	93,439	8.08	5.35	15	4.81
Actor Network	702,388	29,397,908	83.71	3.91	14	3.04
Citation Network	449,673	4,707,958	10.43	11.21	42	5.55
E. Coli Metabolism	1,039	5,802	5.58	2.98	8	4.04
Protein Interactions	2,018	2,930	2.90	5.61	14	7.14

**Figura 2.15:** Distanze medie in alcune reti reali. I valori si avvicinano approssimativamente proprio al valore 6

### 2.5.2 Reti scale-free e legge "power-law"

Nelle reti reali, la distribuzione dei gradi è ben approssimata da una specifica distribuzione, che prende il nome di **power-law**.

Formalmente, una distribuzione  $p_k$  segue una legge power-law se:

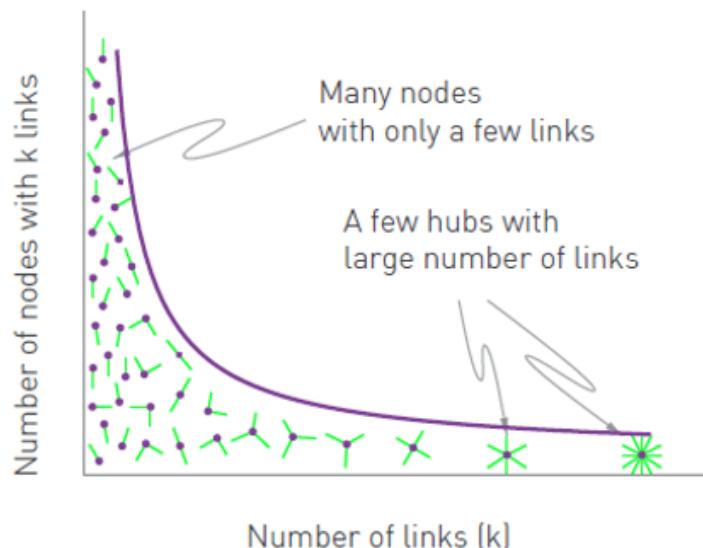
$$p_k \sim k^{-\gamma}$$

dove  $\gamma$  è detto **esponente di grado**.

Nella maggior parte delle reti reali si ha  $2 < \gamma < 3$ .

Le reti la cui distribuzione dei gradi segue una power-law sono dette **reti scale-free**.

Come si può notare in figura 2.16, nelle reti scale-free vi è un basso numero di nodi che hanno un grado elevato (chiamati **hub**), mentre si ha un elevato numero di nodi con basso grado.



**Figura 2.16:** Distribuzione power-law

### 2.5.2.1 Robustezza di una rete scale-free

Grazie alla presenza di un numero ristretto di nodi importanti, cioè gli hub, una rete scale-free risulta essere **più robusta ad attacchi o situazioni in cui uno o più collegamenti nella rete vengono meno**.

Ciò è dovuto proprio al fatto che il problema è grave solo se il "malfunzionamento" coinvolge un hub. Se invece esso coinvolge un nodo periferico, il funzionamento della rete nella maggior parte dei casi non viene compromesso.

Ciò è cruciale dal punto di vista della sicurezza, ed è uno degli aspetti di cui tenere maggiormente conto nella progettazione di reti infrastrutturali, tra cui, ad esempio, proprio la rete aeroportuale.

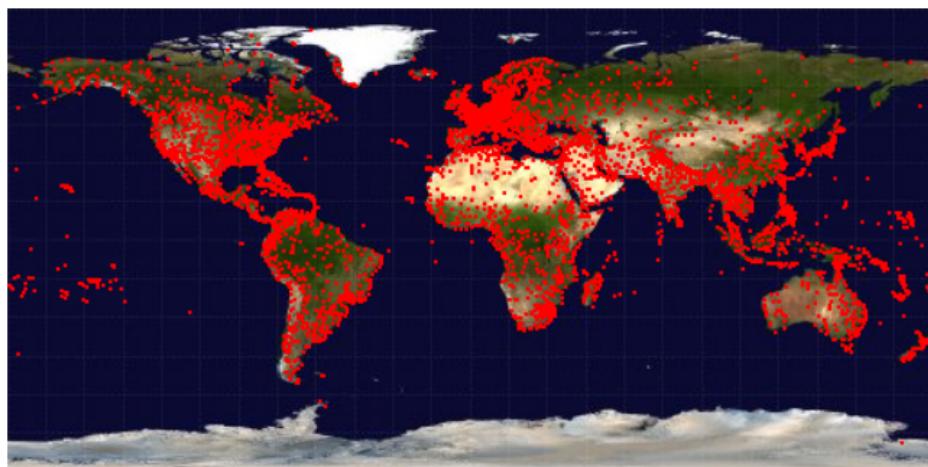
# Capitolo 3

## Dataset ”OpenFlights”

In questo capitolo vengono presentate le caratteristiche del dataset utilizzato per la realizzazione del presente lavoro di tesi, il cui nome è *OpenFlights*. [2]

### 3.1 Struttura del dataset

Il dataset OpenFlights contiene informazioni relative a **oltre 10.000 aeroporti, stazioni ferroviarie e porti** sparsi in tutto il mondo e alle **rotte esistenti tra loro**. A causa delle sue dimensioni, esso è suddiviso in diversi ”sotto-dataset”, ognuno dei quali contiene dati relativi ad uno specifico aspetto delle infrastrutture prese in esame.



**Figura 3.1:** Aeroporti, stazioni ferroviarie e porti le cui informazioni sono presenti nel dataset ”OpenFlights”

### 3.1.1 Airport Database

L'OpenFlight **Airport Database** contiene informazioni riguardanti i singoli aeroporti presenti nel dataset, aggiornate fino al Gennaio 2017. Per ognuno di essi sono riportate le seguenti informazioni:

- **Airport ID:** identificativo univoco per lo specifico aeroporto, assegnato da OpenFlights.
- **Name:** Nome dell'aeroporto.
- **City:** Città in cui si trova l'aeroporto.
- **Country:** nazione, contea, territorio, ... in cui si trova l'aeroporto. In formato ISO 3166-1 (vedere la sez. 3.1.3 per i relativi codici ISO 3166-1).
- **IATA:** codice IATA di 3 lettere. "NULL" se il codice non è assegnato o è sconosciuto.
- **ICAO:** codice ICAO di 4 lettere. "NULL" se il codice non è assegnato o è sconosciuto.
- **Latitude:** Gradi decimali, con precisione alla sesta cifra significativa. Numero negativo coincide con il Sud, numero positivo con il Nord.
- **Longitude:** Gradi decimali, con precisione alla sesta cifra significativa. Numero negativo coincide con l'Ovest, numero positivo con l'Est.
- **Altitude:** altitudine in *ft* ("piedi").
- **Timezone:** Offset rispetto a UTC. Ore "frazionarie" sono espresse sotto forma di numero decimale, es. la timezone dell'India è 5.5.
- **DST:** "Daylight savings time". Un valore tra E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) o U (Unknown).
- **Tz database time:** Timezone del campo "tz" in formato "Olson", es. "America/Los\_Angeles".
- **Type:** in questo caso è sempre "airport".
- **Source:** Fonte dei dati.

### 3.1.2 Airline Database

L'OpenFlight **Airlines Database** contiene informazioni riguardanti le diverse compagnie aeree, aggiornate fino al Gennaio 2012. As of January 2012, the OpenFlights Airlines Database contains 5888 airlines. Per ognuna di esse sono riportate le seguenti informazioni:

- **Airline ID:** identificativo univoco per la specifica compagnia, assegnato da OpenFlights.
- **Name:** nome della compagnia.
- **Alias:** alias della compagnia, es. per la compagnia "All Nippon Airways" è "ANA".
- **IATA:** codice IATA di 2 lettere, se disponibile.
- **ICAO:** codice ICAO di 3 lettere, se disponibile.
- **Callsign:** nominativo della compagnia.
- **Country:** nazione di origine della compagnia. In formato ISO 3166-1 (vedere la sez. 3.1.3 per i relativi codici ISO 3166-1).
- **Active:** "Y" se la compagnia è attiva, "N" se non lo è più.

### 3.1.3 Countries Database

L'OpenFlight **Country Database** contiene informazioni relative ai codici ISO 3166-1 delle nazioni, utilizzabili per capire a quale nazione si riferiscono i codici ISO 3166-1 presenti nei dataset relativi agli aeroporti e compagnie aeree. In particolare sono riportate le seguenti informazioni:

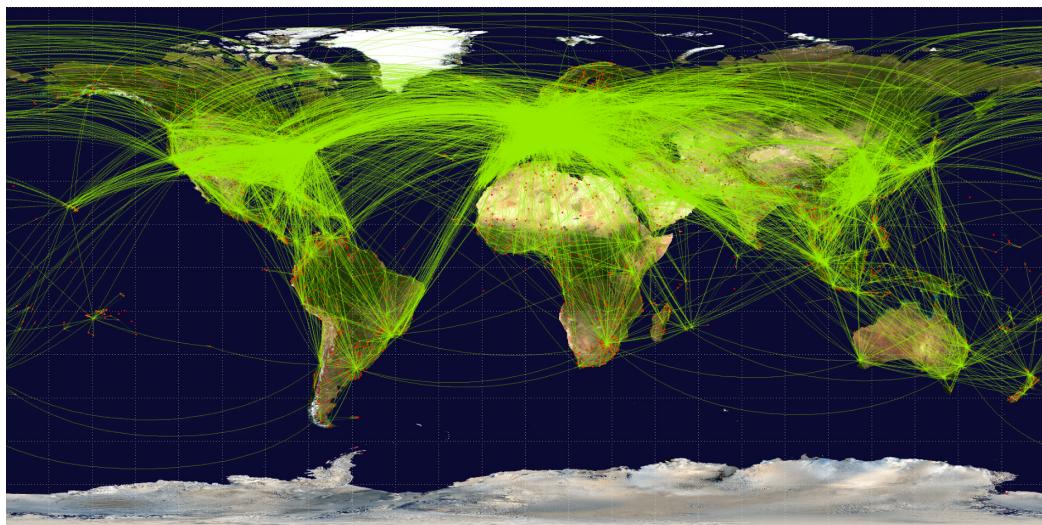
- **Name:** nome completo della nazione, territorio, contea, ...
- **iso\_code:** codice ISO 3166-1 per la nazione, territorio, contea, ... in questione. Il codice è formto da 2 lettere ed è unico.
- **dafif\_code:** codice FIPS della nazione usato dall'organizzazione DAFIF. Sono dei codici ormai obsoleti.

### 3.1.4 Routes Database

L'**OpenFlight Routes Database** contiene informazioni relative a più di 67.000 rotte aeree in tutto il mondo, come si può notare in figura Figura 3.2. Per ognuna di esse sono riportate le seguenti informazioni:

- **Airline**: codice IATA di 2 lettere o codice ICAO di 3 lettere che identifica la compagnia che effettua tale rotta.
- **Airline ID**: identificativo univoco per la specifica compagnia, assegnato da OpenFlights.
- **Source airport**: codice IATA di 3 lettere o codice ICAO di 4 lettere che identifica l'aeroporto di partenza.
- **Source airport ID**: identificativo univoco per l'aeroporto di partenza, assegnato da OpenFlights.
- **Destination airport**: codice IATA di 3 lettere o codice ICAO di 4 lettere che identifica l'aeroporto di arrivo.
- **Destination airport ID**: identificativo univoco per l'aeroporto di arrivo, assegnato da OpenFlights.
- **Codeshare**: "Y" se il volo è un "codeshare", cioè è operato da una compagnia diversa da quella riportata nel campo "*Airline*", altrimenti il campo è vuoto.
- **Stops**: numero di scali ("0" per voli diretti)
- **Equipment**: codice di 3 lettere che identifica il tipo di aereo usato di solito per tale volo.

**N.B.:** le rotte sono direzionali: se una compagnia aerea opera delle rotte da *A* ad un aeroporto *B* e anche da *B* ad *A*, tali rotte sono salvate separatamente



**Figura 3.2:** Rotte aeree presenti nel dataset

### 3.1.5 Planes Database

L'OpenFlight Planes Database contiene informazioni relative a più di 150 aerei per voli civili, i quali sono usati per la maggior parte dei voli operati oggigiorno. Per ognuno di essi sono riportate le seguenti informazioni:

- **Name:** nome completo del velivolo.
- **IATA code:** codice IATA di 3 lettere per il velivolo.
- **ICAO code:** codice ICAO di 4 lettere per il velivolo

# Capitolo 4

## Metodologie di lavoro

### 4.1 Modellazione dei dati di OpenFlights come un multigrafo

La natura interconnessa delle rotte aeree suggerisce l'idea di modellare i dati presenti nel dataset OpenFlights come una struttura a grafo, in cui:

- I nodi corrispondono ai singoli aeroporti
- Gli archi corrispondono alle rotte aeree. In particolare, dati due aeroporti A e B, avremo un arco orientato  $(A, B)$  per ognuna delle rotte aeree che hanno come aeroporto di partenza l'aeroporto A e come aeroporto di destinazione l'aeroporto B.

Sulla base di tali specifiche, il grafo che rappresenterà la rete aeroportuale mondiale sarà un **multigrafo diretto ed etichettato**.

Formalmente, definiamo un **multigrafo etichettato** come una sestupla

$$G = (V, E, \Sigma_V, \Sigma_E, l_V, l_E)$$

in cui:

- $V$  è l'insieme dei nodi
- $E \subseteq (V \times V)$  è l'insieme degli archi
- $\Sigma_V$  è l'insieme delle etichette dei nodi
- $\Sigma_E$  è l'insieme delle etichette degli archi
- $l_V : V \rightarrow \Sigma_V^+$  è una funzione che assegna una o più etichette ad ogni nodo

- $l_E : E \rightarrow \Sigma_E^+$  è una funzione che assegna una o più etichette ad ogni arco

In pratica, un multigrafo è un grafo in cui coppie di nodi possono essere collegate da più di un arco.

Valgono poi tutte le definizioni date nel capitolo introduttivo, riguardanti l'orientamento degli archi, l'adiacenza tra nodi, il grado di un nodo, e così via.

## 4.2 Linguaggio R e librerie utilizzate

Il linguaggio scelto per la realizzazione di questo progetto è il linguaggio R [3].

R è un linguaggio di programmazione open-source ampiamente utilizzato nell'ambito dell'analisi dei dati e della statistica, apprezzato soprattutto per la sua versatilità e la vasta gamma di pacchetti e librerie disponibili. R fornisce un'ampia varietà di strumenti utili ad esempio per metodi statistici avanzati, modelli di machine learning, visualizzazione dei dati, realizzazione di web app, e molto altro.

Alcune delle librerie principali utilizzate e le relative funzionalità sono le seguenti:

- **shiny**: creazione di applicazioni R con interfacce grafiche interattive basate sul web [4]
- **igraph**: creazione e analisi di grafi [5]
- **ggplot2**: creazione di grafici e visualizzazioni interattive di dati [6]
- **leaflet**: creazione di mappe geografiche interattive [7]
- **visNetwork**: creazione e visualizzazione interattiva di grafi [8]

### 4.3 Formato dei dati

OpenFlights mette a disposizione i propri dataset in formato **CSV** ("Comma-Separated Values"). In questa tipologia di file, ogni riga corrisponde a una tupla del dataset, e i valori di ogni riga, separati da virgole, corrispondono al valore di uno specifico campo della tupla in questione.

### 4.4 Caricamento del dataset in R

La prima operazione da effettuare è importare i dati presenti nel dataset OpenFlight all'interno dell'applicazione R. Una volta scaricati i diversi file CSV, i dati vengono importati in degli appositi "*dataframe*", ovvero una struttura dati che organizza i dati in tabelle a due dimensioni.

Questa operazione viene effettuata per ognuno dei cinque dataset scaricati.

**Codice 4.1:** Caricamento del dataset

```

1 # CARICAMENTO DATI RELATIVI AGLI AEROPORTI
2
3 airportsData <- read.csv("dataset\\airports.txt", header =
4   FALSE, sep = ",", quote = "\"")
5 colnames(airportsData) <- c("openflightCode", "name", "city",
6   "country", "IATA", "ICAO", "latitude", "longitude", "altitude",
7   "timezone", "DST", "tzdb", "type", "source")
8
9 # CARICAMENTO DATI RELATIVI ALLE ROTTE AEREE
10 routeData <- read.csv("dataset\\routes.txt", header = FALSE,
11   sep = ",", quote = "\"")
12 colnames(routeData) <- c("airline", "airlineID", "sourceAirport",
13   "sourceAirportID", "destinationAirport", "destinationAirportID",
14   "codeshare", "stops", "equipment")
15
16 # CARICAMENTO DATI RELATIVI ALLE COMPAGNIE AEREE
17 airlinesData <- read.csv("dataset\\airlines.txt", header =
18   FALSE, sep = ",", quote = "\"")
19 colnames(airlinesData) <- c("openflightCode", "name", "alias",
20   "IATA", "ICAO", "callsign", "country", "active")
21
22 # CARICAMENTO DEI DATI RELATIVI ALLE NAZIONI
23 countriesData <- read.csv("dataset\\countries.txt", header =
24   FALSE, sep = ",", quote = "\"")
25 colnames(countriesData) <- c("name", "isoCode", "dafifCode")
26
27 # CARICAMENTO DEI DATI RELATIVI AI VELIVOLI
28 planesData <- read.csv("dataset\\planes.txt", header = FALSE,
29   sep = ",", quote = "\"")
30 colnames(planesData) <- c("name", "IATA", "ICAO")

```

## 4.5 Fase di Pre-processing

Una volta importati i dati provenienti da OpenFlights, è necessario effettuare una fase preliminare di pre-processing, al fine di adattare tali dati alle esigenze dell'applicazione.

Osservando attentamente i dati importati, si notano dei problemi riguardanti alcune tuple, che saranno risolti rimuovendo queste ultime.

All'interno del dataframe riguardante gli aeroporti, notiamo che alcune tuple non riportano il codice IATA dei relativi aeroporti, il quale risulterà fondamentale nelle analisi successive per identificare i singoli aeroporti.

Per questo motivo, rimuoviamo dal dataset tutte le tuple riguardanti aeroporti per cui non si conosce il relativo codice IATA.

Successivamente, si nota che nel dataframe riguardante le rotte aeree, sono presenti delle tuple in cui, nel campo relativo al codice dell'aeroporto di partenza o destinazione, sono presenti dei codici IATA sconosciuti, oppure l'aeroporto di partenza coincide con l'aeroporto di arrivo: anche queste tuple devono essere rimosse.

Infine, per semplificare il processo di analisi eseguito dall'applicazione, si è scelto di considerare soltanto voli diretti, ovvero quelli per cui il campo "stops" ha valore pari a 0.

### Codice 4.2: Rimozione di tuple errate

```

1 # CONSIDERIAMO SOLO VOLI DIRETTI
2 routeData <- subset(routeData , stops==0)
3
4 # RIMOZIONE DEGLI AEROPORTI DI CUI NON SI CONOSCE IL CODICE
   IATA
5 airportsData <- airportsData[airportsData$IATA != "\\\n", ]
6
7 # RIMOZIONE DEGLI AEROPORTI IN CUI IL CODICE IATA E'
   SCONOSCIUTO
8 routeData <- routeData[routeData$sourceAirport %in%
   airportsData$IATA & routeData$destinationAirport %in%
   airportsData$IATA, ]
9
10 #RIMOZIONE DELLE ROTTE AEREI IN CUI L'AEROPORTO SORGENTE E'
    UGUALE ALL'AEROPORTO DI DESTINAZIONE
11 routeData <- routeData[routeData$sourceAirport != routeData$destinationAirport, ]
```

## 4.6 Creazione del grafo

Terminata la fase di pre-processing, è possibile usare i dataframe precedentemente definiti per creare il grafo che rappresenterà la rete aeroportuale, così come definito nella sezione 4.1.

La libreria scelta per la creazione del grafo è la libreria ***"igraph"***, la quale permette la creazione di grafi attraverso l'apposita funzione *graph\_from\_data\_frame*.

Prima di chiamare tale funzione, è necessario modificare la struttura dei dataframe relativi agli aeroporti e alle rotte aeree, in quanto la funzione richiede che questi ultimi abbiano una struttura ben precisa:

- la prima colonna del dataframe che sarà usato per definire i nodi del grafo dovrà contenere gli identificativi di questi ultimi. In questo caso, il valore scelto come identificativo per i singoli aeroporti sarà il relativo codice IATA. Tutte le successive colonne saranno interpretate come etichette/attributi dei nodi.
- le prime due colonne del dataframe che sarà usato per definire gli archi del grafo dovranno contenere gli identificativi del nodo sorgente e del nodo destinazione. In questo caso, la prima colonna dovrà contenere i codici IATA di tutti gli aeroporti di partenza, mentre la seconda colonna dovrà contenere i codici IATA di tutti gli aeroporti di arrivo. Tutte le successive colonne saranno interpretate come etichette/attributi dei nodi.

È necessario quindi riordinare le colonne dei due dataframe *airportsData* e *routeData* in maniera tale da soddisfare questi requisiti.

Una volta fatto ciò, basterà eseguire la funzione apposita di *igraph*, *graph\_from\_data\_frame*, per creare il grafo.

**Codice 4.3:** Creazione del grafo che rappresenta la rete aeroportuale

```

1 # cambio dell'ordine delle colonne del dataframe
2 airportsData <- airportsData[, c(5,1,2,3,4,6:13)]
3 routeData <- routeData[, c(3,5,1,2,4,6:9)]
4
5 # CREAZIONE DEL GRAFO, A PARTIRE DAI DUE DATAFRAME
6 graph <- graph_from_data_frame(d=routeData , vertices=
    airportsData , directed=T)

```

## 4.7 Assegnamento dei pesi agli archi

Una volta creato il grafo, sarà necessario assegnare ad ognuno degli archi un nuovo attributo *"weight"* che indica il peso di tale arco.

In questo caso, il peso corrisponderà alla ***distanza geografica*** tra gli aeroporti che danno vita a tale arco.

Il calcolo della distanza viene effettuato attraverso una libreria di R, chiamata ***"geosphere"***, la quale offre un'apposita funzione *"distm"*, che permette di calcolare la distanza geografica tra due punti, a partire dalle rispettive coordinate, con uno degli algoritmi messi a disposizione dalla libreria stessa.

**Codice 4.4:** Assegnazione dei pesi agli archi

```

1 for(i in 1:length(E(graph))) {
2   fromAirport = get.edgelist(graph)[i,][1] #id del nodo dell'
      aeroporto "sorgente"
3   toAirport = get.edgelist(graph)[i,][2] #id del nodo dell'
      aeroporto "destinazione"
4
5   # ottenimento coordinate aeroporto sorgente
6
7   xFrom <- vertex_attr(graph, "longitude", index = fromAirport
      )
8   yFrom <- vertex_attr(graph, "latitude", index = fromAirport)
9
10  # ottenimento coordinate aeroporto destinazione
11
12  xTo <- vertex_attr(graph, "longitude", index = toAirport)
13  yTo <- vertex_attr(graph, "latitude", index = toAirport)
14
15  dist <- distm (c(xFrom, yFrom), c(xTo, yTo), fun =
      distHaversine) #distanza in metri
16
17  E(graph)[i]$weight = dist/1000 #imposto il peso dell'arco
      uguale alla distanza (in Km) tra i due aeroporti
18 }
```

## 4.8 Scrittura del grafo su file

Una volta ultimata la creazione del grafo che rappresenta la rete e aver assegnato i relativi pesi ad ognuno degli archi, risulta molto utile memorizzare il grafo su un file con un apposito formato.

In questo caso, il formato scelto è ”**GraphML**”, il quale permette di memorizzare un grafo con la struttura di un file XML (esempio: codice 4.6).

Tale salvataggio può essere effettuato con l'apposita funzione *write\_graph* messa a disposizione da *igraph*.

Nel momento in cui si dovrà aprire il file, dovrà essere invece utilizzata la funzione *read\_graph*.

**Codice 4.5:** Salvataggio del grafo in un file GraphML

```
1 # SCRITTURA DEL GRAFO IN UN FILE IN FORMATO "GRAPHML"
2
3 write_graph(graph, file = "airportsGraph.graphml", format = "
4   graphml")
5
6 # CARICAMENTO DEL GRAFO
7 graph <- read_graph(file="airportsGraph.graphml",format = "
8   graphml")
```

**Codice 4.6:** Frammento del file GraphML della rete aeroportuale

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <graphml xmlns="http://graphml.graphdrawing.org/xmlns"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://graphml.graphdrawing.org/
5                               xmlns
6                               http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd
7           " >
8   <!-- Created by igraph -->
9   <key id="v_name" for="node" attr.name="name" attr.type="string"/>
10  <key id="v_openflightCode" for="node" attr.name="openflightCode" attr.type="double"/>
11  ...
12  <key id="e_sourceAirportID" for="edge" attr.name="sourceAirportID" attr.type="string"/>
13  <key id="e_destinationAirportID" for="edge" attr.name="destinationAirportID" attr.type="string"/>
14  ...
15  ...
16
17  <graph id="G" edgedefault="directed">
18    <node id="n0">
19      <data key="v_name">Goroka Airport</data>
20      <data key="v_openflightCode">1</data>
21      ...
22      <data key="v_type">airport</data>
23      <data key="v_id">n0</data>
24    </node>
25
26  ...
27  ...
28
29  <edge source="n2262" target="n2284">
30    <data key="e_airline">2B</data>
31    <data key="e_airlineID">410</data>
32    ...
33    <data key="e_equipment">CR2</data>
34    <data key="e_weight">1508.51359917546</data>
35  </edge>
```

# Capitolo 5

## Struttura dell'applicazione

In questo capitolo viene presentata la struttura dell'applicazione che permetterà all'utente di effettuare un'ampia varietà di analisi riguardanti la rete aeroportuale mondiale.

L'applicazione sarà creata grazie a **Shiny**.

Shiny è una libreria di R che permette la creazione di web application in maniera molto semplice e efficace.

Il vantaggio principale è che non è necessario avere alcuna conoscenza dei classici linguaggi di programmazione, scripting e mark-up tipici del web, come HTML, CSS o JavaScript.

Un'altra importante caratteristica di Shiny è quella di permettere la creazione di app completamente customizzabili, dando la possibilità di inserire un'ampia varietà di componenti all'interno della User Interface e di creare il backend dell'applicazione da zero, il tutto utilizzando solamente il linguaggio R.

## 5.1 Struttura di un'applicazione Shiny

Ogni applicazione creata con Shiny assume una struttura ben precisa, costituita da due parti fondamentali:

- La parte di **User Interface**
- La parte **Server**

### 5.1.1 User Interface

La parte di User Interface (UI) di una web app Shiny è responsabile dell'aspetto visivo e dell'interfaccia utente dell'applicazione.

In essa vengono definiti i componenti che l'utente vedrà all'interno della pagina, ad esempio pulsanti, caselle di testo, grafici e tabelle. Gli elementi della UI potranno poi essere personalizzati esteticamente utilizzando fogli di stile CSS.

Un'altro aspetto importante della UI è la definizione delle interazioni tra i diversi componenti. Ad esempio, è possibile impostare un'azione da eseguire quando un pulsante viene premuto o quando un'opzione viene selezionata da un menu a tendina.

Nella UI è inoltre possibile definire degli output dinamici, che vengono aggiornati automaticamente quando un input cambia. Questo consente agli utenti di interagire con i dati e visualizzare i risultati in tempo reale: questo particolare paradigma prende il nome di "**reactive programming**".

Quando viene definita inizialmente la user interface, sarà inoltre necessario specificare la tipologia di **layout** della pagina.

Esistono essenzialmente due tipologie di layout:

- Layout "**fluidPage**": le dimensioni della pagina è fissa. Potrebbero quindi verificarsi problemi di visualizzazione su dispositivi con diverse dimensioni e aspect-ratio;
- Layout "**fixedPage**": le dimensioni della pagina si adattano al dispositivo e al contenuto da visualizzare.

Esistono poi altre tipologie di layout custom che possono essere importate all'interno dell'applicazione: uno di questi è il layout "**dashboardPage**" il quale, come suggerisce il nome stesso, permette di creare web app con lo stile di una dashboard, cioè una pagina adatta alla visualizzazione di tabelle, grafici, mappe e molto altro.

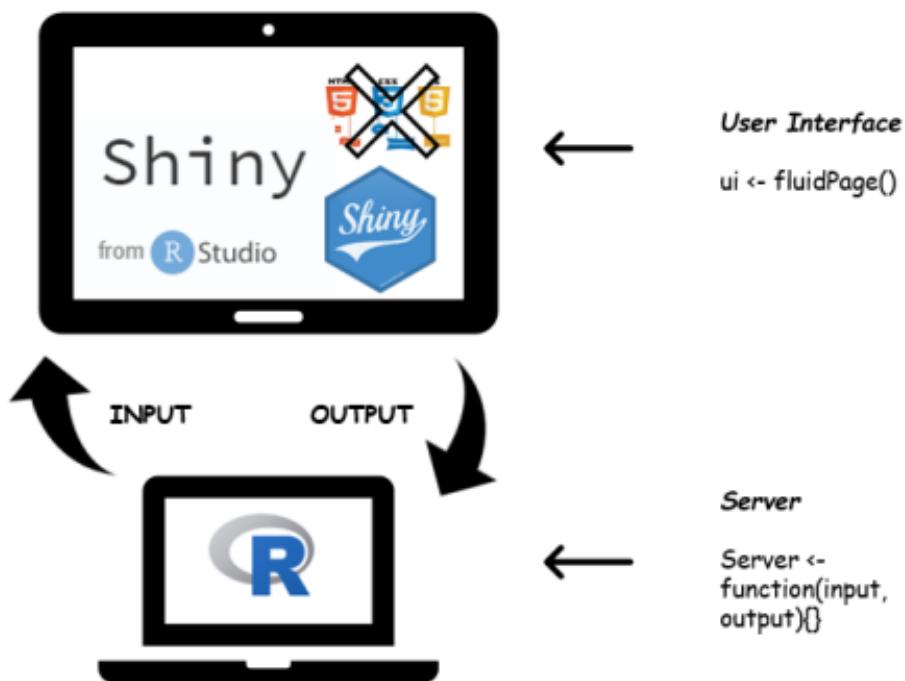
Il codice 5.1 mostra la definizione della UI dell'applicazione FlightAnalyzer.

### 5.1.2 Server

La parte server di una web app Shiny è responsabile della logica dell'applicazione e l'elaborazione dei dati. In essa sono quindi definite le funzioni e le istruzioni R che vengono eseguite quando l'utente interagisce con i componenti della UI.

Tutto ciò può essere fatto definendo delle cosiddette *"reactive expressions"*: esse sono delle espressioni che ricevono in input i valori definiti dall'utente attraverso la UI, i quali prendono il nome di *"reactive values"*, ed effettuano diverse elaborazioni su tali dati.

Nel momento in cui un *reactive value* cambia, le *reactive expression* che calcolano i dati di output vengono automaticamente eseguite e i risultati vengono aggiornati dinamicamente nell'interfaccia utente, e possono essere visualizzati sotto forma grafici, tabelle, testo o qualsiasi altro elemento definito nella UI.



**Figura 5.1:** Struttura di base di un'applicazione Shiny

**Codice 5.1:** Definizione della UI dell'applicazione. Grazie al layout *dashboardPage* è possibile definire una sidebar che permette di navigare tra le diverse pagine dell'applicazione, il cui contenuto sarà inserito all'interno di un cosiddetto *tabItem* (uno per ogni pagina)

```

1 "C"
2 ui <- dashboardPage(
3   dashboardHeader(title = "FlightAnalyzer"),
4   dashboardSidebar(
5     sidebarMenu(
6       menuItem("Dashboard", tabName = "dashboard", icon = icon(
7         "dashboard")),
8       menuItem("Ricerca", tabName = "searchAirport", icon =
9         icon("search")),
10      menuItem("Globo 3D", tabName = "3dGlobe", icon = icon("globe")),
11      % menuItem("Misure di centralita'", tabName = "
12        centrality", icon = icon("arrows-to-circle")),
13      menuItem("Ricerca di motivi", tabName = "motifSearch",
14        icon = icon("diagram-project")))
15    )
16  ),
17  dashboardBody(
18    tabItems(
19      tabItem(tabName = "dashboard",
20        ...
21      ),
22      tabItem(tabName = "searchAirport",
23        ...
24      ),
25      tabItem( tabName = "3dGlobe",
26        ...
27      ),
28      tabItem( tabName = "centrality",
29        ...
30      ),
31      tabItem( tabName = "motifSearch",
32        ...
33    )
34  )

```

# Capitolo 6

## Funzionalità dell'applicazione

Questo capitolo fornisce una panoramica dettagliata delle funzionalità offerte dall'applicazione per l'analisi del grafo che rappresenta la rete aeroportuale, grazie alle quali l'utente avrà la possibilità di esplorare e comprendere la struttura della rete.

All'interno dell'Appendice sarà poi possibile trovare alcuni screenshot relativi alle sezioni principali dell'applicazione.

### 6.1 Dashboard

La prima funzionalità offerta dall'applicazione è la **dashboard principale**, che rappresenta un'interfaccia visiva chiara e informativa per accedere a un'ampia gamma di informazioni sul grafo della rete aeroportuale. La dashboard offre un resoconto sintetico delle caratteristiche più importanti del grafo, consentendo quindi di ottenere una panoramica immediata e comprensibile delle proprietà strutturali e delle dinamiche di connettività della rete.

### 6.1.1 Statistiche della rete

La prima parte della Dashboard permette di visualizzare informazioni di base riguardanti il grafo che rappresenta la rete aeroportuale. Tra queste informazioni abbiamo:

- Numero di nodi e archi
- Coefficiente di clustering
- Assortatività
- Numero di triangoli
- Distanza media
- Diametro

#### 6.1.1.1 Nodi e archi

La rete aeroportuale risulta essere formata da:

- **6072 nodi**
- **66922 archi**

Questi numeri testimoniano l'ampia portata e la complessità della rete, sottolineando l'interconnessione tra numerosi aeroporti a livello globale.

Nello specifico, la notevole quantità di archi, ognuno dei quali equivale ad una diversa rotta aerea, mette in luce l'interdipendenza tra gli aeroporti e la loro importanza nel facilitare il movimento su scala globale.

#### 6.1.1.2 Assortatività

L'assortatività è una *misura delle correlazioni del grado dei nodi* di una rete. Più in generale, essa indica la misura in cui i vertici con le stesse proprietà si connettono tra loro.

L'assortatività varia tra  $-1$  e  $1$ , proprio come un coefficiente di correlazione. Nello specifico:

- Coefficienti di assortatività vicini a  $1$  indicano che esiste un'**alta probabilità** che due vertici con le stesse proprietà siano collegati
- Coefficienti di assortatività vicini a  $-1$  indicano che esiste una **bassa probabilità** che due vertici con le stesse proprietà siano collegati.

In questo caso, il coefficiente di assortatività della rete risulta essere pari a 0.0052700458902037, cioè prossimo a 0: tale valore indica una **rete "neutra"**, senza una preferenza evidente nella formazione delle connessioni in base al grado dei nodi.

### 6.1.1.3 Triangoli

Possiamo definire un **triangolo** come un *grafo completo con 3 nodi e 3 archi*.

L'analisi dei triangoli può essere utile per studiare la struttura del grafo, compresa la sua densità e la presenza di raggruppamenti di nodi.

In particolare, la presenza di numerosi triangoli può indicare una maggiore coesione o simmetria nel grafo.

Nello specifico, nel caso della rete aeroportuale mondiale sono stati individuati 100637 triangoli.

### 6.1.1.4 Coefficiente di clustering

Il **coefficiente di clustering**, detto anche *coefficiente di transitività*, è la misura del grado in cui i nodi di un grafo tendono ad essere connessi fra loro. Può essere misurato in due modi diversi:

- **Coefficiente di clustering locale:** dato un nodo  $n$ , si misura quanto gli adiacenti di  $n$  siano connessi tra loro.

Formalmente, il coefficiente di clustering locale di un nodo  $n$ , indicato con  $C_n$ , è dato da:

$$C_n = \frac{2L_n}{k_n \times (k_n - 1)}$$

dove:

- $k_n$  è il grado di  $n$
- $L_n$  è il numero di archi esistenti tra i  $k_n$  nodi adiacenti ad  $n$
- $C_n \in [0, 1]$

- **Coefficiente di clustering globale:** misura di clustering sull'intera rete.

Formalmente, il coefficiente di clustering globale  $C_\Delta$  è dato dal rapporto tra il **numero di triangoli** e il **numero di triple di nodi connessi fra loro**, ovvero **cammini di tre nodi oppure triangoli**.

Una misura alternativa del coefficiente di clustering dell'intera rete consiste nel calcolare il cosiddetto **coefficiente di clustering medio**, ovvero una media tra i coefficienti di clustering locali dei nodi del grafo. Formalmente, il **coefficiente di clustering medio**  $\langle C \rangle$  è definito come:

$$\langle C \rangle = \frac{1}{|V|} \sum_{n \in V} C_n$$

Nel caso della rete aeroportuale, il coefficiente di clustering globale è pari a  $0.248901665577057 \approx 0.25$

#### 6.1.1.5 Distanza media

Come suggerisce il nome stesso, la **distanza media** del grafo è calcolata come la *media delle distanze* tra tutte le possibili coppie di vertici del grafo in questione.

Formalmente, la distanza media è data da:

$$\langle d \rangle = \frac{\sum_{(u,v) \in V \times V} d(u, v)}{|V|(|V| - 1)}$$

#### 6.1.1.6 Diametro

Il **diametro** di un grafo è la *distanza massima tra una qualsiasi coppia di vertici del grafo*.

Formalmente, definiamo il diametro come:

$$\text{diameter} = \max_{u \in V} \max_{v \in V} d(u, v)$$

**Codice 6.1:** Calcolo e visualizzazione sulla UI delle misure precedentemente descritte

```

1 h4("Numero di nodi: ", length(V(graph))),  

2 h4("Numero di archi: ", length(E(graph))),  

3 h4("Numero di triangoli: ", length(triangles(graph))/3),  

4 h4("Coeficente di clustering globale: " , round(transitivity(  

    graph,type = "global",isolates = "NaN"),4)),  

5 h4("Assortatività: ", round(assortativity_degree(graph,  

    directed = TRUE),4)),  

6 h4("Distanza media: " , round(mean_distance(graph,directed =  

    TRUE),4) , " (Km)"),  

7 h4("Diametro: " , round(diameter(graph),4) , " (Km)")
```

### 6.1.2 Tipologie di tratte

La distinzione tra diverse tipologie di tratte, sulla base della distanza percorsa, può risultare molto utile per diversi motivi.

Identificare rotte a corto raggio può permettere infatti di evidenziare collegamenti tra aeroporti vicini e fornire informazioni sulle connessioni regionali; le rotte a medio raggio potrebbero indicare collegamenti tra diverse regioni o paesi limitrofi; infine, le rotte a lungo raggio potrebbero rappresentare collegamenti intercontinentali.

Questa distinzione può essere utile per valutare possibilità di ottimizzazione delle connessioni tra gli aeroporti creando nuove tratte di diversa tipologia. Per queste motivazioni, è stato inserito all'interno dell'applicazione un grafico interattivo che mostra il numero di tratte aeree per ognuna delle diverse tipologie, sulla base della loro *lunghezza in termini di distanza*.

#### 6.1.2.1 Suddivisione delle tratte

Nel visualizzare le diverse tipologie di tratte, l'utente ha la possibilità di stabilire quali sono le soglie, in termini di distanza, che permettono di suddividere le diverse tipologie di tratte aeree sulla base della loro lunghezza.

Nello specifico, saranno individuate *tre tipologie di tratte: corto raggio, medio raggio e lungo raggio*.

L'applicazione fornisce inoltre delle "soglie standard", basate su un'*analisi statistica dei valori delle distanze*: in particolare, le soglie consigliate saranno date dai valori dei **quartili della popolazione che rappresenta i possibili valori delle distanze**.

In statistica, i **quartili** sono quei valori che si trovano in posizioni tali da dividere una distribuzione in **quattro parti uguali**.

In particolare:

- Il **primo quartile**  $Q_1$  è il valore tale che il 25% delle osservazioni (nel nostro caso, i valori di lunghezza delle tratte) sono minori di esso
- Il **secondo quartile**  $Q_2$  è il valore tale che il 50% delle osservazioni (nel nostro caso, i valori di lunghezza delle tratte) sono minori di esso
- Il **terzo quartile**  $Q_3$  è il valore tale che il 75% delle osservazioni (nel nostro caso, i valori di lunghezza delle tratte) sono minori di esso

**Codice 6.2:** Suddivisione in tratte di diversa lunghezza e visualizzazione del relativo grafico

```

1 #parte UI: definizione degli "slider" che permettono di
   modificare le soglie
2
3 sliderInput("firstThresh", "Soglia voli a corto raggio", value
   = 0 , min = 0 , max = as.integer(max(E(graph)$weight)-2))
4
4 sliderInput("secondThresh", "Soglia voli a medio raggio",
   value = 0 , min = 0 , max = as.integer(max(E(graph)$weight
   )-1)),
5 checkboxInput("suggestedThresh" , "Soglie consigliate" , value
   = FALSE)
6
7 #parte Server: calcolo e visualizzazione dell'istogramma
8
9 output$threshHistogram <- renderPlot({
10    n1 <- sum(E(graph)$weight < input$firstThresh)
11    n2 <- sum(E(graph)$weight >= input$firstThresh & E(graph)$
      weight <= input$secondThresh)
12    n3 <- sum(E(graph)$weight > input$secondThresh)
13    barplot(c(n1, n2, n3), names.arg = c("Corto raggio", "
      Medio raggio", "Lungo raggio") , col = c("red","blue",
      "green"))
14 })

```

### 6.1.3 Grafici vari

La successiva sezione della Dashboard è dedicata alla visualizzazione di diversi grafici che mostrano diverse informazioni riguardanti la rete aeroportuale. Tra essi abbiamo dei grafici che mostrano:

- le compagnie che operano più rotte
- gli stati al mondo con più aeroporti
- le città al mondo con più aeroporti

**Codice 6.3:** Calcolo e visualizzazione dei grafici

```

1 #parte UI
2
3 box(width = 4,plotOutput("mostActiveAirlines")),
4 box(width = 4,plotOutput("mostDenseCountry")),
5 box(width = 4,plotOutput("denseCities"))
6
7 #parte Server
8
9 output$mostActiveAirlines <- renderPlot({
10   tableAirlines <- table(get.edge.attribute(graph , "airline
11   "))
11   topAirlines <- head(sort(tableAirlines , decreasing=T) ,
12   3)
12   names(topAirlines) = c(na.omit(airlinesData[airlinesData$  

13   IATA == names(topAirlines)[1],])$name , na.omit(
14   airlinesData[airlinesData$IATA == names(topAirlines)
15   [2],])$name , na.omit(airlinesData[airlinesData$IATA
16   == names(topAirlines)[3],])$name)
13   barplot(topAirlines, main="Compagnie con piu' rotte", xlab
17   ="Compagnia", ylab="# rotte", col="lightblue")
14 })
15
16 output$mostDenseCountry <- renderPlot({
17   tableCountry <- table(airportsData[airportsData$country !=  

18   "",]$country)
18   topCountry <- head(sort(tableCountry , decreasing=T) , 3)
19   barplot(topCountry, main="Stati con piu' aeroporti", xlab=
20   "Stato", ylab="# aeroporti", col="lightblue")
20 })
21
22 output$denseCities <- renderPlot({
23   tableCities <- table(airportsData[airportsData$city != ""
24   ,$city])
24   topCities <- head(sort(tableCities , decreasing=T) , 3)
25   barplot(topCities, main="Citta' con piu' aeroporti", xlab=
26   "Citta'", ylab="# aeroporti", col="lightblue")
26 })

```

### 6.1.4 Distribuzione dei gradi

L'ultima sezione della Dashboard è dedicata alla visualizzazione della **distribuzione dei gradi** dei nodi del grafo.

La distribuzione viene mostrata all'utente tramite un *istogramma* in cui:

- Sull'asse X abbiamo il grado dei nodi
- Sull'asse Y abbiamo il numero di nodi che hanno uno specifico grado

È inoltre possibile selezionare un range di valori di interesse dell'utente per quanto riguarda il grado dei nodi.

**Codice 6.4:** Calcolo e visualizzazione della distribuzione dei gradi dei nodi del grafo

```

1 #parte UI: vengono definiti, oltre al grafico, anche i comandi
  interattivi per l'utente
2
3 box(width = 12,
4      div(h2("Distribuzione dei gradi"), style = "text-align:
        center;"),
5      checkboxInput("showZero", "Considera nodi isolati", value=
        FALSE),
6      numericInput("minDegree", "Grado minimo da considerare" ,
        min=0, max=max(degree(graph), v = V(graph)), mode =
        "all"))-1, value=1),
7      numericInput("maxDegree", "Grado massimo da considerare" ,
        min=1, max=max(degree(graph), v = V(graph)), mode =
        "all")), value=max(degree(graph), v = V(graph)), mode =
        "all"))),
8      plotOutput("degreeDistribution")
9 )
10
11 #parte Server: oltre a visualizzare il grafico, quest'ultimo
  viene modificato in maniera "reactive" ogni volta che l'
  utente modifica i parametri dall'UI
12
13 output$degreeDistribution <- renderPlot({
14   newDegreeMeasure <- degreeMeasure
15   if(!input$showZero) {
16     newDegreeMeasure <- degreeMeasure[degreeMeasure != 0] # 
        rimuovo nodi isolati
17   }
18   degreeFrequencies <- table(newDegreeMeasure)
19   degreeFrequencies <- subset(degreeFrequencies, names(
        degreeFrequencies) >= minDegree() & names(
        degreeFrequencies) <= maxDegree())

```

```

20     barplot(degreeFrequencies, main="Distribuzione dei gradi"
21     , xlab="Grado", ylab="# nodi", col="blue" , las=2)
22   })
23 observeEvent(input$minDegree , {
24   output$degreeDistribution <- renderPlot({
25     newDegreeMeasure <- degreeMeasure
26     if (!input$showZero) {
27       newDegreeMeasure <- degreeMeasure[degreeMeasure != 0]
28       #rimuovo nodi isolati
29     }
30     degreeFrequencies <- table(newDegreeMeasure)
31     degreeFrequencies <- subset(degreeFrequencies, as.integer(
32       names(degreeFrequencies)) >= minDegree() & as.integer(
33       names(degreeFrequencies)) <= maxDegree())
34     barplot(degreeFrequencies, main="Distribuzione dei gradi"
35     , xlab="Grado", ylab="# nodi", col="blue" , las=2)
36   })
37 })
38 observeEvent(input$maxDegree , {
39   output$degreeDistribution <- renderPlot({
40     newDegreeMeasure <- degreeMeasure
41     if (!input$showZero) {
42       newDegreeMeasure <- degreeMeasure[degreeMeasure != 0]
43       #rimuovo nodi isolati
44     }
45     degreeFrequencies <- table(newDegreeMeasure)
46     degreeFrequencies <- subset(degreeFrequencies, as.integer(
47       names(degreeFrequencies)) >= minDegree() & as.integer(
48       names(degreeFrequencies)) <= maxDegree())
49     barplot(degreeFrequencies, main="Distribuzione dei gradi"
50     , xlab="Grado", ylab="# nodi", col="blue" , las=2)
51   })
52 })

```

## 6.2 Ricerca di aeroporti

Una delle funzionalità principali dell'applicazione è quella di consentire la ricerca di aeroporti all'interno della rete, al fine di ottenere diverse informazioni su di essi.

A tal fine, è consentito all'utente ricercare gli aeroporti sulla base di diverse informazioni riguardanti l'aeroporto cercato.

I possibili parametri di ricerca sono:

- Codice IATA
- Codice ICAO
- Nome
- Nazione
- Compagnie aeree che operano in tale aeroporto

Sulla base dei parametri scelti, l'applicazione restituirà:

- una lista degli aeroporti della rete che soddisfano i parametri inseriti dall'utente;
- una cartina geografica interattiva sulla quale sono riportate le posizioni geografiche degli aeroporti trovati. È inoltre possibile cliccare sui pin della mappa che rappresentano i singoli aeroporti per ottenere informazioni aggiuntive su questi ultimi;
- una lista delle compagnie aeree che servono gli aeroporti trovati.

#### **Codice 6.5:** Ricerca di aeroporti

```

40     }
41   else if(key() == "Compagnia aerea") {
42     edges <- E(graph) [E(graph)$airline == airlinesData[
43       airlinesData$name == input$value , ]$IATA]
44     air <- unlist(unique(as.list(ends(graph , edges))))
45     occ <- airportsData[airportsData$name %in% air, ]
46   }
47   if(nrow(occ)>0) {
48     leaflet(data = occ) %>%
49       addProviderTiles(providers$CartoDB.Positron) %>%
50       addMarkers(~longitude, ~latitude, popup = paste(occ$  

51         name, "<br>" , occ$country , "<br>IATA: ", occ$  

52         IATA , "<br>ICAO: ", occ$ICAO) , clusterOptions =  

53           markerClusterOptions())
54   }
55 }
56 })
57
58 output$tabledata <- renderDT({
59   if(input$value!=""){
60     if(key() == "Nome"){
61       occ <- airportsData[grep(input$value, airportsData$  

62         name , ignore.case = TRUE), ][,c("IATA","ICAO","  

63         name","city","country")]
64     }
65     else if(key() == "Nazione"){
66       occ <- airportsData[grep(input$value, airportsData$  

67         country , ignore.case = TRUE), ][,c("IATA","ICAO","  

68         name","city","country")]
69     }
70     else if(key() == "IATA" | key() == "ICAO"){
71       occ <- airportsData[airportsData[,key()] == toupper(  

72         input$value), ][,c("IATA","ICAO","name","city","  

73         country")]
74     }
75     else if(key() == "Compagnia aerea"){
76       edges <- E(graph) [E(graph)$airline == airlinesData[
77         airlinesData$name == input$value , ]$IATA]
78       air <- unlist(unique(as.list(ends(graph , edges))))
79       occ <- airportsData[airportsData$name %in% air, ][,c("IATA","ICAO","name","city","country")]
80     }
81     if(nrow(occ) > 0 ){
82       nRes <- nrow(occ)
83     }
84   }
85 }
86 })

```

## 6.3 Visualizzazione 3D della rete

Al fine di comprendere a pieno la struttura della rete aeroportuale, può risultare molto utile avere una rappresentazione semplice e intuitiva della struttura della rete. Per questo motivo, una sezione dell'applicazione FlightAnalyzer è dedicata alla **visualizzazione interattiva in 3D della rete**.

Attraverso l'uso della libreria **Globe4R**, viene costruito un **globo 3D**, sul quale vengono collocati dei PIN in corrispondenza dei diversi aeroporti. Sono poi messe a disposizione diverse opzioni aggiuntive attivabili dall'utente:

- Visualizzazione di tutti gli aeroporti
- Visualizzazione degli aeroporti di specifici continenti
- Possibilità di mostrare le diverse tratte, specificando un valore massimo di lunghezza.

È inoltre possibile visualizzare diverse informazioni riguardanti i singoli aeroporti della rete, semplicemente evidenziandoli col puntatore del mouse.

Le informazioni visualizzate sono nello specifico **nome**, **città** e **stato** dell'aeroporto evidenziato.

### Codice 6.6: Visualizzazione 3D della rete aeroportuale

```

1 #parte UI: vengono definiti, oltre al globo 3D, anche i
  comandi interattivi per l'utente
2
3 column( width = 5,
4   box(
5     radioButtons("filters","Filtrri",choiceNames = list(
6       "Tutti gli aeroporti","Per continente"),
7       choiceValues = list("all","continent")),
8     conditionalPanel(condition = "input.filters == '"
9       "continent'",
10      checkboxGroupInput("continentCheckbox","Seleziona
11        il Continente:",choices = c("Europa"="EL", "
12          Asia"="UOZVRW", "Africa"="GDHF", "America"=
13            "BCKMTS", "Oceania"="YAPN")
14      )
15    )
16  ),
17  checkboxInput("globeArcs" , "Mostra rotte"),
18  conditionalPanel(condition = "input.globeArcs",
19    sliderInput("maxLength" , "Lunghezza massima rotte
20      visualizzate",value = floor(min(E(graph)$weight)))
21  )
22 )
```

```

+500 , min = floor(min(E(graph)$weight)) , max =
ceiling(max(E(graph)$weight)))
14      )
15 ),
16 column( width = 1,
17   globeOutput("globe", width = "650px" , height = "650px")
18 )
19
20 #parte Server
21
22 output$globe <- renderGlobe({
23   temp <- airportsData[,c(1,3,4,5,6,7,8)]
24   airportsDataSubset <- temp
25   globeArcs <- data.frame(yFrom = double() , xFrom = double()
26     , yTo = double() , xTo = double())
27   if (input$filter == "continent"){
28     if(length(input$continentCheckbox) !=0){
29       validChars <- strsplit(paste(input$continentCheckbox,
30         collapse=""), "")[[1]]
31       airportsDataSubset <- temp[substr(temp$ICAO, 1, 1) %in%
32         % validChars, ]
33     }
34   }
35   if(input$globeArcs == TRUE){
36     df <- as_data_frame(get.edgelist(graph, names = FALSE) ,
37       get.edge.attribute(graph, name="weight"))
38     weights <- get.edge.attribute(graph, name="weight")
39     arcs <- cbind(df,weights)
40     arcs <- arcs[arcs$weights <= maxLength() & get.vertex.
41       attribute(graph , "name" , arcs$V1) %in%
42       airportsDataSubset$name & get.vertex.attribute(graph
43         , "name" , arcs$V2) %in% airportsDataSubset$name,
44       ]
45     xFrom <- V(graph)[arcs$V1]$longitude
46     yFrom <- V(graph)[arcs$V1]$latitude
47     xTo <- V(graph)[arcs$V2]$longitude
48     yTo <- V(graph)[arcs$V2]$latitude
49     globeArcs <- cbind(yFrom,xFrom,yTo,xTo)
50   }
51   airportsDataSubset <- as.data.frame(sapply(
52     airportsDataSubset,as.character))
53   airportsDataSubset$labels <- apply(airportsDataSubset, 1,
54     function(row) {
55       values <- row[c(2,3,4)]
56       names <- names(airportsDataSubset)[c(2,3,4)]
57       combined <- paste(sprintf("%s: %s", names, values),
58         collapse = " ; ")
59       combined
60     })

```

```
50     create_globe() %>%
51       globe_arcs(
52         coords(
53           start_lat = yFrom,
54           start_lon = xFrom,
55           end_lat = yTo,
56           end_lon = xTo,
57           stroke = 0.5
58         ),
59         data = as.data.frame(globeArcs)
60       ) %>%
61       globe_bars(
62         coords(latitude, longitude, label = labels),
63         data = airportsDataSubset
64       )
65   })
```

## 6.4 Misure di centralità

Le misure di centralità sono metriche utilizzate per quantificare l'importanza o l'influenza di un nodo all'interno di un grafo. Tali misure sono quindi finalizzate a identificare i nodi che hanno un ruolo in qualche modo significativo nella struttura o nel funzionamento della rete.

Più nello specifico, le misure di centralità restituiscono valore numerico per ogni nodo, il quale indica la centralità del nodo in questione rispetto agli altri nodi del grafo.

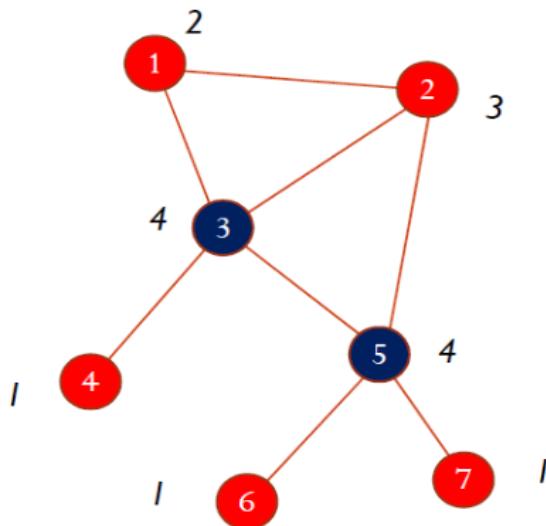
Possiamo quindi affermare che una misura di centralità, nella pratica, indica "quanto è importante" un nodo all'interno della rete di appartenenza.

Esistono diverse misure di centralità, ognuna delle quali adotta un approccio diverso per valutare l'importanza dei nodi.

### 6.4.1 Degree Centrality

La Degree Centrality è una misura di centralità dei nodi di un grafo che tiene conto del grado di ciascun nodo, così come definito in sez.2.2.2.

Nella pratica, più alto è il grado di un nodo, più esso è considerato "importante".



**Figura 6.1:** Esempio di calcolo di Degree Centrality

### 6.4.2 Betweenness Centrality

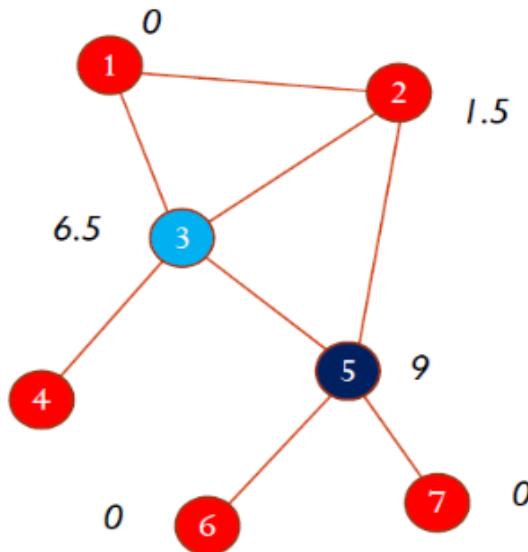
La Betweenness Centrality è una misura di centralità dei nodi che tiene conto del numero di volte che un nodo è presente nel cammino minimo, così come definito in sez.2.2.3, tra due nodi qualsiasi del grafo. In altre parole, un nodo ha una alta Betweenness Centrality se molti cammini minimi attraversano il nodo in questione.

La Betweenness Centrality di un generico nodo  $u$  è data da:

$$B(u) = \sum_{s,t \in V, s \neq u \neq t} \frac{\sigma_{st}(u)}{\sigma_{st}}$$

dove:

- $\sigma_{st}$  è il **numero di cammini minimi dal nodo  $s$  al nodo  $t$**
- $\sigma_{st}(u)$  è il **numero di cammini minimi da  $s$  a  $t$  che passano per il nodo  $u$**



**Figura 6.2:** Esempio di calcolo di Betweenness Centrality

### 6.4.3 Closeness Centrality

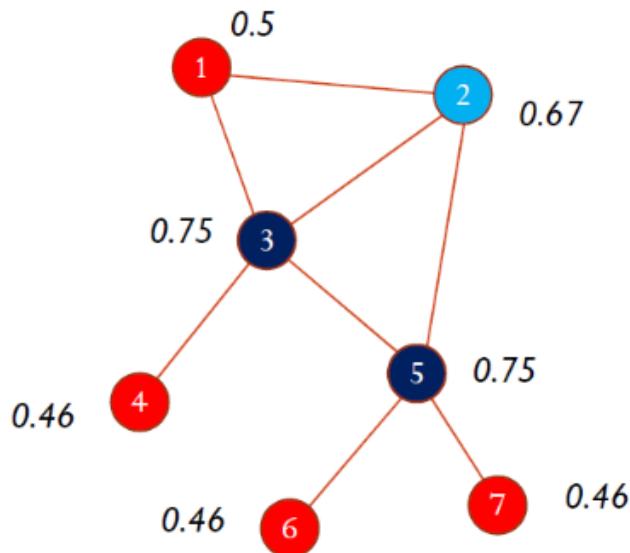
La Closeness Centrality è una misura di centralità dei nodi che si basa sulla vicinanza media di un nodo rispetto a tutti gli altri nodi del grafo.

In pratica, la closeness centrality di un nodo  $v$  è una misura della "velocità media" con cui un'informazione, partendo da  $v$ , può raggiungere tutti gli altri nodi del grafo.

Per definire la Closeness Centrality di un nodo  $u$ , si deve prima di tutto calcolare la **lunghezza media  $L_u$**  dei cammini minimi tra  $u$  e gli altri nodi del grafo.

La Closeness Centrality di  $u$  è poi definita come il **reciproco di  $L_u$** :

$$C(u) = \frac{1}{L_u}$$



**Figura 6.3:** Esempio di calcolo di Closeness Centrality

#### 6.4.4 PageRank Centrality

La PageRank Centrality è una misura di centralità che si basa sull'osservazione che le connessioni di un nodo con gli altri nodi non hanno tutte lo stesso valore.

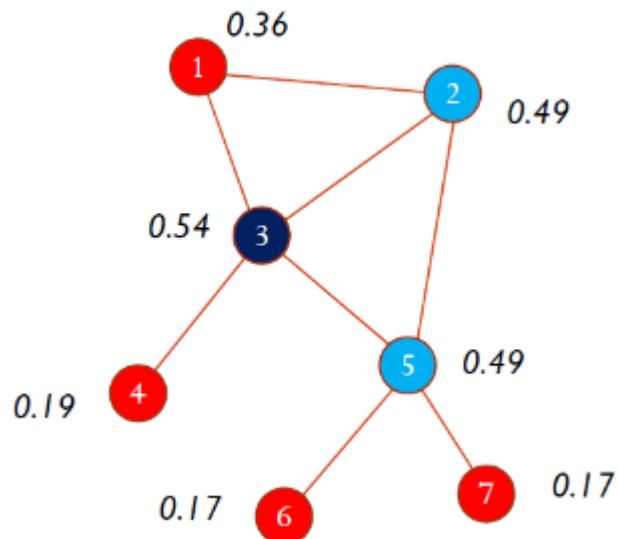
Infatti, connessioni a nodi con elevato grado hanno un «peso» maggiore rispetto alle connessioni a nodi di grado minore. Possiamo anche dire, in “maniera ricorsiva”, che un nodo è tanto più importante quanto più è connesso ad altri nodi importanti nella rete.

Definiamo la PageRank Centrality di un nodo  $u$  come:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{k_v}$$

dove:

- $B_u$  è l'insieme dei nodi che hanno  $u$  come adiacente
- $k_v$  è il grado uscente di  $v$



**Figura 6.4:** Esempio di calcolo di PageRank Centrality

### 6.4.5 Implementazione del calcolo delle misure di centralità

L'applicazione permette di calcolare le diverse misure di centralità per ognuno dei vertici del grafo.

Nello specifico, sono mostrati i ***nodi che massimizzano le singole misure di centralità***, e un ***globo 3D interattivo*** in cui l'utente può visualizzare il valore di una specifica misura di centralità relativo ai singoli aeroporti della rete.

Inoltre, i PIN rappresentanti i singoli aeroporti sul globo avranno "altezze" in scala rispetto al valore di centralità del nodo in questione.

**Codice 6.7:** Calcolo e visualizzazione della centralità dei nodi

```

1 #parte UI
2
3 fluidRow(
4   box(width = 6,
5     div(h2("DEGREE CENTRALITY"), style = "text-align:
6       center;"),
7     verbatimTextOutput("degree")
8   ),
9   box(width = 6,
10     div(h2("CLOSENESS CENTRALITY"), style = "text-align:
11       center;"),
12     verbatimTextOutput("closeness")
13   )
14 ),
15 fluidRow(
16   box(width = 6,
17     div(h2("BETWEENNESS CENTRALITY"), style = "text-align:
18       center;"),
19     verbatimTextOutput("betwennes")
20   ),
21   box(width = 6,
22     div(h2("PAGE-RANK CENTRALITY"), style = "text-align:
23       center;"),
24     verbatimTextOutput("pagerank")
25   )
26 ),
27 fluidRow(
28   column( width = 5,
29   box(
30     radioButtons("measureFilters", "Misura", choiceNames
31     = list("Degree C.", "Closeness C.", "Betweennes

```

```

            C.", "PageRank C."), choiceValues = list("
            degree", "closeness", "betweennes", "pagerank"))
27        ),
28        column( width = 1,
29                 globeOutput("measureGlobe", width = "650px" ,
30                           height = "650px")
31        )
32    )
33
34 #parte Server
35
36 output$betweennes <- renderText({
37     betweennesMeasure <- betweenness(graph, v = V(graph),
38         directed = TRUE, weights = E(graph)$weight, normalize =
39         TRUE)
40     max_index <- which.max(betweennesMeasure)
41     mostCentralAirport <- names(betweennesMeasure) [max_index]
42     paste("Aeroporto che massimizza la Betweenness Centrality:\n",
43           mostCentralAirport, "\nValore: " ,
44           betweennesMeasure[mostCentralAirport])
45   })
46
47 output$degree <- renderText({
48     newDegreeMeasure <- degreeMeasure
49     if (!input$showZero) {
50         newDegreeMeasure <- degreeMeasure[degreeMeasure != 0] #
51             rimuovo nodi isolati
52     }
53     max_index <- which.max(newDegreeMeasure)
54     highestDegree <- names(newDegreeMeasure) [max_index]
55     paste("Aeroporto che massimizza la Degree Centrality:\n",
56           highestDegree , "(Grado:" , newDegreeMeasure[
57           highestDegree] , ") \nGrado medio: " , mean(  

58           newDegreeMeasure))
59   })
60
61 output$closeness <- renderText({
62     closenessMeasure <- closeness(graph, v = V(graph), weights =
63         E(graph)$weight, normalized = TRUE)
64     max_index <- which.max(closenessMeasure)
65     mostCLoseAirport <- names(closenessMeasure) [max_index]
66     paste("Aeroporto che massimizza la Closeness Centrality:\n",
67           mostCLoseAirport , "\nValore: " , closenessMeasure
68           [mostCLoseAirport]))
69   })
70
71 output$pagerank <- renderText ({
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1103
1104
1105
1105
1106
1107
1107
1108
1109
1109
1110
1111
1111
1112
1113
1113
1114
1115
1115
1116
1117
1117
1118
1119
1119
1120
1121
1121
1122
1123
1123
1124
1125
1125
1126
1127
1127
1128
1129
1129
1130
1131
1131
1132
1133
1133
1134
1135
1135
1136
1137
1137
1138
1139
1139
1140
1141
1141
1142
1143
1143
1144
1145
1145
1146
1147
1147
1148
1149
1149
1150
1151
1151
1152
1153
1153
1154
1155
1155
1156
1157
1157
1158
1159
1159
1160
1161
1161
1162
1163
1163
1164
1165
1165
1166
1167
1167
1168
1169
1169
1170
1171
1171
1172
1173
1173
1174
1175
1175
1176
1177
1177
1178
1179
1179
1180
1181
1181
1182
1183
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1191
1192
1193
1193
1194
1195
1195
1196
1197
1197
1198
1199
1199
1200
1201
1201
1202
1203
1203
1204
1205
1205
1206
1207
1207
1208
1209
1209
1210
1211
1211
1212
1213
1213
1214
1215
1215
1216
1217
1217
1218
1219
1219
1220
1221
1221
1222
1223
1223
1224
1225
1225
1226
1227
1227
1228
1229
1229
1230
1231
1231
1232
1233
1233
1234
1235
1235
1236
1237
1237
1238
1239
1239
1240
1241
1241
1242
1243
1243
1244
1245
1245
1246
1247
1247
1248
1249
1249
1250
1251
1251
1252
1253
1253
1254
1255
1255
1256
1257
1257
1258
1259
1259
1260
1261
1261
1262
1263
1263
1264
1265
1265
1266
1267
1267
1268
1269
1269
1270
1271
1271
1272
1273
1273
1274
1275
1275
1276
1277
1277
1278
1279
1279
1280
1281
1281
1282
1283
1283
1284
1285
1285
1286
1287
1287
1288
1289
1289
1290
1291
1291
1292
1293
1293
1294
1295
1295
1296
1297
1297
1298
1299
1299
1300
1301
1301
1302
1303
1303
1304
1305
1305
1306
1307
1307
1308
1309
1309
1310
1311
1311
1312
1313
1313
1314
1315
1315
1316
1317
1317
1318
1319
1319
1320
1321
1321
1322
1323
1323
1324
1325
1325
1326
1327
1327
1328
1329
1329
1330
1331
1331
1332
1333
1333
1334
1335
1335
1336
1337
1337
1338
1339
1339
1340
1341
1341
1342
1343
1343
1344
1345
1345
1346
1347
1347
1348
1349
1349
1350
1351
1351
1352
1353
1353
1354
1355
1355
1356
1357
1357
1358
1359
1359
1360
1361
1361
1362
1363
1363
1364
1365
1365
1366
1367
1367
1368
1369
1369
1370
1371
1371
1372
1373
1373
1374
1375
1375
1376
1377
1377
1378
1379
1379
1380
1381
1381
1382
1383
1383
1384
1385
1385
1386
1387
1387
1388
1389
1389
1390
1391
1391
1392
1393
1393
1394
1395
1395
1396
1397
1397
1398
1399
1399
1400
1401
1401
1402
1403
1403
1404
1405
1405
1406
1407
1407
1408
1409
1409
1410
1411
1411
1412
1413
1413
1414
1415
1415
1416
1417
1417
1418
1419
1419
1420
1421
1421
1422
1423
1423
1424
1425
1425
1426
1427
1427
1428
1429
1429
1430
1431
1431
1432
1433
1433
1434
1435
1435
1436
1437
1437
1438
1439
1439
1440
1441
1441
1442
1443
1443
1444
1445
1445
1446
1447
1447
1448
1449
1449
1450
1451
1451
1452
1453
1453
1454
1455
1455
1456
1457
1457
1458
1459
1459
1460
1461
1461
1462
1463
1463
1464
1465
1465
1466
1467
1467
1468
1469
1469
1470
1471
1471
1472
1473
1473
1474
1475
1475
1476
1477
1477
1478
1479
1479
1480
1481
1481
1482
1483
1483
1484
1485
1485
1486
1487
1487
1488
1489
1489
1490
1491
1491
1492
1493
1493
1494
1495
1495
1496
1497
1497
1498
1499
1499
1500
1501
1501
1502
1503
1503
1504
1505
1505
1506
1507
1507
1508
1509
1509
1510
1511
1511
1512
1513
1513
1514
1515
1515
1516
1517
1517
1518
1519
1519
1520
1521
1521
1522
1523
1523
1524
1525
1525
1526
1527
1527
1528
1529
1529
1530
1531
1531
1532
1533
1533
1534
1535
1535
1536
1537
1537
1538
1539
1539
1540
1541
1541
1542
1543
1543
1544
1545
1545
1546
1547
1547
1548
1549
1549
1550
1551
1551
1552
1553
1553
1554
1555
1555
1556
1557
1557
1558
1559
1559
1560
1561
1561
1562
1563
1563
1564
1565
1565
1566
1567
1567
1568
1569
1569
1570
1571
1571
1572
1573
1573
1574
1575
1575
1576
1577
1577
1578
1579
1579
1580
1581
1581
1582
1583
1583
1584
1585
1585
1586
1587
1587
1588
1589
1589
1590
1591
1591
1592
1593
1593
1594
1595
1595
1596
1597
1597
1598
1599
1599
1600
1601
1601
1602
1603
1603
1604
1605
1605
1606
1607
1607
1608
1609
1609
1610
1611
1611
1612
1613
1613
1614
1615
1615
1616
1617
1617
1618
1619
1619
1620
1621
1621
1622
1623
1623
1624
1625
1625
1626
1627
1627
1628
1629
1629
1630
1631
1631
1632
1633
1633
1634
1635
1635
1636
1637
1637
1638
1639
1639
1640
1641
1641
1642
1643
1643
1644
1645
1645
1646
1647
1647
1648
1649
1649
1650
1651
1651
1652
1653
1653
1654
1655
1655
1656
1657
1657
1658
1659
1659
1660
1661
1661
1662
1663
1663
1664
1665
1665
1666
1667
1667
1668
1669
1669
1670
1671
1671
1672
1673
1673
1674
1675
1675
1676
1677
1677
1678
1679
1679
1680
1681
1681
1682
1683
1683
1684
1685
1685
1686
1687
1687
1688
1689
1689
1690
1691
1691
1692
1693
1693
1694
1695
1695
1696
1697
1697
1698
1699
1699
1700
1701
1701
1702
1703
1703
1704
1705
1705
1706
1707
1707
1708
1709
1709
1710
1711
1711
1712
1713
1713
1714
1715
1715
1716
1717
1717
1718
1719
1719
1720
1721
1721
1722
1723
1723
1724
1725
1725
1726
1727
1727
1728
1729
1729
1730
1731
1731
1732
1733
1733
1734
1735
1735
1736
1737
1737
1738
1739
1739
1740
1741
1741
1742
1743
1743
1744
1745
1745
1746
1747
1747
1748
1749
1749
1750
1751
1751
1752
1753
1753
1754
1755
1755
1756
1757
1757
1758
1759
1759
1760
1761
1761
1762
1763
1763
1764
1765
1765
1766
1767
1767
1768
1769
1769
1770
1771
1771
1772
1773
1773
1774
1775
1775
1776
1777
1777
1778
1779
1779
1780
1781
1781
1782
1783
1783
1784
1785
1785
1786
1787
1787
1788
1789
1789
1790
1791
1791
1792
1793
1793
1794
1795
1795
1796
1797
1797
1798
1799
1799
1800
1801
1801
1802
1803
1803
1804
1805
1805
1806
1807
1807
1808
1809
1809
1810
1811
1811
1812
1813
1813
1814
1815
1815
1816
1817
1817
1818
1819
1819
1820
1821
1821
1822
1823
1823
1824
1825
1825
1826
1827
1827
1828
1829
1829
1830
1831
1831
1832
1833
1833
1834
1835
1835
1836
1837
1837
1838
1839
1839
1840
1841
1841
1842
1843
1843
1844
1845
1845
1846
1847
1847
1848
1849
1849
1850
1851
1851
1852
1853
1853
1854
1855
1855
1856
1857
1857
1858
1859
1859
1860
1861
1861
1862
1863
1863
1864
1865
1865
1866
1867
1867
1868
1869
1869
1870
1871
1871
1872
1873
1873
1874
1875
1875
1876
1877
1877
1878
1879
1879
1880
1881
1881
1882
1883
1883
1884
1885
1885
1886
1887
1887
1888
1889
1889
1890
1891
1891
1892
1893
1893
1894
1895
1895
```

```

61 pagerankMeasure <- page_rank(graph, algo = "prpack", v = V(
62   graph), weights = E(graph)$weight)
63 max_index <- which.max(pagerankMeasure$bvector)
64 highestPageRank <- names(pagerankMeasure$bvector)[max_index
65   ]
66 paste("Aeroporto che massimizza la Page-Rank Centrality:\n",
67   ", highestPageRank, "\nValore: ", pagerankMeasure$b
68   vector[highestPageRank])
69 })
70
71 output$measureGlobe <- renderGlobe({
72   airportsList <- airportsData[,c(3,7,8)]
73   indici <- match(airportsList$name, V(graph)$name)
74   if (input$measureFilters == "degree"){
75     val <- degree(graph)[indici]
76     airportsList <- cbind(airportsList, values = val)
77   }
78   else if (input$measureFilters == "closeness"){
79     val <- closeness(graph,v = V(graph),weights = E(graph)$
80       weight,normalized = TRUE)[indici]
81     airportsList <- cbind(airportsList, values = val)
82   }
83   else if (input$measureFilters == "betweennes"){
84     val <- betweenness(graph,v = V(graph),directed = TRUE,
85       weights = E(graph)$weight,normalize = TRUE)[indici]
86     airportsList <- cbind(airportsList, values = val)
87   }
88   else if (input$measureFilters == "pagerank"){
89     val <- page_rank(graph,algo = "prpack",v = V(graph),
90       weights = E(graph)$weight)$vector[indici]
91     airportsList <- cbind(airportsList, values = val)
92   }
93   airportsList <- as.data.frame(sapply(airportsList,as.
94     character))
95   airportsList$labels <- paste(airportsList$name, "\n",
96     airportsList$values)
97   airportsList$values <- as.double(airportsList$values)
98   create_globe()%>%
99     globe_bars(
100       coords(latitude,longitude, label = labels , altitude =
101         values),
102       data = airportsList
103     )%>%
104       scale_bars_altitude()
105   })

```

## 6.5 Ricerca di motivi

La "ricerca dei motivi" è una particolare variante del problema del Graph Mining, in cui il database dei grafi contiene un unico grafo.

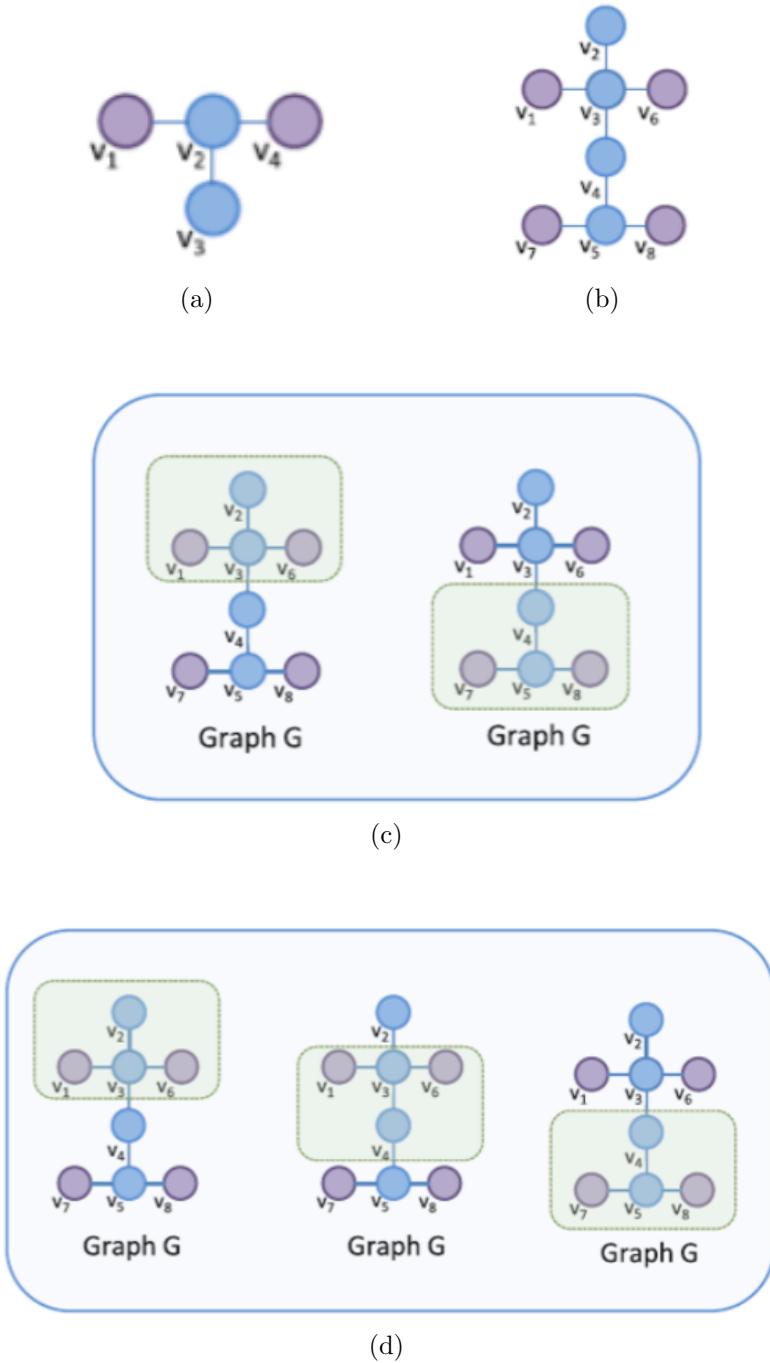
Possiamo quindi riformulare il problema della ricerca di motivi nel seguente modo: dato un grafo  $G$ , si vogliono trovare tutti i sottografi frequenti in  $G$ . La frequenza in questo caso viene definita in termini di *numero di occorrenze di un sottografo in  $G$* .

Quando un sottografo risulta essere frequente in  $G$ , esso prende il nome di "**motivo**".

Le occorrenze di un sottografo possono essere contate in due modi:

- **Senza overlap:** non si hanno sovrapposizioni di archi e/o nodi tra i diversi sottografi candidati
- **Con overlap:** si possono avere delle sovrapposizioni di archi e/o nodi tra i diversi sottografi candidati

A seconda del metodo con cui vengono contate le occorrenze, l'output dell'algoritmo, applicato sullo stesso grafo di partenza, potrà essere diverso, come si può notare dall'esempio in figura Figura 6.5.



**Figura 6.5:** (a) Sottografo candidato (b) Grafo iniziale (c) Ricerca senza overlap delle occorrenze (d) Ricerca con overlap delle occorrenze

### 6.5.1 Significatività statistica nel mining di un singolo grafo

Come già visto nel caso del FSM (sez. 2.4.2), anche nel caso di mining in un singolo grafo si ha la necessità di verificare se un sottografo frequente è in qualche modo rilevante.

In maniera simile al caso precedente, ciò può essere fatto calcolando con opportuni metodi la **"significatività statistica"** del sottografo in questione.

### 6.5.2 Strategie di ricerca dei motivi

Per effettuare l'operazione di ricerca di motivi all'interno di un grafo esistono numerosi algoritmi, i quali applicano una tra le seguenti **strategie di ricerca**:

- **Strategia "network-centric"**: consiste nel ricercare ed enumerare tutti i possibili sottografi con un certo numero  $n$  di nodi che occorrono nel grafo iniziale, per poi verificare i "match isomorfi": tale operazione consiste nel raggruppare occorrenze di diversi sottografi che hanno la stessa struttura.
  - **Vantaggi**: i sottografi non presenti nel grafo da analizzare non vengono mai considerati.
  - **Svantaggi**: occorre individuare tutti i sottografi nel grafo da analizzare: questa operazione può risultare molto costosa in termini di tempo e risorse computazionali.
- **Strategia "motif-centric"**: consiste nell'enumerare tutti i possibili sottografi con un certo numero  $n$  di nodi e ricercarli uno per uno nel grafo iniziale.
  - **Vantaggi**: è possibile verificare in maniera diretta se un sottografo è un motivo o meno.
  - **Svantaggi**: l'enumerazione dei sottografi diventa sempre più costosa in termini di tempo e risorse computazionali all'aumentare di  $n$ .
- **Strategia "set-centric"**: consiste nel ricercare un insieme di sottografi e, con un'unica passata sul grafo iniziale, trovare tutti i match di questi sottografi, per poi raggruppare tutti i match isomorfi.  
Tale strategia è nella pratica un mix delle due precedenti.

### 6.5.3 Algoritmo "MultiMotif"

Come spiegato nella sez. 4.1, la rete aeroportuale mondiale può essere trattata come un ***multigrafo diretto ed etichettato***.

Nello specifico, consideriamo:

- ***nome*** e ***nazione*** come etichette dei nodi
- ***tipologia di tratta (corto, medio o lungo raggio)*** come etichette degli archi

Nell'ambito della ricerca di motivi statisticamente significativi, i più comuni algoritmi hanno problemi legati all'efficienza della ricerca all'interno di multigrafi etichettati.

Sono quindi necessari degli algoritmi appositi che permettano di trovare motivi significativi in tale contesto. Tra questi algoritmi, quello implementato in FlightAnalyzer è l'algoritmo ***MultiMotif*** [9].

MultiMotif è un algoritmo che permette di trovare motivi statisticamente significativi all'interno di multigrafi, diretti e non, e di calcolare in maniera analitica il relativo p-value.

Più nello specifico, MultiMotif fa uso di una versione customizzata dell'algoritmo di graph matching **RI** [10] per il conteggio delle occorrenze, ed implementa un metodo per il calcolo della significatività statistica dei singoli motivi ***senza generare grafi random***.

Sono di seguito riportati i passi fondamentali dell'algoritmo MultiMotif. Per maggiori dettagli, consultare l'apposito articolo [9].

#### 6.5.3.1 Modello "Expected Degree Distribution" per multigrafi etichettati

L'approccio di base dell'algoritmo per *determinare quali sottografi in un dato grafo  $G$  sono over-represented* si basa su un cosiddetto modello ***EDD*** [11], il quale **dipende dalle etichette di nodi e archi**. Tale modello viene esteso per funzionare con i multigrafi, ed è realizzato sulla base del calcolo del numero atteso (e della corrispondente varianza) dei motivi nel multigrafo. Tuttavia, quando si cercano motivi di grandi dimensioni (ad esempio, numero di nodi  $> 6$ ), il calcolo della varianza richiede troppo tempo. In questi casi, è necessario utilizzare un'*approssimazione del modello*, che è in grado di stimare il p-value senza il calcolo di tale varianza.

In entrambi i casi, tale modello analitico restituisce delle curve precision-recall, utili a valutarne le performance, molto buone su diversi grafi reali, in confronto al modello che prevede la simulazione di grafi random.

### 6.5.3.2 Varianti del subgraph matching: ESM e MSM

Esistono diverse varianti del problema di subgraph matching, sulla base del tipo di comparazione da effettuare rispetto alle etichette degli archi.

Le due varianti principali considerate dall'algoritmo MultiMotif sono:

- **”Exact Subgraph Matching”**: tutti gli archi etichettati tra coppie di nodi etichettati in un motivo  $M$  devono essere presenti anche in qualche sottografo  $s$  del grafo target  $G$  con le stesse etichette, e viceversa.  
Per capire se un motivo è di interesse, si calcola la cosiddetta **”exact occurrence probability”**.

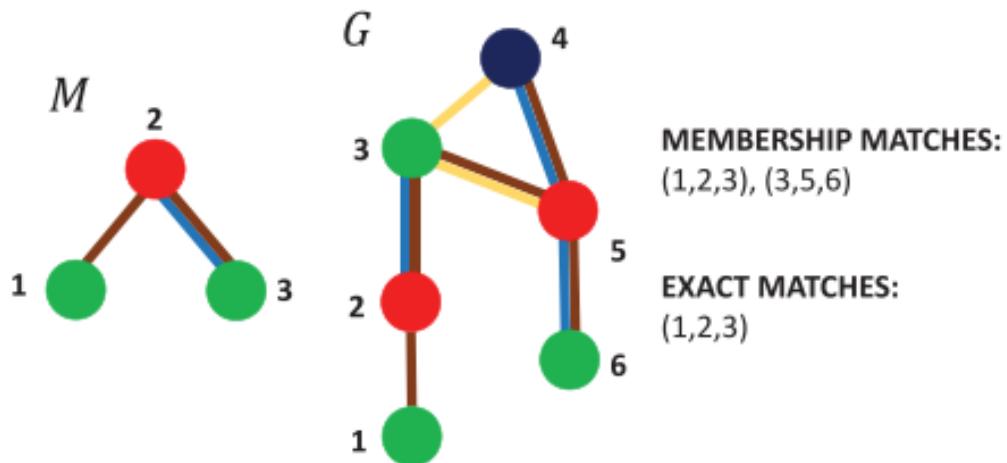
Formalmente, dati un grafo query  $M = (V, E, \Sigma_V, \Sigma_E, l_V, l_E)$  e un grafo target  $G = (V', E', \Sigma'_V, \Sigma'_E, l'_V, l'_E)$ , il problema dell’”Exact Subgraph Matching” (ESM) consiste nel trovare una **funzione di matching iniettiva**  $R : V \rightarrow V'$  che **mappa ogni nodo di  $M$  in un nodo di  $G$** , in maniera tale che  $\forall(u, v) \in E$ :

- $e' = (R(u), R(v)) \in E'$ ;
- $l_V(u) = l'_V(R(u))$ ;
- $l_V(v) = l'_V(R(v))$ ;
- $l_E(e) = l'_E(e)$ .

- **”Membership Subgraph Matching”**: tutti gli archi etichettati in un motivo  $M$  devono essere presenti anche in qualche sottografo  $s$  del grafo target  $G$  con le stesse etichette, ma non è necessario il contrario  
In questo caso, per capire se un motivo è di interesse, si calcola la cosiddetta **”membership occurrence probability”**.

Formalmente, dati un grafo query  $M = (V, E, \Sigma_V, \Sigma_E, l_V, l_E)$  e un grafo target  $G = (V', E', \Sigma'_V, \Sigma'_E, l'_V, l'_E)$ , il problema del ”Membership Subgraph Matching” (MSM) consiste nel trovare una **funzione di matching iniettiva**  $R : V \rightarrow V'$  che **mappa ogni nodo di  $M$  in un nodo di  $G$** , in maniera tale che  $\forall(u, v) \in E$ :

- $e' = (R(u), R(v)) \in E'$ ;
- $l_V(u) = l'_V(R(u))$ ;
- $l_V(v) = l'_V(R(v))$ ;
- $l_E(e) \subseteq l'_E(e)$ .



**Figura 6.6:** Esempio pratico della differenza tra ESM e MSM

#### 6.5.3.3 Probabilità di occorrenza esatta del motivo

Dato un motivo  $M = (V, E, \Sigma_V, \Sigma_E, l_V, l_E)$  con  $|V| = k$ , vogliamo calcolare la **probabilità di occorrenza** di  $M$  in un grafo **target**  $G = (V', E', \Sigma'_V, \Sigma'_E, l'_V, l'_E)$  secondo il modello EDD e il problema del *ESM*. Si può però dimostrare che la *"membership occurrence probability"* di  $M$  è **una funzione** delle *"exact occurrence probabilities"* di uno o più motivi.

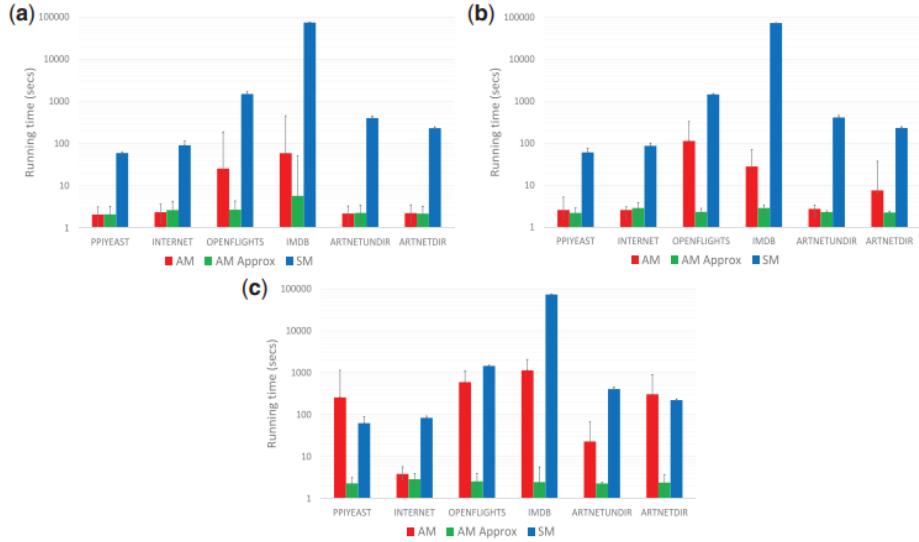
#### 6.5.3.4 Media e varianza del conteggio di motivi

Il passo successivo consiste nel **calcolo della media e della varianza del numero di occorrenze** di un motivo  $M = (V, E, \Sigma_V, \Sigma_E, l_V, l_E)$  con  $|V| = k$  in un multigrafo target random  $G = (V', E', \Sigma'_V, \Sigma'_E, l'_V, l'_E)$ , con  $|V'| = N$  e **molteplicità**  $\nabla$ , sempre secondo il modello EDD.

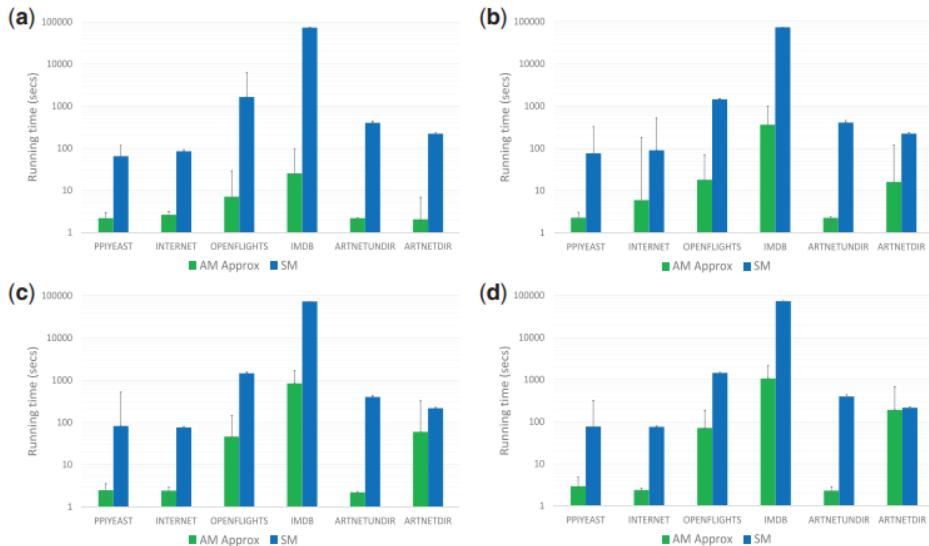
#### 6.5.3.5 Calcolo del p-value

Per concludere, al fine di stabilire se un motivo  $M$  è *"over-represented"* in un multigrafo  $G$ , si deve calcolare la probabilità  $P[N_{MSM}(M, R) \geq N_{MSM}(M, G)]$ , detta **"p-value"**, dove:

- $R$  è un grafo random generato con un modello scelto come riferimento;
- $N_{MSM}(M, G)$ : variabile aleatoria che rappresenta il numero di *"membership occurrences"* di un grafo query  $M$  in un grafo target  $G$ .



**Figura 6.7:** Tempo medio di esecuzione per motivo (e relativa deviazione standard) del ”pure analytical model” (AM), ”approximate analytical model” (AM Approx) e ”simulation-based model” (SM) per query random di (a) 4 nodi, (b) 5 nodi e (c) 6 nodi su grafi reali.



**Figura 6.8:** Tempo medio di esecuzione per motivo (e relativa deviazione standard) del ”approximate analytical model” (AM Approx) e ”simulation-based model” (SM) per query random di (a) 7 nodi, (b) 8 nodi, (c) 9 nodi e (d) 10 nodi su grafi reali.

### 6.5.4 Implementazione della ricerca di motivi con MultiMotif

L’implementazione dell’algoritmo MultiMotif all’interno dell’applicazione è avvenuta grazie al codice sorgente, scritto in Java, fornito direttamente dagli autori del suddetto algoritmo.

All’interno di FlightAnalyzer, l’utente ha la possibilità di creare le ”query” da dare in input all’algoritmo, ovvero i motivi da ricercare all’interno della rete aeroportuale. Oltre alla definizione dei nodi e degli archi che compongono il motivo, l’utente ha la possibilità di specificare le etichette dei singoli nodi e archi della query.

#### 6.5.4.1 Formato della rete da analizzare

L’algoritmo MultiMotif richiede che la rete da analizzare sia fornita in un apposito file *.txt* con la seguente struttura:

- la prima riga definisce l’**orientamento degli archi della rete**: ”*directed*” se gli archi sono orientati, ”*undirected*” altrimenti;
- la seconda riga contiene il **numero *N* di nodi della rete**. I nodi avranno indici che vanno da 0 a  $N - 1$ ;
- le successive *N* righe conterranno le ***etichette degli N nodi della rete***
- le successive righe contengono gli ***archi della rete e le relative etichette***. Per ogni riga/archo devono essere presenti ***tre valori separati da un carattere TAB***. Nello specifico:
  - il primo numero è l’***indice del nodo sorgente***;
  - il secondo numero è l’***indice del nodo destinazione***;
  - il terzo valore è la ***lista delle etichette dell’arco***, separate da virgole; l’ordine non è importante.

Per l’implementazione della ricerca di motivi all’interno di FlyghtAnalyzer, è stato necessario creare ***due diversi file di input***:

- il primo rappresenta la rete in cui le etichette dei nodi sono i nomi dei corrispondenti aeroporti;
- il secondo rappresenta la rete in cui le etichette dei nodi sono le nazioni in cui si trovano i rispettivi aeroporti.

#### 6.5.4.2 Formato delle query

Al fine di fornire in input i motivi da ricercare all'interno della rete, è necessario anche in questo caso creare un file **.txt** contenente le query da ricercare, in un apposito formato.

Tale file dovrà avere la seguente struttura:

- la prima riga contiene un **"nome identificativo"** univoco per la query, preceduto dal carattere "#";
- la seconda riga contiene il **numero  $N$  di nodi della query**. I nodi avranno indici che vanno da 0 a  $N - 1$ ;
- le successive  $N$  righe conterranno le **etichette degli  $N$  nodi della query**. In alternativa, è possibile assegnare al nodo un'**etichetta indefinita** tramite il carattere speciale "?";
- le successive righe contengono gli **archi della rete e le relative etichette**. Per ogni riga/arco devono essere presenti **tre valori separati da un carattere TAB**. In maniera del tutto analoga al file rappresentante la rete:
  - il primo numero è l'**indice del nodo sorgente**;
  - il secondo numero è l'**indice del nodo destinazione**;
  - il terzo valore è la **lista delle etichette dell'arco**, separate da virgole; l'ordine non è importante. In questo caso inoltre è possibile specificare delle "**etichette indefinite**" attraverso i caratteri:
    - \* "?", per indicare una singola etichetta indefinita
    - \* "\*", per indicare una o più etichette indefinite

È possibile definire più di una query all'interno dello stesso file.

**Codice 6.8:** Esempio di rete input per MultiMotif: rete degli aeroporti

```
directed
6072
Goroka Airport
Madang Airport
...
Desierto de Atacama Airport
0   1   C
0   2   C
...
6071     2014      C
```

**Codice 6.9:** Esempio di query per MultiMotif

```
#query1
3
Italy
Spain
?
0   1   ?
2   1   C
#query2
4
Italy
Germany
?
?
0   2   L
2   1   C
3   1   ?
2   3   L
```

#### 6.5.4.3 Creazione delle query

La creazione delle query in FlightAnalyzer avviene per mezzo di un'***interfaccia grafica interattiva***, realizzata grazie alla libreria **VisNetwork** di R.

La suddetta interfaccia grafica permette di creare passo passo le singole query, dando la possibilità all'utente, in maniera molto semplice e intuitiva, di:

- Aggiungere o rimuovere nodi o archi
- Aggiungere, rimuovere o modificare le etichette di nodi o archi
- Visualizzare la struttura del grafo query
- Assegnare un nome alla query ed inserirla nel file delle query che sarà successivamente usato come input da MultiMotif.

I dati riguardanti la query in corso di creazione vengono memorizzati e aggiornati in tempo reale all'interno di appositi dataframe fin quando l'utente continua a modificare la query attuale.

Una volta definite le diverse query, sarà l'applicazione stessa ad occuparsi di ***creare il file di input per MultiMotif nel formato corretto***, così come specificato in sez. 6.5.4.2

#### Codice 6.10: Creazione del file query

```

1 #parte UI
2
3 fluidRow(
4     column(style = "border: 4px double #222d32; padding: 5px;
5         margin:20px;",width = 2,
6         fluidRow(
7             radioButtons("labelType","Etichetta dei nodi: ",
8                 choices=c("Nome aeroporto","Nazione") ,
9                 selected="Nome aeroporto"),
10           textInput(inputId = "newQueryName",label =
11                 "Inserisci il nome della nuova query" , width =
12                 "70%"),
13            actionButton("addQuery","Aggiungi query", style=
14                 color: #fff; background-color: #337ab7; border
15                 -color: #2e6da4; align: center"),
16            actionButton("resetQuery","Reset query attuale",
17                 style="color: #fff; background-color: #337ab7;
18                 border-color: #2e6da4")
19        )
20    ),
21    column(width = 9,
22        visNetworkOutput("motifGraph"))
23

```

```

14 )
15 ),
16 fluidRow(
17   actionButton("deleteQuery", "Elimina TUTTE le query", style =
18     ="color: #fff; background-color: #337ab7; border-color:
19       : #2e6da4"),
20   actionButton("searchMotif", "Cerca motivi", style="color:
21     #fff; background-color: #337ab7; border-color: #2e6da4
22     ")
23 )
24
25 #parte Server: e' riportata solo la creazione del file delle
26 query
27
28 observeEvent(input$addQuery, {
29   # se il file delle query non esiste, lo si crea
30   if(!file.exists("query.txt"))
31     file.create("query.txt")
32
33   if(nrow(graph_data$nodes) == 0 | nrow(graph_data$edges) ==
34     0)
35     showNotification(paste("Devi inserire almeno un nodo e
36       un arco"), duration = 4 , type="error")
37
38   else if(newQueryName() == "" | any(grepl(paste0("#",
39     newQueryName()), readLines("query.txt"))))
40     showNotification(paste("Inserire un nome valido (no nome
41       vuoto o nomi duplicati)"), duration = 4 , type="error")
42
43   else{
44     write(paste0("#", newQueryName()), "query.txt", append =
45       TRUE)
46     write(nrow(graph_data$nodes) , "query.txt", append = TRUE
47       )
48
49     # scrittura del numero e delle etichette dei nodi
50
51     for(i in 1:nrow(graph_data$nodes)){
52       if(graph_data$nodes$label[i]==(""))
53         write( "?" , "query.txt",append = TRUE)
54       else
55         write( graph_data$nodes$label[i] , "query.txt",
56           append = TRUE)
57     }
58
59     # se non e' stata specificata un'etichetta per l'arco,
60       essa e' considerata come "?"
61
62     for(i in 1:nrow(graph_data$edges))
63
64   }
65
66   # se non e' stata specificata un'etichetta per l'arco,
67       essa e' considerata come "?"
68
69   for(i in 1:nrow(graph_data$edges))
70
71   }
72
73   # parte Client: si visualizza una lista di nodi e un
74   # menu per aggiungere un nuovo nodo
75   # si visualizza anche un menu per eliminare un nodo
76   # si visualizza anche un menu per cercare motivi
77
78   fluidRow(
79     column(4,
80       uiOutput("listNodes")
81     ),
82     column(4,
83       uiOutput("addNodeForm")
84     )
85   )
86
87   # si visualizza un menu per cercare motivi
88   # si visualizza un menu per eliminare un nodo
89
90   fluidRow(
91     column(4,
92       uiOutput("searchMotifForm")
93     ),
94     column(4,
95       uiOutput("deleteQueryForm")
96     )
97   )
98
99   # si visualizza un menu per eliminare un nodo
100
101 
```

```

49      if(is.na(graph_data$edges$label[i]) | graph_data$edges
50          $label[i] == ""))
51          graph_data$edges$label[i] <- "?"
52
53      # unifico le righe relative a multiarchi (stessa
54      # sorgente e destinazione)
55
56      multiedgeDF <- aggregate(graph_data$edges$label, by =
57          list(graph_data$edges$from, graph_data$edges$to),
58          FUN = function(x) {paste(x, collapse = ",")})
59      colnames(multiedgeDF) <- c("from", "to", "label")
60
61      #scrittura degli archi nel file query
62
63      for(i in 1:nrow(multiedgeDF))
64          write(paste(c(multiedgeDF$from[i], multiedgeDF$to[i]
65              ], multiedgeDF$label[i]), collapse = "\t"), "
66              query.txt", append = TRUE)
67
68  })
69 })
```

#### 6.5.4.4 Esecuzione di MultiMotif

L'esecuzione di MultiMotif avviene richiamando dalla Shell, con appositi parametri, il seguente comando:

```
java -jar MultiMotif.jar -n <networkFile> -q <queryFile>
    [-pa <pvalAnalThresh> -f <minFrequency> -or <
    resultsFile> -ap]
```

L'esecuzione di codice della Shell in R può essere effettuata attraverso la funzione *"system"*, la quale avrà come parametro proprio il comando da eseguire.

**Tabella 6.1:** Parametri dell'algoritmo MultiMotif

<b>Obbligatori</b>	-n	Nome del file .txt contenente la rete
	-q	Nome del file .txt contenente le query
<b>Non obbligatori</b>	-nopval	Conta i motivi senza calcolare il relativo p-value (default = calcola il p-value)
	-pa	Restituisce solo motivi che hanno un p-value al più pari a tale soglia (default = 1.0)
	-f	Restituisce solo motivi che hanno una frequenza almeno pari a tale soglia (default = 1)
	-or	Nome del file in cui sono salvati i risultati (default = "results.txt")
	-ap	Calcola un p-value "approssimato"

**Codice 6.11:** Esecuzione di MultiMotif

```

1 observeEvent(input$searchMotif, {
2   netFile <- NULL
3   if(labelType() == "Nome aeroporto")
4     netFile <- "airportsNetwork.txt"
5   else if(labelType() == "Nazione")
6     netFile <- "airportsNetworkCountry.txt"
7   if(!file.exists(netFile))
8     showNotification(paste("File della rete non trovato"),
9                     duration = 4, type="error")
10  else if(!file.exists("query.txt"))
11    showNotification(paste("File di query non trovato"),
12                     duration = 4, type="error")
13  else{
14    # chiamo "MultiMotif" sui file network (in base alla
15    # label scelta) e query creati
16    system(paste("java -jar MultiMotif\\MultiMotif.jar -n",
17                netFile, "-q query.txt"))
18  }
19 })

```

#### 6.5.4.5 Formato dei risultati

Una volta terminata l'esecuzione dell'algoritmo, i risultati vengono salvati all'interno di un file `"results.txt"` (o in alternativa avente il nome specificato insieme al parametro `-or`).

Tale file avrà una *struttura tabulare*, in cui in ogni riga della tabella saranno presenti specifici valori calcolati per una singola query, separati l'uno dall'altro da un carattere *TAB*.

Per ogni motivo sono presenti i seguenti campi:

- ***Motif\_nodes***: lista delle etichette dei nodi che compongono il motivo.
- ***Motif\_edges***: lista di archi nella forma  $(x, y, label_1, \dots, label_k)$  dove  $x$  e  $y$  sono rispettivamente i nodi sorgente e destinazione dell'arco, mentre  $label_1, \dots, label_k$  sono le etichette dell'arco;
- ***Num\_occ\_input\_graph***: numero di occorrenze del motivo nella rete fornita in input;
- ***Mean\_analytical***: numero medio di occorrenze del motivo nell'insieme dei grafi casuali EDD, secondo il modello analitico (vedere [9] per maggiori informazioni);
- ***Pval\_analytical***: p-value analitico del motivo.

#### 6.5.4.6 Visualizzazione dei motivi

Una volta calcolate le informazioni sui diversi motivi, esse saranno riportate in un'apposita tabella.

Inoltre, l'utente ha la possibilità di selezionare dalla tabella tutti o parte dei motivi trovati, al fine di visualizzarli graficamente su un globo 3D interattivo.

**Codice 6.12:** Visualizzazione della tabella dei risultati e del globo 3D

```

1 #parte UI
2 fluidRow(
3     div(h2("RISULTATI"), style = "text-align: center; "),
4     DTOutput ("motifSearchResults")
5 ),
6 fluidRow(
7     column(width=6,
8         globeOutput("motifGlobe" , width="700px" , height="700
9             px"))
9 ,

```

```

10     column(width=6,
11             uiOutput("legenda")
12         )
13     )
14
15 #parte Server
16
17 output$motifSearchResults <- renderDT({
18     read.delim('results.txt',sep='\t' , header = TRUE ,
19     colClasses = c("character", "character","character",
20     "character","character","character"))
21 })
22
23 observeEvent(input$motifSearchResults_rows_selected, {
24     if(labelType() == "Nome aeroporto"){
25
26         ...
27
28         output$motifGlobe <- renderGlobe({
29             create_globe()%>%
30             globe_arcs(
31                 coords(
32                     start_lat = yFrom,
33                     start_lon = xFrom,
34                     end_lat = yTo,
35                     end_lon = xTo,
36                     color = color,
37                     label = motif,
38                     altitude = arcAltitude,
39                     stroke = 0.7,
40                     dash_length = .2,
41                     dash_gap = .08,
42                     dash_animate_time = 6000
43                 ),
44                 data = cbind(globeArcs,legenda,arcAltitude)
45             )%>%
46             globe_bars(
47                 coords(lat,long, label = name,color=nodeColors),
48                 data = cbind(nodesDF,nodeColors)
49             )
50         })
51
52         output$legenda <- renderUI({
53             HTML('<h3>Legenda</h3>')
54             legendaRows <- apply(unique(legenda), 1, function(row
55                 ) {
56                 generateSquare(row[2], row[1])
57             })
58             HTML(paste(legendaRows, collapse = ""))
59         })
60     })
61
62     output$motifGlobe <- renderGlobe({
63         create_globe()%>%
64         globe_arcs(
65             coords(
66                 start_lat = yFrom,
67                 start_lon = xFrom,
68                 end_lat = yTo,
69                 end_lon = xTo,
70                 color = color,
71                 label = motif,
72                 altitude = arcAltitude,
73                 stroke = 0.7,
74                 dash_length = .2,
75                 dash_gap = .08,
76                 dash_animate_time = 6000
77             ),
78             data = cbind(globeArcs,legenda,arcAltitude)
79         )%>%
80         globe_bars(
81             coords(lat,long, label = name,color=nodeColors),
82             data = cbind(nodesDF,nodeColors)
83         )
84     })
85
86     output$motifGlobe <- renderGlobe({
87         create_globe()%>%
88         globe_arcs(
89             coords(
90                 start_lat = yFrom,
91                 start_lon = xFrom,
92                 end_lat = yTo,
93                 end_lon = xTo,
94                 color = color,
95                 label = motif,
96                 altitude = arcAltitude,
97                 stroke = 0.7,
98                 dash_length = .2,
99                 dash_gap = .08,
100                dash_animate_time = 6000
101            ),
102            data = cbind(globeArcs,legenda,arcAltitude)
103        )%>%
104        globe_bars(
105            coords(lat,long, label = name,color=nodeColors),
106            data = cbind(nodesDF,nodeColors)
107        )
108    })
109
110     output$motifGlobe <- renderGlobe({
111         create_globe()%>%
112         globe_arcs(
113             coords(
114                 start_lat = yFrom,
115                 start_lon = xFrom,
116                 end_lat = yTo,
117                 end_lon = xTo,
118                 color = color,
119                 label = motif,
120                 altitude = arcAltitude,
121                 stroke = 0.7,
122                 dash_length = .2,
123                 dash_gap = .08,
124                 dash_animate_time = 6000
125             ),
126             data = cbind(globeArcs,legenda,arcAltitude)
127         )%>%
128         globe_bars(
129             coords(lat,long, label = name,color=nodeColors),
130             data = cbind(nodesDF,nodeColors)
131         )
132     })
133
134     output$motifGlobe <- renderGlobe({
135         create_globe()%>%
136         globe_arcs(
137             coords(
138                 start_lat = yFrom,
139                 start_lon = xFrom,
140                 end_lat = yTo,
141                 end_lon = xTo,
142                 color = color,
143                 label = motif,
144                 altitude = arcAltitude,
145                 stroke = 0.7,
146                 dash_length = .2,
147                 dash_gap = .08,
148                 dash_animate_time = 6000
149             ),
150             data = cbind(globeArcs,legenda,arcAltitude)
151         )%>%
152         globe_bars(
153             coords(lat,long, label = name,color=nodeColors),
154             data = cbind(nodesDF,nodeColors)
155         )
156     })
157
158     output$motifGlobe <- renderGlobe({
159         create_globe()%>%
160         globe_arcs(
161             coords(
162                 start_lat = yFrom,
163                 start_lon = xFrom,
164                 end_lat = yTo,
165                 end_lon = xTo,
166                 color = color,
167                 label = motif,
168                 altitude = arcAltitude,
169                 stroke = 0.7,
170                 dash_length = .2,
171                 dash_gap = .08,
172                 dash_animate_time = 6000
173             ),
174             data = cbind(globeArcs,legenda,arcAltitude)
175         )%>%
176         globe_bars(
177             coords(lat,long, label = name,color=nodeColors),
178             data = cbind(nodesDF,nodeColors)
179         )
180     })
181
182     output$motifGlobe <- renderGlobe({
183         create_globe()%>%
184         globe_arcs(
185             coords(
186                 start_lat = yFrom,
187                 start_lon = xFrom,
188                 end_lat = yTo,
189                 end_lon = xTo,
190                 color = color,
191                 label = motif,
192                 altitude = arcAltitude,
193                 stroke = 0.7,
194                 dash_length = .2,
195                 dash_gap = .08,
196                 dash_animate_time = 6000
197             ),
198             data = cbind(globeArcs,legenda,arcAltitude)
199         )%>%
200         globe_bars(
201             coords(lat,long, label = name,color=nodeColors),
202             data = cbind(nodesDF,nodeColors)
203         )
204     })
205
206     output$motifGlobe <- renderGlobe({
207         create_globe()%>%
208         globe_arcs(
209             coords(
210                 start_lat = yFrom,
211                 start_lon = xFrom,
212                 end_lat = yTo,
213                 end_lon = xTo,
214                 color = color,
215                 label = motif,
216                 altitude = arcAltitude,
217                 stroke = 0.7,
218                 dash_length = .2,
219                 dash_gap = .08,
220                 dash_animate_time = 6000
221             ),
222             data = cbind(globeArcs,legenda,arcAltitude)
223         )%>%
224         globe_bars(
225             coords(lat,long, label = name,color=nodeColors),
226             data = cbind(nodesDF,nodeColors)
227         )
228     })
229
230     output$motifGlobe <- renderGlobe({
231         create_globe()%>%
232         globe_arcs(
233             coords(
234                 start_lat = yFrom,
235                 start_lon = xFrom,
236                 end_lat = yTo,
237                 end_lon = xTo,
238                 color = color,
239                 label = motif,
240                 altitude = arcAltitude,
241                 stroke = 0.7,
242                 dash_length = .2,
243                 dash_gap = .08,
244                 dash_animate_time = 6000
245             ),
246             data = cbind(globeArcs,legenda,arcAltitude)
247         )%>%
248         globe_bars(
249             coords(lat,long, label = name,color=nodeColors),
250             data = cbind(nodesDF,nodeColors)
251         )
252     })
253
254     output$motifGlobe <- renderGlobe({
255         create_globe()%>%
256         globe_arcs(
257             coords(
258                 start_lat = yFrom,
259                 start_lon = xFrom,
260                 end_lat = yTo,
261                 end_lon = xTo,
262                 color = color,
263                 label = motif,
264                 altitude = arcAltitude,
265                 stroke = 0.7,
266                 dash_length = .2,
267                 dash_gap = .08,
268                 dash_animate_time = 6000
269             ),
270             data = cbind(globeArcs,legenda,arcAltitude)
271         )%>%
272         globe_bars(
273             coords(lat,long, label = name,color=nodeColors),
274             data = cbind(nodesDF,nodeColors)
275         )
276     })
277
278     output$motifGlobe <- renderGlobe({
279         create_globe()%>%
280         globe_arcs(
281             coords(
282                 start_lat = yFrom,
283                 start_lon = xFrom,
284                 end_lat = yTo,
285                 end_lon = xTo,
286                 color = color,
287                 label = motif,
288                 altitude = arcAltitude,
289                 stroke = 0.7,
290                 dash_length = .2,
291                 dash_gap = .08,
292                 dash_animate_time = 6000
293             ),
294             data = cbind(globeArcs,legenda,arcAltitude)
295         )%>%
296         globe_bars(
297             coords(lat,long, label = name,color=nodeColors),
298             data = cbind(nodesDF,nodeColors)
299         )
300     })
301
302     output$motifGlobe <- renderGlobe({
303         create_globe()%>%
304         globe_arcs(
305             coords(
306                 start_lat = yFrom,
307                 start_lon = xFrom,
308                 end_lat = yTo,
309                 end_lon = xTo,
310                 color = color,
311                 label = motif,
312                 altitude = arcAltitude,
313                 stroke = 0.7,
314                 dash_length = .2,
315                 dash_gap = .08,
316                 dash_animate_time = 6000
317             ),
318             data = cbind(globeArcs,legenda,arcAltitude)
319         )%>%
320         globe_bars(
321             coords(lat,long, label = name,color=nodeColors),
322             data = cbind(nodesDF,nodeColors)
323         )
324     })
325
326     output$motifGlobe <- renderGlobe({
327         create_globe()%>%
328         globe_arcs(
329             coords(
330                 start_lat = yFrom,
331                 start_lon = xFrom,
332                 end_lat = yTo,
333                 end_lon = xTo,
334                 color = color,
335                 label = motif,
336                 altitude = arcAltitude,
337                 stroke = 0.7,
338                 dash_length = .2,
339                 dash_gap = .08,
340                 dash_animate_time = 6000
341             ),
342             data = cbind(globeArcs,legenda,arcAltitude)
343         )%>%
344         globe_bars(
345             coords(lat,long, label = name,color=nodeColors),
346             data = cbind(nodesDF,nodeColors)
347         )
348     })
349
350     output$motifGlobe <- renderGlobe({
351         create_globe()%>%
352         globe_arcs(
353             coords(
354                 start_lat = yFrom,
355                 start_lon = xFrom,
356                 end_lat = yTo,
357                 end_lon = xTo,
358                 color = color,
359                 label = motif,
360                 altitude = arcAltitude,
361                 stroke = 0.7,
362                 dash_length = .2,
363                 dash_gap = .08,
364                 dash_animate_time = 6000
365             ),
366             data = cbind(globeArcs,legenda,arcAltitude)
367         )%>%
368         globe_bars(
369             coords(lat,long, label = name,color=nodeColors),
370             data = cbind(nodesDF,nodeColors)
371         )
372     })
373
374     output$motifGlobe <- renderGlobe({
375         create_globe()%>%
376         globe_arcs(
377             coords(
378                 start_lat = yFrom,
379                 start_lon = xFrom,
380                 end_lat = yTo,
381                 end_lon = xTo,
382                 color = color,
383                 label = motif,
384                 altitude = arcAltitude,
385                 stroke = 0.7,
386                 dash_length = .2,
387                 dash_gap = .08,
388                 dash_animate_time = 6000
389             ),
390             data = cbind(globeArcs,legenda,arcAltitude)
391         )%>%
392         globe_bars(
393             coords(lat,long, label = name,color=nodeColors),
394             data = cbind(nodesDF,nodeColors)
395         )
396     })
397
398     output$motifGlobe <- renderGlobe({
399         create_globe()%>%
400         globe_arcs(
401             coords(
402                 start_lat = yFrom,
403                 start_lon = xFrom,
404                 end_lat = yTo,
405                 end_lon = xTo,
406                 color = color,
407                 label = motif,
408                 altitude = arcAltitude,
409                 stroke = 0.7,
410                 dash_length = .2,
411                 dash_gap = .08,
412                 dash_animate_time = 6000
413             ),
414             data = cbind(globeArcs,legenda,arcAltitude)
415         )%>%
416         globe_bars(
417             coords(lat,long, label = name,color=nodeColors),
418             data = cbind(nodesDF,nodeColors)
419         )
420     })
421
422     output$motifGlobe <- renderGlobe({
423         create_globe()%>%
424         globe_arcs(
425             coords(
426                 start_lat = yFrom,
427                 start_lon = xFrom,
428                 end_lat = yTo,
429                 end_lon = xTo,
430                 color = color,
431                 label = motif,
432                 altitude = arcAltitude,
433                 stroke = 0.7,
434                 dash_length = .2,
435                 dash_gap = .08,
436                 dash_animate_time = 6000
437             ),
438             data = cbind(globeArcs,legenda,arcAltitude)
439         )%>%
440         globe_bars(
441             coords(lat,long, label = name,color=nodeColors),
442             data = cbind(nodesDF,nodeColors)
443         )
444     })
445
446     output$motifGlobe <- renderGlobe({
447         create_globe()%>%
448         globe_arcs(
449             coords(
450                 start_lat = yFrom,
451                 start_lon = xFrom,
452                 end_lat = yTo,
453                 end_lon = xTo,
454                 color = color,
455                 label = motif,
456                 altitude = arcAltitude,
457                 stroke = 0.7,
458                 dash_length = .2,
459                 dash_gap = .08,
460                 dash_animate_time = 6000
461             ),
462             data = cbind(globeArcs,legenda,arcAltitude)
463         )%>%
464         globe_bars(
465             coords(lat,long, label = name,color=nodeColors),
466             data = cbind(nodesDF,nodeColors)
467         )
468     })
469
470     output$motifGlobe <- renderGlobe({
471         create_globe()%>%
472         globe_arcs(
473             coords(
474                 start_lat = yFrom,
475                 start_lon = xFrom,
476                 end_lat = yTo,
477                 end_lon = xTo,
478                 color = color,
479                 label = motif,
480                 altitude = arcAltitude,
481                 stroke = 0.7,
482                 dash_length = .2,
483                 dash_gap = .08,
484                 dash_animate_time = 6000
485             ),
486             data = cbind(globeArcs,legenda,arcAltitude)
487         )%>%
488         globe_bars(
489             coords(lat,long, label = name,color=nodeColors),
490             data = cbind(nodesDF,nodeColors)
491         )
492     })
493
494     output$motifGlobe <- renderGlobe({
495         create_globe()%>%
496         globe_arcs(
497             coords(
498                 start_lat = yFrom,
499                 start_lon = xFrom,
500                 end_lat = yTo,
501                 end_lon = xTo,
502                 color = color,
503                 label = motif,
504                 altitude = arcAltitude,
505                 stroke = 0.7,
506                 dash_length = .2,
507                 dash_gap = .08,
508                 dash_animate_time = 6000
509             ),
510             data = cbind(globeArcs,legenda,arcAltitude)
511         )%>%
512         globe_bars(
513             coords(lat,long, label = name,color=nodeColors),
514             data = cbind(nodesDF,nodeColors)
515         )
516     })
517
518     output$motifGlobe <- renderGlobe({
519         create_globe()%>%
520         globe_arcs(
521             coords(
522                 start_lat = yFrom,
523                 start_lon = xFrom,
524                 end_lat = yTo,
525                 end_lon = xTo,
526                 color = color,
527                 label = motif,
528                 altitude = arcAltitude,
529                 stroke = 0.7,
530                 dash_length = .2,
531                 dash_gap = .08,
532                 dash_animate_time = 6000
533             ),
534             data = cbind(globeArcs,legenda,arcAltitude)
535         )%>%
536         globe_bars(
537             coords(lat,long, label = name,color=nodeColors),
538             data = cbind(nodesDF,nodeColors)
539         )
540     })
541
542     output$motifGlobe <- renderGlobe({
543         create_globe()%>%
544         globe_arcs(
545             coords(
546                 start_lat = yFrom,
547                 start_lon = xFrom,
548                 end_lat = yTo,
549                 end_lon = xTo,
550                 color = color,
551                 label = motif,
552                 altitude = arcAltitude,
553                 stroke = 0.7,
554                 dash_length = .2,
555                 dash_gap = .08,
556                 dash_animate_time = 6000
557             ),
558             data = cbind(globeArcs,legenda,arcAltitude)
559         )%>%
560         globe_bars(
561             coords(lat,long, label = name,color=nodeColors),
562             data = cbind(nodesDF,nodeColors)
563         )
564     })
565
566     output$motifGlobe <- renderGlobe({
567         create_globe()%>%
568         globe_arcs(
569             coords(
570                 start_lat = yFrom,
571                 start_lon = xFrom,
572                 end_lat = yTo,
573                 end_lon = xTo,
574                 color = color,
575                 label = motif,
576                 altitude = arcAltitude,
577                 stroke = 0.7,
578                 dash_length = .2,
579                 dash_gap = .08,
580                 dash_animate_time = 6000
581             ),
582             data = cbind(globeArcs,legenda,arcAltitude)
583         )%>%
584         globe_bars(
585             coords(lat,long, label = name,color=nodeColors),
586             data = cbind(nodesDF,nodeColors)
587         )
588     })
589
590     output$motifGlobe <- renderGlobe({
591         create_globe()%>%
592         globe_arcs(
593             coords(
594                 start_lat = yFrom,
595                 start_lon = xFrom,
596                 end_lat = yTo,
597                 end_lon = xTo,
598                 color = color,
599                 label = motif,
600                 altitude = arcAltitude,
601                 stroke = 0.7,
602                 dash_length = .2,
603                 dash_gap = .08,
604                 dash_animate_time = 6000
605             ),
606             data = cbind(globeArcs,legenda,arcAltitude)
607         )%>%
608         globe_bars(
609             coords(lat,long, label = name,color=nodeColors),
610             data = cbind(nodesDF,nodeColors)
611         )
612     })
613
614     output$motifGlobe <- renderGlobe({
615         create_globe()%>%
616         globe_arcs(
617             coords(
618                 start_lat = yFrom,
619                 start_lon = xFrom,
620                 end_lat = yTo,
621                 end_lon = xTo,
622                 color = color,
623                 label = motif,
624                 altitude = arcAltitude,
625                 stroke = 0.7,
626                 dash_length = .2,
627                 dash_gap = .08,
628                 dash_animate_time = 6000
629             ),
630             data = cbind(globeArcs,legenda,arcAltitude)
631         )%>%
632         globe_bars(
633             coords(lat,long, label = name,color=nodeColors),
634             data = cbind(nodesDF,nodeColors)
635         )
636     })
637
638     output$motifGlobe <- renderGlobe({
639         create_globe()%>%
640         globe_arcs(
641             coords(
642                 start_lat = yFrom,
643                 start_lon = xFrom,
644                 end_lat = yTo,
645                 end_lon = xTo,
646                 color = color,
647                 label = motif,
648                 altitude = arcAltitude,
649                 stroke = 0.7,
650                 dash_length = .2,
651                 dash_gap = .08,
652                 dash_animate_time = 6000
653             ),
654             data = cbind(globeArcs,legenda,arcAltitude)
655         )%>%
656         globe_bars(
657             coords(lat,long, label = name,color=nodeColors),
658             data = cbind(nodesDF,nodeColors)
659         )
660     })
661
662     output$motifGlobe <- renderGlobe({
663         create_globe()%>%
664         globe_arcs(
665             coords(
666                 start_lat = yFrom,
667                 start_lon = xFrom,
668                 end_lat = yTo,
669                 end_lon = xTo,
670                 color = color,
671                 label = motif,
672                 altitude = arcAltitude,
673                 stroke = 0.7,
674                 dash_length = .2,
675                 dash_gap = .08,
676                 dash_animate_time = 6000
677             ),
678             data = cbind(globeArcs,legenda,arcAltitude)
679         )%>%
680         globe_bars(
681             coords(lat,long, label = name,color=nodeColors),
682             data = cbind(nodesDF,nodeColors)
683         )
684     })
685
686     output$motifGlobe <- renderGlobe({
687         create_globe()%>%
688         globe_arcs(
689             coords(
690                 start_lat = yFrom,
691                 start_lon = xFrom,
692                 end_lat = yTo,
693                 end_lon = xTo,
694                 color = color,
695                 label = motif,
696                 altitude = arcAltitude,
697                 stroke = 0.7,
698                 dash_length = .2,
699                 dash_gap = .08,
700                 dash_animate_time = 6000
701             ),
702             data = cbind(globeArcs,legenda,arcAltitude)
703         )%>%
704         globe_bars(
705             coords(lat,long, label = name,color=nodeColors),
706             data = cbind(nodesDF,nodeColors)
707         )
708     })
709
710     output$motifGlobe <- renderGlobe({
711         create_globe()%>%
712         globe_arcs(
713             coords(
714                 start_lat = yFrom,
715                 start_lon = xFrom,
716                 end_lat = yTo,
717                 end_lon = xTo,
718                 color = color,
719                 label = motif,
720                 altitude = arcAltitude,
721                 stroke = 0.7,
722                 dash_length = .2,
723                 dash_gap = .08,
724                 dash_animate_time = 6000
725             ),
726             data = cbind(globeArcs,legenda,arcAltitude)
727         )%>%
728         globe_bars(
729             coords(lat,long, label = name,color=nodeColors),
730             data = cbind(nodesDF,nodeColors)
731         )
732     })
733
734     output$motifGlobe <- renderGlobe({
735         create_globe()%>%
736         globe_arcs(
737             coords(
738                 start_lat = yFrom,
739                 start_lon = xFrom,
740                 end_lat = yTo,
741                 end_lon = xTo,
742                 color = color,
743                 label = motif,
744                 altitude = arcAltitude,
745                 stroke = 0.7,
746                 dash_length = .2,
747                 dash_gap = .08,
748                 dash_animate_time = 6000
749             ),
750             data = cbind(globeArcs,legenda,arcAltitude)
751         )%>%
752         globe_bars(
753             coords(lat,long, label = name,color=nodeColors),
754             data = cbind(nodesDF,nodeColors)
755         )
756     })
757
758     output$motifGlobe <- renderGlobe({
759         create_globe()%>%
760         globe_arcs(
761             coords(
762                 start_lat = yFrom,
763                 start_lon = xFrom,
764                 end_lat = yTo,
765                 end_lon = xTo,
766                 color = color,
767                 label = motif,
768                 altitude = arcAltitude,
769                 stroke = 0.7,
770                 dash_length = .2,
771                 dash_gap = .08,
772                 dash_animate_time = 6000
773             ),
774             data = cbind(globeArcs,legenda,arcAltitude)
775         )%>%
776         globe_bars(
777             coords(lat,long, label = name,color=nodeColors),
778             data = cbind(nodesDF,nodeColors)
779         )
780     })
781
782     output$motifGlobe <- renderGlobe({
783         create_globe()%>%
784         globe_arcs(
785             coords(
786                 start_lat = yFrom,
787                 start_lon = xFrom,
788                 end_lat = yTo,
789                 end_lon = xTo,
790                 color = color,
791                 label = motif,
792                 altitude = arcAltitude,
793                 stroke = 0.7,
794                 dash_length = .2,
795                 dash_gap = .08,
796                 dash_animate_time = 6000
797             ),
798             data = cbind(globeArcs,legenda,arcAltitude)
799         )%>%
800         globe_bars(
801             coords(lat,long, label = name,color=nodeColors),
802             data = cbind(nodesDF,nodeColors)
803         )
804     })
805
806     output$motifGlobe <- renderGlobe({
807         create_globe()%>%
808         globe_arcs(
809             coords(
810                 start_lat = yFrom,
811                 start_lon = xFrom,
812                 end_lat = yTo,
813                 end_lon = xTo,
814                 color = color,
815                 label = motif,
816                 altitude = arcAltitude,
817                 stroke = 0.7,
818                 dash_length = .2,
819                 dash_gap = .08,
820                 dash_animate_time = 6000
821             ),
822             data = cbind(globeArcs,legenda,arcAltitude)
823         )%>%
824         globe_bars(
825             coords(lat,long, label = name,color=nodeColors),
826             data = cbind(nodesDF,nodeColors)
827         )
828     })
829
830     output$motifGlobe <- renderGlobe({
831         create_globe()%>%
832         globe_arcs(
833             coords(
834                 start_lat = yFrom,
835                 start_lon = xFrom,
836                 end_lat = yTo,
837                 end_lon = xTo,
838                 color = color,
839                 label = motif,
840                 altitude = arcAltitude,
841                 stroke = 0.7,
842                 dash_length = .2,
843                 dash_gap = .08,
844                 dash_animate_time = 6000
845             ),
846             data = cbind(globeArcs,legenda,arcAltitude)
847         )%>%
848         globe_bars(
849             coords(lat,long, label = name,color=nodeColors),
850             data = cbind(nodesDF,nodeColors)
851         )
852     })
853
854     output$motifGlobe <- renderGlobe({
855         create_globe()%>%
856         globe_arcs(
857             coords(
858                 start_lat = yFrom,
859                 start_lon = xFrom,
860                 end_lat = yTo,
861                 end_lon = xTo,
862                 color = color,
863                 label = motif,
864                 altitude = arcAltitude,
865                 stroke = 0.7,
866                 dash_length = .2,
867                 dash_gap = .08,
868                 dash_animate_time = 6000
869             ),
870             data = cbind(globeArcs,legenda,arcAltitude)
871         )%>%
872         globe_bars(
873             coords(lat,long, label = name,color=nodeColors),
874             data = cbind(nodesDF,nodeColors)
875         )
876     })
877
878     output$motifGlobe <- renderGlobe({
879         create_globe()%>%
880         globe_arcs(
881             coords(
882                 start_lat = yFrom,
883                 start_lon = xFrom,
884                 end_lat = yTo,
885                 end_lon = xTo,
886                 color = color,
887                 label = motif,
888                 altitude = arcAltitude,
889                 stroke = 0.7,
890                 dash_length = .2,
891                 dash_gap = .08,
892                 dash_animate_time = 6000
893             ),
894             data = cbind(globeArcs,legenda,arcAltitude)
895         )%>%
896         globe_bars(
897             coords(lat,long, label = name,color=nodeColors),
898             data = cbind(nodesDF,nodeColors)
899         )
900     })
901
902     output$motifGlobe <- renderGlobe({
903         create_globe()%>%
904         globe_arcs(
905             coords(
906                 start_lat = yFrom,
907                 start_lon = xFrom,
908                 end_lat = yTo,
909                 end_lon = xTo,
910                 color = color,
911                 label = motif,
912                 altitude = arcAltitude,
913                 stroke = 0.7,
914                 dash_length = .2,
915                 dash_gap = .08,
916                 dash_animate_time = 6000
917             ),
918             data = cbind(globeArcs,legenda,arcAltitude)
919         )%>%
920         globe_bars(
921             coords(lat,long, label = name,color=nodeColors),
922             data = cbind(nodesDF,nodeColors)
923         )
924     })
925
926     output$motifGlobe <- renderGlobe({
927         create_globe()%>%
928         globe_arcs(
929             coords(
930                 start_lat = yFrom,
931                 start_lon = xFrom,
932                 end_lat = yTo,
933                 end_lon = xTo,
934                 color = color,
935                 label = motif,
936                 altitude = arcAltitude,
937                 stroke = 0.7,
938                 dash_length = .2,
939                 dash_gap = .08,
940                 dash_animate_time = 6000
941             ),
942             data = cbind(globeArcs,legenda,arcAltitude)
943         )%>%
944         globe_bars(
945             coords(lat,long, label = name,color=nodeColors),
946             data = cbind(nodesDF,nodeColors)
947         )
948     })
949
950     output$motifGlobe <- renderGlobe({
951         create_globe()%>%
952         globe_arcs(
953             coords(
954                 start_lat = yFrom,
955                 start_lon = xFrom,
956                 end_lat = yTo,
957                 end_lon = xTo,
958                 color = color,
959                 label = motif,
960                 altitude = arcAltitude,
961                 stroke = 0.7,
962                 dash_length = .2,
963                 dash_gap = .08,
964                 dash_animate_time = 6000
965             ),
966             data = cbind(globeArcs,legenda,arcAltitude)
967         )%>%
968         globe_bars(
969             coords(lat,long, label = name,color=nodeColors),
970             data = cbind(nodesDF,nodeColors)
971         )
972     })
973
974     output$motifGlobe <- renderGlobe({
975         create_globe()%>%
976         globe_arcs(
977             coords(
978                 start_lat = yFrom,
979                 start_lon = xFrom,
980                 end_lat = yTo,
981                 end_lon = xTo,
982                 color = color,
983                 label = motif,
984                 altitude = arcAltitude,
985                 stroke = 0.7,
986                 dash_length = .2,
987                 dash_gap = .08,
988                 dash_animate_time = 6000
989             ),
990             data = cbind(globeArcs,legenda,arcAltitude)
991         )%>%
992         globe_bars(
993             coords(lat,long, label = name,color=nodeColors),
994             data = cbind(nodesDF,nodeColors)
995         )
996     })
997
998     output$motifGlobe <- renderGlobe({
999         create_globe()%>%
1000         globe_arcs(
1001             coords(
1002                 start_lat = yFrom,
1003                 start_lon = xFrom,
1004                 end_lat = yTo,
1005                 end_lon = xTo,
1006                 color = color,
1007                 label = motif,
1008                 altitude = arcAltitude,
1009                 stroke = 0.7,
1010                 dash_length = .2,
1011                 dash_gap = .08,
1012                 dash_animate_time = 6000
1013             ),
1014             data = cbind(globeArcs,legenda,arcAltitude)
1015         )%>%
1016         globe_bars(
1017             coords(lat,long, label = name,color=nodeColors),
1018             data = cbind(nodesDF,nodeColors)
1019         )
1020     })
1021
1022     output$motifGlobe <- renderGlobe({
1023         create_globe()%>%
1024         globe_arcs(
1025             coords(
1026                 start_lat = yFrom,
1027                 start_lon = xFrom,
1028                 end_lat = yTo,
1029                 end_lon = xTo,
1030                 color = color,
1031                 label = motif,
1032                 altitude = arcAltitude,
1033                 stroke = 0.7,
1034                 dash_length = .2,
1035                 dash_gap = .08,
1036                 dash_animate_time = 6000
1037             ),
1038             data = cbind(globeArcs,legenda,arcAltitude)
1039         )%>%
1040         globe_bars(
1041             coords(lat,long, label = name,color=nodeColors),
1042             data = cbind(nodesDF,nodeColors)
1043         )
1044     })
1045
1046     output$motifGlobe <- renderGlobe({
1047         create_globe()%>%
1048         globe_arcs(
1049             coords(
1050                 start_lat = yFrom,
1051                 start_lon = xFrom,
1052                 end_lat = yTo,
1053                 end_lon = xTo,
1054                 color = color,
1055                 label = motif,
1056                 altitude = arcAltitude,
1057                 stroke = 0.7,
1058                 dash_length = .2,
1059                 dash_gap = .08,
1060                 dash_animate_time = 6000
1061             ),
1062             data = cbind(globeArcs,legenda,arcAltitude)
1063         )%>%
1064         globe_bars(
1065             coords(lat,long, label = name,color=nodeColors),
1066             data = cbind(nodesDF,nodeColors)
1067         )
1068     })
1069
1070     output$motifGlobe <- renderGlobe({
1071         create_globe()%>%
1072         globe_arcs(
1073             coords(
1074                 start_lat = yFrom,
1075                 start_lon = xFrom,
1076                 end_lat = yTo,
1077                 end_lon = xTo,
1078                 color = color,
1079                 label = motif,
1080                 altitude = arcAltitude,
1081                 stroke = 0.7,
1082                 dash_length = .2,
1083                 dash_gap = .08,
1084                 dash_animate_time = 6000
1085             ),
1086             data = cbind(globeArcs,legenda,arcAltitude)
1087         )%>%
1088         globe_bars(
1089             coords(lat,long, label = name,color=nodeColors),
1090             data = cbind(nodesDF,nodeColors)
1091         )
1092     })
1093
1094     output$motifGlobe <- renderGlobe({
1095         create_globe()%>%
1096         globe_arcs(
1097             coords(
1098                 start_lat = yFrom,
1099                 start_lon = xFrom,
1100                 end_lat = yTo,
1101                 end_lon = xTo,
1102                 color = color,
1103                 label = motif,
1104                 altitude = arcAltitude,
1105                 stroke = 0.7,
1106                 dash_length = .2,
1107                 dash_gap = .08,
1108                 dash_animate_time = 6000
1109             ),
1110             data = cbind(globeArcs,legenda,arcAltitude)
1111         )%>%
1112         globe_bars(
1113             coords(lat,long, label = name,color=nodeColors),
1114             data = cbind(nodesDF,nodeColors)
1115         )
1116     })
1117
1118     output$motifGlobe <- renderGlobe({
1119         create_globe()%>%
1120         globe_arcs(
1121             coords(
1122                 start_lat = yFrom,
1123                 start_lon = xFrom,
1124                 end_lat = yTo,
1125                 end_lon = xTo,
1126                 color = color,
1127                 label = motif,
1128                 altitude = arcAltitude,
1129                 stroke = 0.7,
1130                 dash_length = .2,
1131                 dash_gap = .08,
1132                 dash_animate_time = 6000
1133             ),
1134             data = cbind(globeArcs,legenda,arcAltitude)
1135         )%>%
1136         globe_bars(
1137             coords(lat,long, label = name,color=nodeColors),
1138             data = cbind(nodesDF,nodeColors)
1139         )
1140     })
1141
1142     output$motifGlobe <- renderGlobe({
1143         create_globe()%>%
1144         globe_arcs(
1145             coords(
1146                 start_lat = yFrom,
1147                 start_lon = xFrom,
1148                 end_lat = yTo,
1149                 end_lon = xTo,
1150                 color = color,
1151                 label = motif,
1152                 altitude = arcAltitude,
1153                 stroke = 0.7,
1154                 dash_length = .2,
1155                 dash_gap = .08,
1156                 dash_animate_time = 6000
1157             ),
1158             data = cbind(globeArcs,legenda,arcAltitude)
1159         )%>%
1160         globe_bars(
1161             coords(lat,long, label = name,color=nodeColors),
1162             data = cbind(nodesDF,nodeColors)
1163         )
1164     })
1165
1166     output$motifGlobe <- renderGlobe({
1167         create_globe()%>%
1168         globe_arcs(
1169             coords(
1170                 start_lat = yFrom,
1171                 start_lon = xFrom,
1172                 end_lat = yTo,
1173                 end_lon = xTo,
1174                 color = color,
1175                 label = motif,
1176                 altitude = arcAltitude,
1177                 stroke = 0.7,
1178                 dash_length = .2,
1179                 dash_gap = .08,
1180                 dash_animate_time = 6000
1181             ),
1182             data = cbind(globeArcs,legenda,arcAltitude)
1183         )%>%
1184         globe_bars(
1185             coords(lat,long, label = name,color=nodeColors),
1186             data = cbind(nodesDF,nodeColors)
1187         )
1188     })
1189
1190     output$motifGlobe <- renderGlobe({
1191         create_globe()%>%
1192         globe_arcs(
1193             coords(
1194                 start_lat = yFrom,
1195                 start_lon = xFrom,
1196                 end_lat = yTo,
1197                 end_lon = xTo,
1198                 color = color,
1199                 label = motif,
1200                 altitude = arcAltitude,
1201                 stroke = 0.7,
1202                 dash_length = .2,
1203                 dash_gap = .08,
1204                 dash_animate_time = 6000
1205             ),
1206             data = cbind(globeArcs,legenda,arcAltitude)
1207         )%>%
1208         globe_bars(
1209             coords(lat,long, label = name,color=nodeColors),
1210             data = cbind(nodesDF,nodeColors)
1211         )
1212     })
1213
1214     output$motifGlobe <- renderGlobe({
1215         create_globe()%>%
1216         globe_arcs(
1217             coords(
1218                 start_lat = yFrom,
1219                 start_lon = xFrom,
1220                 end_lat = yTo,
1221                 end_lon = xTo,
1222                 color = color,
1223                 label = motif,
1224                 altitude = arcAltitude,
1225                 stroke = 0.7,
1226                 dash_length = .2,
1227                 dash_gap = .08
```

```

56         })
57     }
58
59     else if(labelType() == "Nazione") {
60       data(world.cities)
61
62       ...
63
64       output$motifGlobe <- renderGlobe({
65         create_globe()%>%
66         globe_arcs(
67           coords(
68             start_lat = yFrom,
69             start_lon = xFrom,
70             end_lat = yTo,
71             end_lon = xTo,
72             color = color,
73             label = motif,
74             altitude = arcAltitude,
75             stroke = 0.7,
76             dash_length = .2,
77             dash_gap = .08,
78             dash_animate_time = 6000
79           ),
80           data = cbind(globeArcs,legenda,arcAltitude)
81         )%>%
82         globe_choropleth(
83           coords(country = name, long, label = name,cap_color
84             =nodeColors , altitude=0.01),
85           data = cbind(nodesDF,nodeColors),
86           match = "auto"
87         )
88
89       output$legenda <- renderUI({
90         HTML('<h3>Legenda</h3>')
91         legendaRows <- apply(unique(legenda), 1, function(row)
92           {
93             generateSquare(row[2], row[1])
94           })
95         HTML(paste(legendaRows, collapse = ""))
96       })
97     })

```

# Conclusione

Il principale obiettivo di questo lavoro di tesi è stato analizzare la struttura della rete aeroportuale mondiale, data la sua rilevanza nell'ambito dell'economia e della società contemporanea.

I dati riguardanti la rete aeroportuale sono stati ottenuti grazie ad Open-Flights, il quale ha fornito diversi dataset contenenti una vasta gamma di informazioni riguardanti la rete.

Dopo aver discusso le caratteristiche attese da una rete di questo tipo da un punto di vista teorico, sono state condotte diverse analisi sulla rete aeropor-tuale, rappresentata adeguatamente come un multigrafo.

Per facilitare tutte le operazioni previste, è stata sviluppata l'applicazione FlightAnalyzer, che ha consentito di svolgere tutte le analisi previste in maniera intuitiva e veloce.

Le analisi condotte hanno riguardato, nello specifico, il calcolo misure gene-riche della rete, come la *distribuzione dei gradi*, il *coefficiente di clustering*, la *distanza media tra i nodi*, e molte altre. Si è passato poi a implementare la ricerca di aeroporti sulla base di specifici parametri, permettendo poi di visualizzare tutte le occorrenze trovate sulla base dei parametri di ricerca. Sono stati poi calcolati e visualizzati i valori centralità di ognuno dei nodi della rete, sulla base di diverse possibili misure. Infine, è stata implementata la visualizzazione della rete su un globo 3D interattivo.

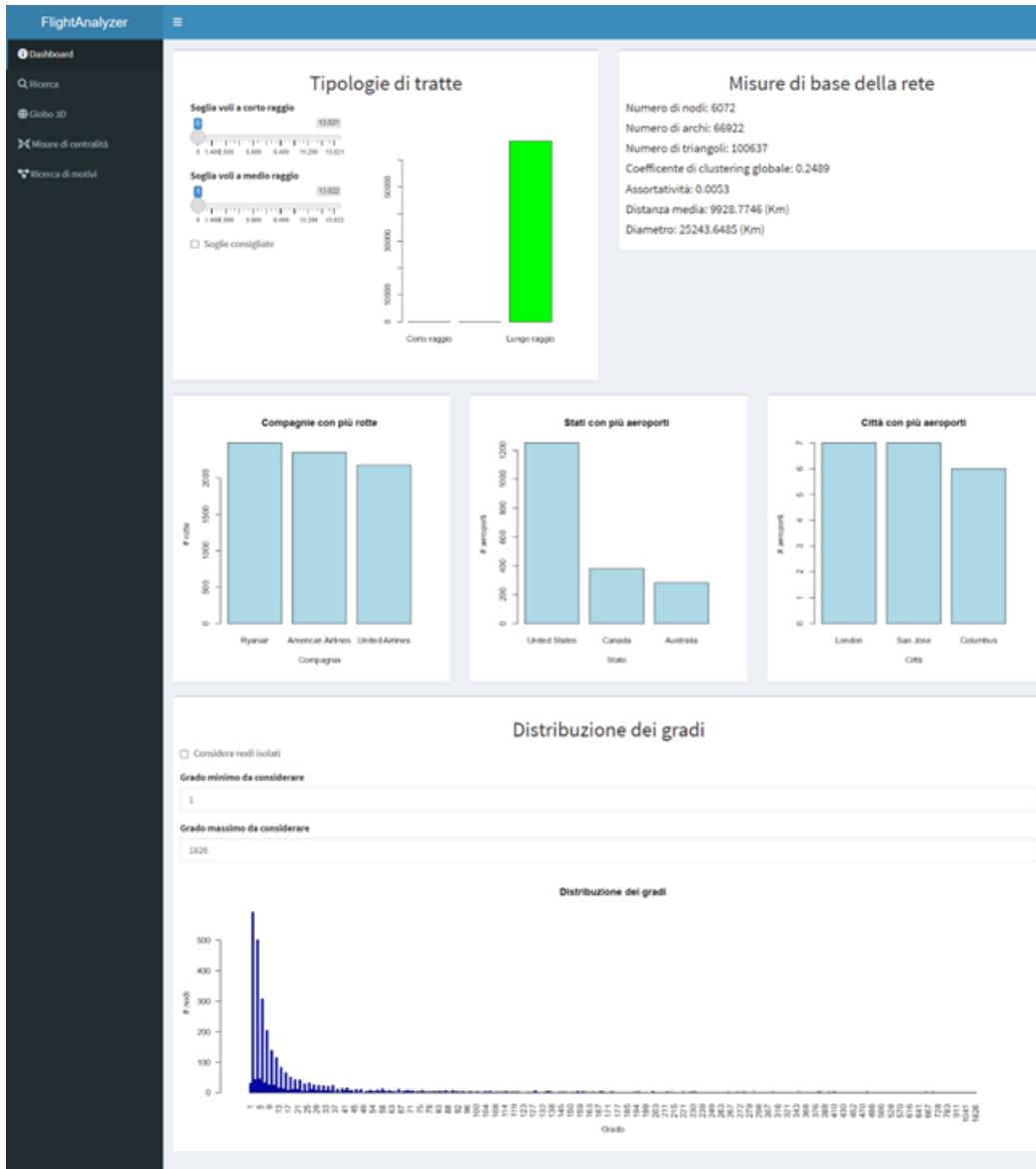
La principale funzionalità offerta dall'applicazione, la ricerca di motivi, ha inoltre permesso l'individuazione di specifici pattern all'interno della rete.

In generale, tutte le informazioni ricavate grazie all'utilizzo di FlightAnalyzer hanno permesso di analizzare al meglio la struttura della rete aeroportuale, e potranno poi essere utilizzate per migliorare la struttura e l'efficienza di quest'ultima.

# **Appendice: screenshot dell'applicazione FlightAnalyzer**

Sono di seguito riportati degli screenshot relativi alle parti salienti presenti all'interno di FlightAnalyzer, l'applicazione sviluppata per questo progetto di tesi.

Per ogni immagine, nella descrizione è riportato il riferimento alla sezione in cui si descrive la parte dell'applicazione riguardante l'immagine in questione.



**Figura 6.9:** Sezione "Dashboard" (ref: sez.6.1)

### Misure di base della rete

Numero di nodi: 6072

Numero di archi: 66922

Numero di triangoli: 100637

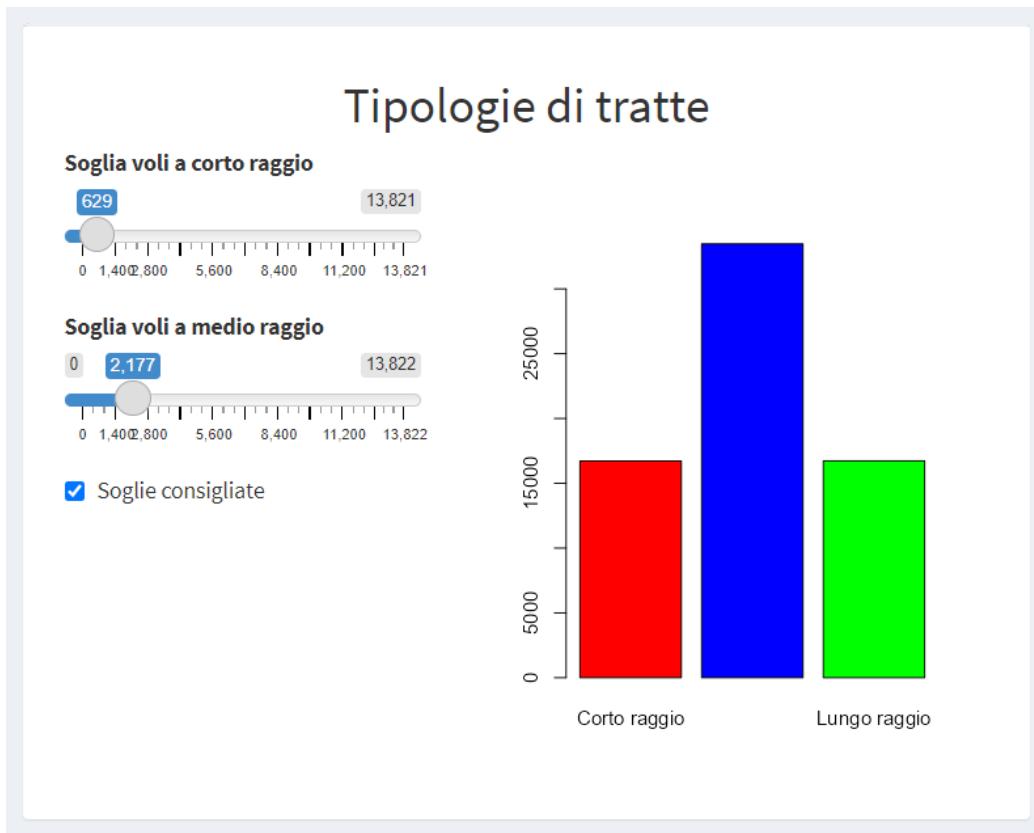
Coefficiente di clustering globale: 0.2489

Assortatività: 0.0053

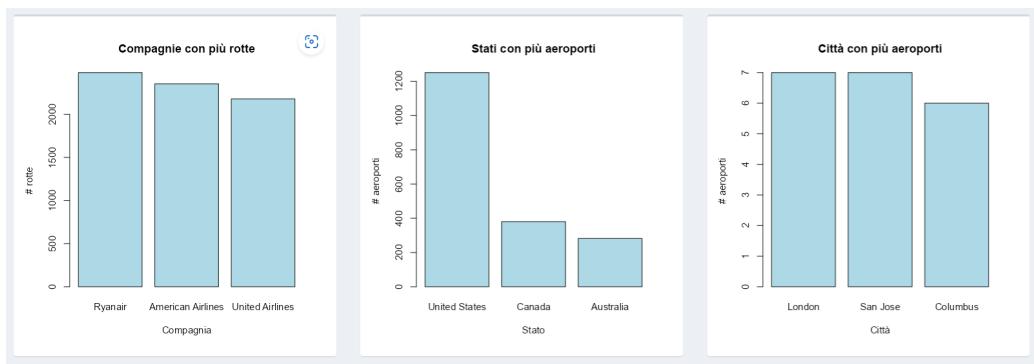
Distanza media: 9928.7746 (Km)

Diametro: 25243.6485 (Km)

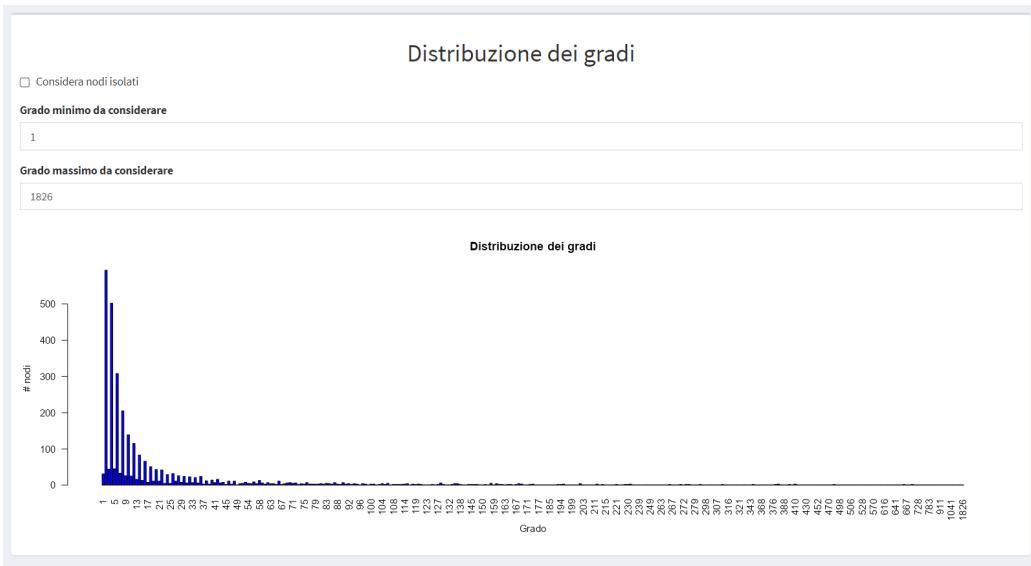
**Figura 6.10:** Dashboard: misure principali della rete (ref: sez.6.1.1)



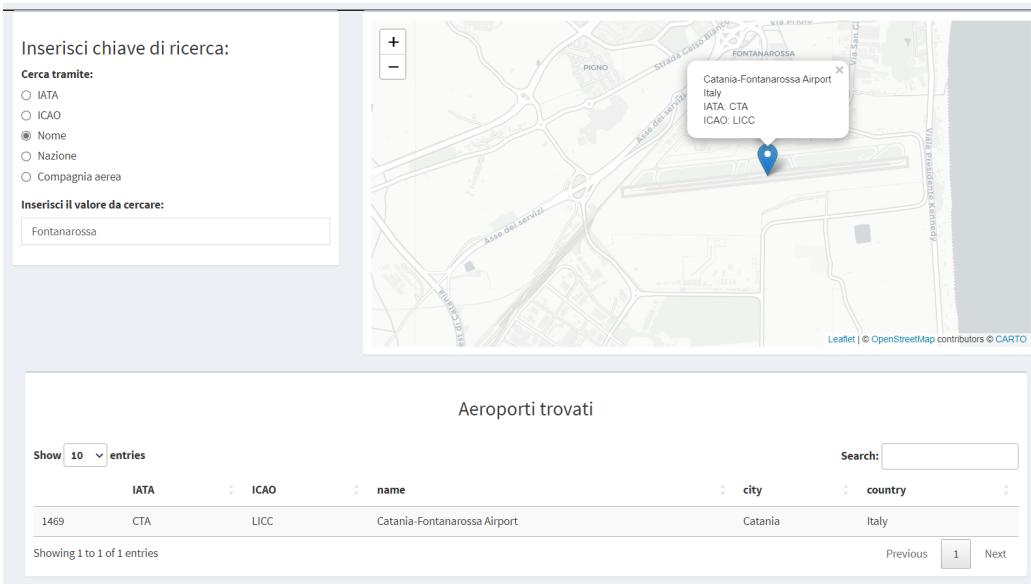
**Figura 6.11:** Dashboard: grafico interattivo della suddivisione delle tratte (ref: sez.6.1.2)



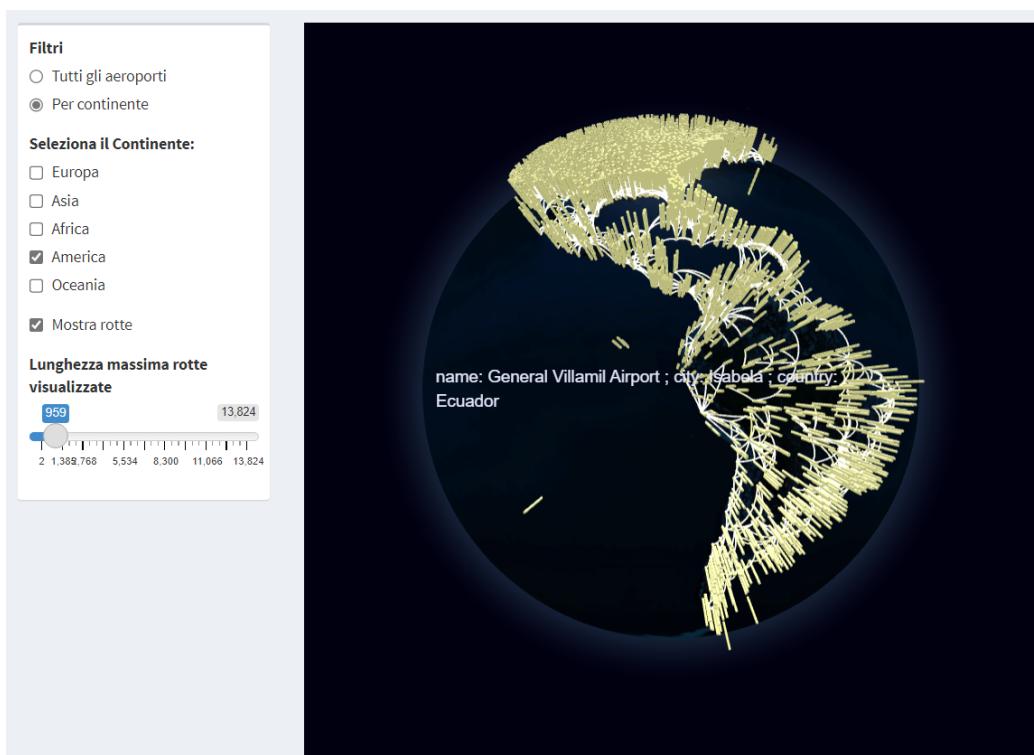
**Figura 6.12:** Dashboard: grafici vari (ref: sez.6.1.3)



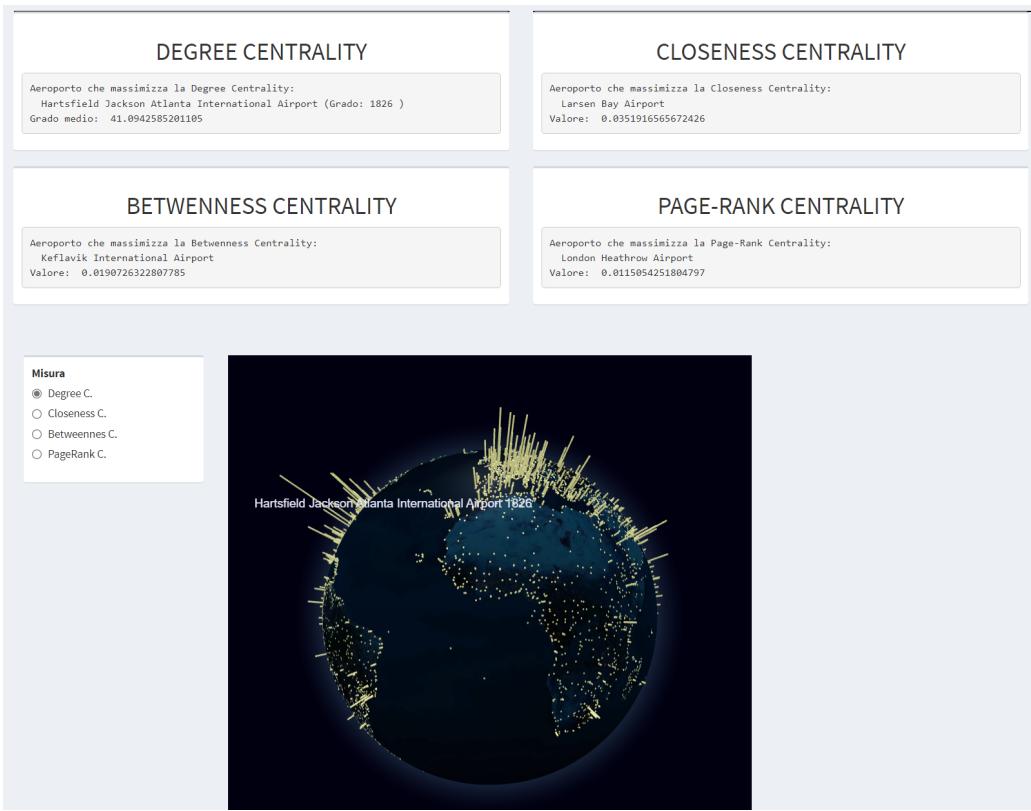
**Figura 6.13:** Dashboard: dsitribuzione dei gradi interattiva (ref: sez.6.1.4)



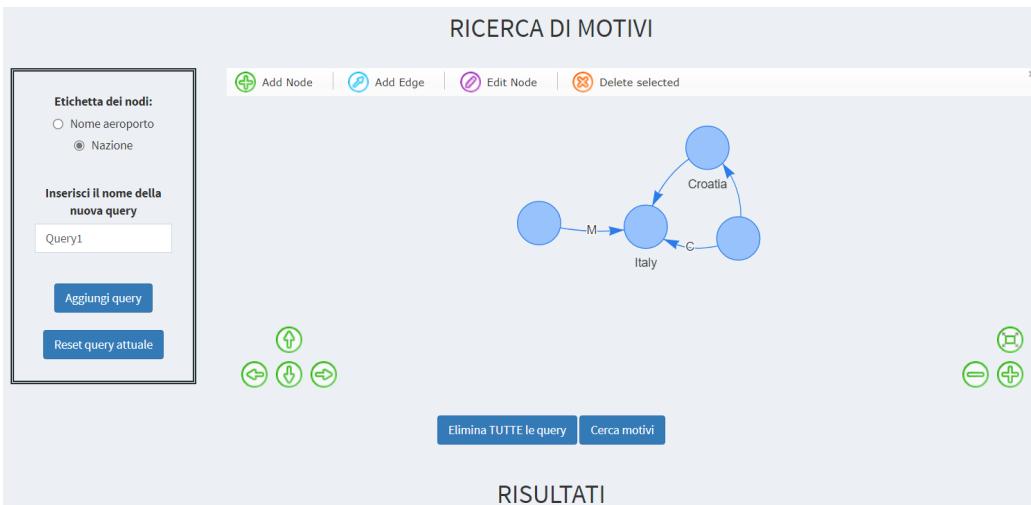
**Figura 6.14:** Sezione di ricerca aeroporti. Nell'esempio, ricerchiamo l'aeroporto "Fontanarossa" di Catania (ref: sez.6.2)



**Figura 6.15:** Sezione di visualizzazione 3D della rete e tutte le possibili opzioni (ref: sez.6.3)



**Figura 6.16:** Sezione di calcolo misure di centralità e visualizzazione su globo 3D. Nell'esempio, visualizziamo sul globo la Degree Centrality dei nodi (ref: sez.6.4.5)



**Figura 6.17:** Sezione di ricerca dei motivi: disegno di un motivo da ricercare (ref: sez.6.5.4.3)

RISULTATI						
Show	10	entries	Search:			
Motif_Nodes	Motif_edges	Num_occ_input_graph	Mean_analytical	Variance_analytical	Pval_analytical	
1 Cape Verde,Spain,Italy	(0,1,M)(2,1,L)	2	0.017604185786787654	0.029881696540205876	8.994569011405718E-4	
2 Ireland,Spain,Italy	(0,1,L)(2,1,L)	47	1.282668966329429	4.404426529784769	5.991995788434679E-11	
3 Italy,Spain,Argentina	(0,1,C)(2,1,L)	1	0.10294339234915797	0.1567649517614852	0.0187415358636992	
4 Italy,Spain,Argentina	(0,1,M)(2,1,L)	29	0.9392415073806148	4.690546534420082	1.1100112172979237E-5	
5 Sweden,Spain,Italy	(0,1,M)(2,1,M)	23	14.936440419404809	181.82071847284797	0.22049041018679327	
6 Croatia,Spain,Italy	(0,1,M)(2,1,M)	30	13.39537910629161	146.85499051233114	0.0946066840618206	
7 Italy,Spain,Pakistan	(0,1,M)(2,1,L)	30	1.7269924490546809	9.286474060455625	5.89566234124872E-5	
8 Morocco,Spain,Italy	(0,1,M)(2,1,M)	145	13.513922284223401	140.11305122421408	4.9175805316714616E-8	
9 Croatia,Spain,Italy	(0,1,M)(2,1,C)	2	0.3547677826399044	0.6395485987230844	0.0305709975455124	
10 Morocco,Spain,Italy	(0,1,M)(2,1,L)	6	0.5017192949234489	1.2612776408777029	0.003895258038542515	

Showing 1 to 10 of 173 entries

Previous 1 2 3 4 5 ... 18 Next

**Figura 6.18:** Sezione di ricerca dei motivi: visualizzazione dei risultati della ricerca di motivi (ref: sez.6.5.4.5)



**Figura 6.19:** Sezione di ricerca dei motivi: visualizzazione di alcuni dei motivi trovati su un globo 3D (ref: sez.6.5.4.6)

# Bibliografia

- [1] IEA, World air passenger traffic evolution, 1980-2020, IEA, Paris <https://www.iea.org/data-and-statistics/charts/world-air-passenger-traffic-evolution-1980-2020>, IEA. Licence: CC BY 4.0. <https://www.iea.org/data-and-statistics/charts/world-air-passenger-traffic-evolution-1980-2020>.
- [2] Jani Patokallio. OpenFlights Reference. <https://openflights.org/about> , <https://openflights.org/data>.
- [3] R Development Team. <https://www.r-project.org/>.
- [4] Shiny Development Team. <https://shiny.posit.co/>.
- [5] iGraph Development Team. <https://igraph.org/>.
- [6] ggplot2 Development Team. <https://ggplot2.tidyverse.org/>.
- [7] Leaflet Development Team. <https://leafletjs.com/>.
- [8] visNetwork Development Team. <https://datastorm-open.github.io/visNetwork/>.
- [9] G. Micale, A. Pulvirenti, A. Ferro, R. Giugno, and D. Shasha. Fast methods for finding significant motifs on labelled multi-relational networks. *Journal of Complex Networks*, 2019.
- [10] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics*, 2013.
- [11] Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002.