Università degli Studi di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea in Inforamtica

Regole di Associazione per Mutazioni Genetiche e loro visualizzazione su Reti Biologiche

Simone Basile

Relatore
Prof. Alfredo Ferro

Correlatore
Dott. Giovanni Micale

Anno Accademico 2019/20

Ai miei amici,
alla mia famiglia,
a mio padre.

Indice

Sommario

Il presente elaborato di tesi propone un'analisi delle reti biologiche (o pathway) al fine di individuare in quest'ultime quali geni subiscono delle mutazioni se fissato a priori un **gene driver** come mutato. Il tutto, cominciando da un'analisi generale, verrà focalizzato sui geni e le pathway che possono determinare il tumore al seno.
Il progetto di tesi, partendo dall'algoritmo **DMGSFinder** sviluppato dal dottor Giovanni Micale, è stato svolto mediante due linguaggi di programmazione ausiliari, ovvero **C++** e **R**, i quali grazie alle loro librerie native o di terze parti hanno permesso l'analisi e la manipolazione dei dati a partire dalla fase della loro estrapolazione fino alla loro visualizzazione mediante la libreria **Shiny** di R.

1
Introduzione

1.1 Aspetti caratteristici

Nonostante si concentri maggiormente nell'ambito informatico, l'elaborato di tesi qui presentato si dirama in altre discipline, quali la *Biologia* e la *Biomedicina*, le quali introduciamo rapidamente.

1.1.1 Biologia

È la scienza che studia la **vita**, ovvero i processi fisici e chimici dei fenomeni che caratterizzano i sistemi viventi, includendo la loro **genetica**, meccanismi molecolari, anatomia, fisiologia, biochimica, nonché **processi emergenti** che mostrano come un sistema complesso abbia proprietà macroscopiche ben definite, difficilmente predicibili sulla base delle leggi che governano le sue componenti considerate singolarmente. Esempi di questi processi emergenti sono l'adattamento, l'evoluzione, l'interazione tra gli organismi, lo sviluppo ed infine il comportamento.

Di fondo esistono alcuni concetti comuni e unificanti all'interno della biologia che ne determinano lo studio e la ricerca. Si riconosce infatti la **cellula** come l'unità base della vita, i **geni** come struttura di base dell'ereditarietà e l'evoluzione darwiniana per selezione naturale che ne regola il processo di nascita ed estinzione della specie.

1.1.2 Biomedicina

È un ramo della scienza medica che inserisce i principi biologici delle scienze naturali alla pratica in ambito clinico.

Attualmente la biomedicina è il fulcro in cui la sanità moderna e la diagnostica di laboratorio si incontrano. Tale ambito è basato sulla biologia molecolare e combina tutte le questioni relative allo sviluppo della medicina molecolare in relazioni strutturali e funzionali su larga scala del genoma, del trascrittoma, del protenoma, del fisioma e del metaboloma umano al fine di ideare nuove tecnologie per la previsione, la diagnosi e la terapia.

1.2 Concetti preliminari

In questa sezione si introducono alcuni concetti essenziali, in ambito sia biologico che informatico, che saranno utili nei capitoli successivi della tesi.

1.2.1 I grafi

Si definisce grafo G una struttura relazionale formata da una coppia di insiemi (V, E) dove V è detto insieme dei nodi e E è detto insieme degli archi ed è un sottoinsieme di tutti le possibili coppie di nodi in V. Se le coppie di nodi sono ordinate, il grafo è detto **orientato**, se non sono ordinate è detto **non orientato**.

1.2.2 Misure di centralità di un grafo

Le più utilizzate nell'ambito della teoria dei grafi sono: Betweenness, Closeness, Degree e PageRank centrality.

- **Betweenness centrality** è una misura di centralità di una rete basata su percorsi più brevi. Per ogni coppia di vertici in un grafo collegato, esiste almeno un percorso più breve tra i vertici in modo che il numero di archi che il percorso attraversa (per i grafi non pesati) o la somma dei pesi degli archi (per i grafi pesati) sia minima.

- **Closeness centrality** è una misura della centralità di una rete, calcolata come il reciproco della somma della lunghezza dei percorsi più brevi tra il nodo e tutti gli altri nodi nel grafo. Pertanto, più un nodo è centrale, più è vicino a tutti gli altri nodi.

- **Degree centrality** è una semplice misura di centralità che conta quanti vicini ha un nodo. Se la rete è diretta, abbiamo due versioni della misura: *in-degree* è il numero di collegamenti in arrivo o il numero di nodi precedenti; *out-degree* è il numero di link in uscita o il numero di nodi successivi. In genere, siamo interessati alla misura di in-degree, poiché i collegamenti in entrata sono dati da altri nodi nella rete, mentre i collegamenti in uscita sono determinati dal nodo stesso.

- **PageRank centrality** si basa sul concetto che un nodo è tanto più importante tanto più è collegato ad altri nodi importanti. Ci sono tre fattori distinti che determinano il PageRank di un nodo, ovvero, il numero di collegamenti che riceve, la propensione dei collegamenti e la centralità dei collegamenti.

1.2.3 La cellula

È la più piccola struttura definibile come vivente, nonché l'unità morfofunzionale di ciascun organismo.

La cellula rappresenta uno spazio allo stesso tempo separato dall'ambiente esterno e in contatto con esso; la presenza di una **membrana cellulare** divide infatti l'ambiente cellulare da quello esterno ma meccanismi di trasporto sia **attivo** che **passivo** permettono lo scambio di sostanze necessario all'acquisizione dei nutrienti, all'eliminazione degli scarti metabolici e al mantenimento del giusto equilibrio tra le varie sostanze disciolte nel liquido che riempie lo spazio cellulare, detto **citosol**.

Tutti gli organismi viventi si possono differenziare in due gruppi a seconda della struttura fondamentale delle loro cellule: i **procarioti** e gli **eucarioti**. La differenza sostanziale tra questi due gruppi sta nell'organizzazione cellulare e sopratutto nella locazione del DNA. Mentre nelle cellule eucariote il DNA è situato all'interno del

nucleo sottoforma di **cromatina**, nelle procariote si trova libero nel citoplasma in una regione interna della cellula detta **nucleoide**.

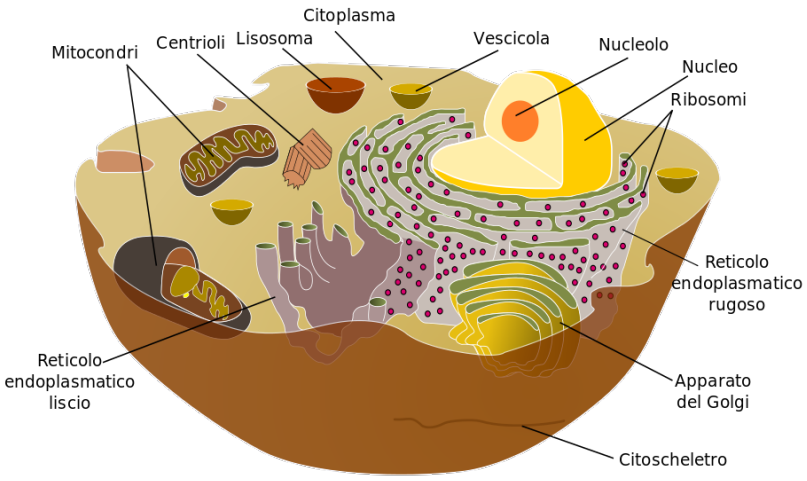

Figura 1.1: Struttura della cellula

1.2.4 Il DNA

Il **DNA**, o **acido desossiribonucleico**, è il patrimonio genetico di moltissimi organismi viventi, essere umano compreso.

Contenuto nel nucleo della cellula sotto forma di ***cromatina***, il DNA appartiene alla categoria degli **acidi nucleici**, cioè grandi molecole biologiche formate da unità più piccole dette ***nucleotidi***.

I nucleotidi che compongono il DNA sono: l'adenina (A), la citosina (C), la guanina (G) e la timina (T).

Un generico nucleotide comprende 3 elementi: un **gruppo fosfato**, lo zucchero **desossiribosio** e una **base azotata**.

Organizzato in **cromosomi**, il DNA serve alla generazione delle proteine, le quali giocano un ruolo fondamentale nel regolare tutti i meccanismi cellulari di un organismo.

1.2.5 Il gene

È una specifica porzione di DNA che contiene l'informazione per l'espressione di un particolare carattere o una particolare funzione. I geni sono l'unità funzionale in cui è suddiviso il DNA e sono la base molecolare dell'ereditarietà dei caratteri.

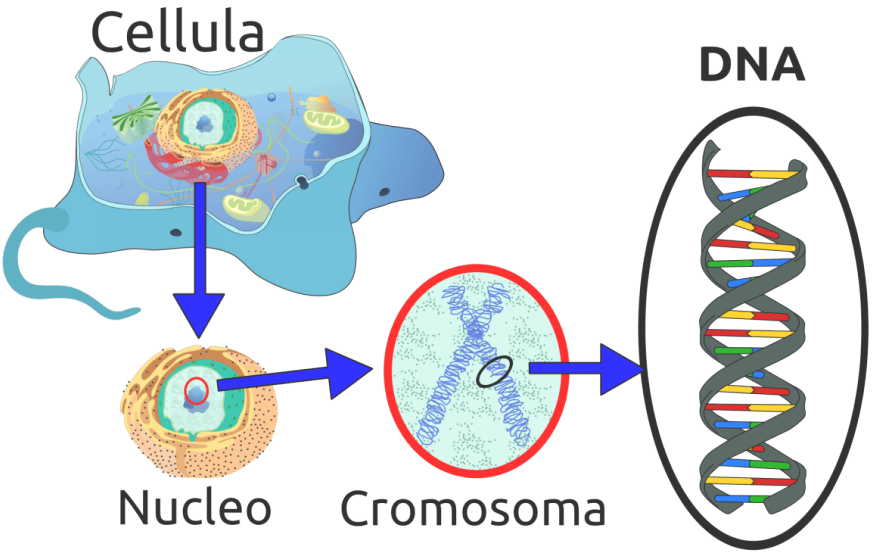

Figura 1.2: Dalla cellula al DNA

1.2.6 Le mutazioni genetiche

È un cambiamento di coppia di basi o un cambiamento in cromosoma che altera la lettura della sequenza delle basi del DNA.
Se una mutazione avviene a livello di basi si parla di **mutazioni geniche** e quando coinvolge una sola base si parla di **mutazioni puntiformi**.

Le mutazioni possono inoltre essere **spontanee**, ovvero che avvengono naturalmente, o **indotte**, che avvengono quando un organismo è esposto deliberatamente o casualmente ad agenti fisici o chimici, detti ***mutageni***, che interagiscono con il DNA provocando delle mutazioni.

1.2.7 Le pathway biologiche

È un percorso biologico che ritrae una serie di interazioni tra molecole in una cellula che porta ad una determinata funzione o cambiamento cellulare. Durante il percorso si può incorrere all'innesco di nuove molecole, come una proteina o altri geni interessati.

I percorsi possono attivare o inibire determinati geni o stimolare una cellula nel cammino predestinato. I percorsi biologici più comuni sono:

- Metabolic pathway
- Genetic pathway
- Signal transduction pathway

Al giorno d'oggi esistono diversi database che offrono la possibilità di visionare questo grafo di interazioni tra geni come ad esempio: KEGG, Reactome, PANTHER e molti altri ancora. Nella lavoro di tesi viene utilizzato il database KEGG per ottenere la pathway relativa al tumore al seno.

1.2.8 Il tumore al seno

Il seno è costituito da un insieme di ghiandole e tessuto adiposo ed è posto tra la pelle e la parete del torace. In realtà non è una ghiandola sola, ma un insieme di strutture ghiandolari, chiamate ***lobuli***, unite tra loro a formare un lobo. Il latte giunge al capezzolo dai lobuli attraverso piccoli tubi chiamati ***dotti galattofori*** (o lattiferi).

Il tumore al seno è una malattia potenzialmente grave se non è individuata e curata per tempo. È dovuto alla **moltiplicazione incontrollata** di alcune cellule della ghiandola mammaria che si trasformano in cellule maligne. Ciò significa che hanno la capacità di staccarsi dal tessuto che le ha generate per invadere i tessuti circostanti e, col tempo, anche gli altri organi del corpo. In teoria si possono formare tumori da tutti i tipi di tessuti del seno, ma i più frequenti nascono dalle **cellule ghiandolari** (dai lobuli) o da quelle che formano la **parete dei dotti**.

Distinguiamo due forme di cancro al seno:

- Le forme non invasive, tra le quali troviamo la ***neoplasia duttale intraepiteliale*** (**DIN**) e la ***neoplasia lobulare intraepiteliale*** (**LIN**)
- Le forme invasive come il ***carcinoma duttale*** o il ***carcinoma lobulare***

Il tumore al seno può essere classificato in 5 stadi, dallo stadio 0 allo stadio 4. Sia la prognosi che il trattamento sono influenzati dallo stadio in cui la neoplasia (ovvero la crescita incontrollata e scoordinata di un gruppo di cellule) si trova al momento della diagnosi.

Al giorno d'oggi esistono diverse cure mediche come: la chirurgia, la radioterapia, la chemioterapia, l'ormonoterapia e le terapie biologiche.

2
Ricerca del DMGS

In questo capitolo verrà illustrata la prima fase del progetto, ovvero la ricerca dei DMGS, che è stata effettuata mediante l'algoritmo **DMGSFinder**, sviluppato dal dottor Giovanni Micale.

Prima di passare alla spiegazione di ciò che è stato fatto partiamo con una serie di definizioni preliminari utili a comprendere al meglio l'obiettivo dell'algoritmo e la sua applicazione ai fini del progetto.

2.1 Definizioni preliminari

2.1.1 Matrice delle mutazioni

È la matrice $\mathcal{A}(G, S)$ di mutazioni geniche in un set di campioni, di dimensione $n \times m$. G è l'insieme dei geni, S è l'insieme dei campioni. Le righe sono i geni e le colonne sono i campioni.

$\mathcal{A}$(G,S)$[i][j] = 1$ se il gene i è stato segnalato come mutato nel campione j.

$\mathcal{A}(G, S)[i][j] = 0$ se il gene i è stato segnalato come non-mutato nel campione j.

2.1.2 Insiemi positivi e negativi

Dall'insieme dei campioni è possibile definire due partizioni, **positivo** (P) e **negativo** (N), le quali rappresentano due classi di pazienti con differenti fenotipi.

I campioni nell'insieme positivo sono chiamati positivi, mentre quelli nell'inseme negativo sono chiamati negativi.

2.1.3 Coperture di un gene

Sia $g \in G$ e $S' \subseteq S$. La **copertura** di g in S', $Cov(g, S')$, è il set di campioni di S' nel quale g è mutato.

Con $Cov'(g, S')$ si denota il complemento di $Cov(g, S')$ rispetto a S. Se $s \in Cov(g, S')$ allora s è *coperto* da g, altrimenti se $s \notin S'$ allora s è *non coperto* da g.

2.1.4 Coperture di un set di geni

Sia $S' \subseteq S$ e $X = \{x_1, x_2, ..., x_{\mathrm{h}}, \bar{x}_{\mathrm{h+1}}, ..., \bar{x}_{\mathrm{n}}\} \subseteq G$ un sottoinsieme di n geni di G.

La **copertura** di X in S', $Cov(X, S')$, è il sottoinsieme dei campioni di S' che sono:

- Coperti da almeno uno dei geni $x_1, x_2, ..., x_{\mathrm{h}}$;
- Oppure non sono coperti da almeno un gene $\bar{x}_{\mathrm{h+1}}, ..., \bar{x}_{\mathrm{n}}$.

Formalmente:

$$Cov(X, S') = \big(\bigcup_{i=1}^{h} Cov(x_{\mathrm{i}}, S') \big) \cup \big(\bigcup_{j=h+1}^{n} Cov'(x_{\mathrm{i}}, S') \big)$$

2.1.5 Supporto di un set di geni

Sia $S' \subseteq S$. Il supporto di un set di geni $X = \{x_1, x_2, ..., x_{\mathrm{h}}, \bar{x}_{\mathrm{h+1}}, ..., \bar{x}_{\mathrm{n}}\} \subseteq G$ in S' è la percentuale di campioni di S' coperta da X.

Formalmente:

$$Supp(X, S') = \frac{|Cov(X, S')|}{|S'|}$$

2.1.6 Supporto differenziale

Sia $X = \{x_1, x_2, ..., x_{\mathrm{h}}, \bar{x}_{\mathrm{h+1}}, ..., \bar{x}_{\mathrm{n}}\} \subseteq G$ un sottoinsieme di n geni di G. Sia $P \subseteq S$ e $N \subseteq S$ i set positivi e negativi rispettivamente. Il **supporto differenziale** di X rispetto P e N è definito come:

$$DiffSupp(X, P, N) = Supp(X, P) - Supp(X, N)$$

2.1.7 Set di geni differenzialmente mutati (DMGS)

Sia $P \subseteq S$ un set di *positivi* e $N \subseteq S$ un set di *negativi*. Un **set di geni differenzialmente mutato** (**DMGS**) è un sottoinsieme X di geni che massimizzano il supporto differenziale di X rispetto a P e N, ovvero $DiffSupp(X, P, N)$.

2.1.8 Problema del Min-DMGS

Il problema del **Min-DMGS** consiste nella ricerca del DMGS minimale. Il DMGS *minimale* è il DMGS con la cardinalità minima.

2.2 DMGSFinder

2.2.1 Breve descrizione dell'algoritmo

DMGSFinder è un algoritmo **greedy** per la ricerca del DMGS.

Un DMGS è un set di geni che massimizzano il supporto differenziale rispetto a un set di campioni positivi e un set di campioni negativi.

È possibile risolvere due problemi di ottimizzazione utilizzando DMGSFinder:

- Il problema del min-DMGS: ricercando il DMGS minimale, ovvero una DMGS con il minimo numero di geni;
- Il problema del DMGS ponderato massimo: trovando il DMGS che massimizza il peso di una soluzione, dato un peso per ogni gene.

2.2.2 Flusso di esecuzione di DMGSFinder

Descriviamo brevemente il flusso di esecuzione di DMGSFinder:

1. Viene aggiunto ogni gene mutato e ogni gene non-mutato alla lista dei candidati, ovvero l'insieme di elementi che possono far parte della soluzione finale;
2. Vengono calcolati due punteggi di copertura:

- Il punteggio di copertura dei positivi;
- Il punteggio di copertura dei negativi.

3. Vengono ordinati i candidati sulla base dei loro punteggi di copertura;
4. Vengono processati i candidati uno alla volta seguendo l'ordine del punto precedente e per ogni candidato c:
   (a) Decide se includere o meno c nell'attuale soluzione ottimale trovata;
   (b) Se c è stato incluso nella corrente soluzione ottimale, riduce la soluzione rimuovendo i candidati precedentemente aggiunti, se necessario.
5. Ritorna la soluzione migliore che è stata trovata.

2.3 Applicazione al progetto

L'algoritmo DMGSFinder ha posto le basi per lo sviluppo del progetto di tesi. Ponendoci infatti l'obiettivo di ricercare per ogni gene driver mutato il relativo DMGS, è stato possibile raccogliere tutti i dati necessari per gli step successivi di analisi e la conclusiva fase di visualizzazione dei risultati ottenuti.

Partendo dalla matrice di germinazione e la lista dei geni driver è stato possibile sviluppare un piccolissimo script **C++** che ha permesso l'automazione della generazione dei DMGS per tutti i geni driver e il consecutivo salvataggio dei risultati.

```
1 int main() {
2     while(!geneList.eof()) {
3         string gene;
4         geneList >> gene;
5 
6         system(("java -cp ./out DMGSFinder -m data/
    BRCA_germline_matrix_EXON.txt -p genes " + gene + " -o results/"
     + gene + ".txt").c_str());
7     }
8 }
```

3

Ricerca dei geni mutati nelle pathway

In questo capitolo verrà illustrata la fase di estrazione dei dati utili dagli output dell'algoritmo DMGSFinder e la fase di ricerca dei geni mutati in ogni pathway per ogni gene driver considerato. In entrambe le fasi citate sono stati generati degli **script ausiliari in linguaggio C++**, al fine di velocizzare la fase di estrazione e di matching. Prima però sarà necessario dare una definizione preliminare alle **espressioni regolari**, uno strumento molto potente che ha permesso di trovare le corrispondenze dei pattern sulle stringhe analizzate.

3.1 Le espressioni regolari

Dato un alfabeto Σ e dato l'insieme di simboli $\{+, *, (,), ., \emptyset\}$, si definisce *espressione regolare* sull'alfabeto Σ una stringa $r \in (\Sigma \cup \{+, *, (,), ., \emptyset\})^+$ tale che valga una delle seguenti condizioni:

1. $r = \emptyset$
2. $r \in \Sigma$
3. $r = (s + t)$, oppure $r = (s \cdot t)$, oppure $r = s^*$, dove s e t sono espressioni regolari sull'alfabeto Σ.

3.2 Estrazione dei dati utili

L'algoritmo DMGSFinder, descritto nel capitolo precedente, ha generato in output tanti file di testo quanti sono i geni driver per i quali è stata effettuata la ricerca del DMGS.

I risultati ottenuti sono stati poi ulteriormente manipolati al fine di ottenere dei file strutturati secondo dei precisi parametri per poter procedere con la fase successiva del progetto.

```
DMGS    Average positive coverage       Average negative coverage       Differential coverage
[DOCK11, SGCD, OR10H3, TTC3, CASP4, ERBIN, PLEKHA6, KIAA1958, FER1L6, TANC2]    74.78%  17.98%  56.8%
```

Figura 3.1: Esempio di DMGS calcolata sul gene driver BRCA1

Per il raggiungimento di tale obiettivo, è stato necessario costruire un'espressione regolare in grado di estrarre solo i geni appartenenti al campo DMGS delle soluzioni generate dall'algoritmo, escludendo le parentesi quadre (Figura 3.1). Il codice sottostante mostra l'espressione regolare che è stata applicata.

```
1    regex e("(<?\\[)((\\w*,)*\\w*)", ECMAScript | icase );
```

Successivamente, mediante l'ausilio del metodo *split* appositamente creato, è stato possibile ottenere una collezione di stringhe rappresentanti i singoli geni facenti parte della soluzione; tale collezione è stata riportata in più file di testo, uno per gene driver. Il metodo *split* citato, come mostrato dal codice sottostante, prende in input due parametri, rispettivamente la stringa da voler splittare e il separatore su cui voler effettuare lo split, e restituisce in output un **vector di stringhe** che verrà poi utilizzato nel *main* per effettuare la scrittura su file.

```
1    vector<string> split(string str, string sep) {
2        char* cstr = const_cast<char*>(str.c_str());
3        char* current;
4        vector<string> arr;
5        current = strtok(cstr,sep.c_str());
6        while(current != NULL) {
7            arr.push_back(current);
8            current = strtok(NULL, sep.c_str());
9        }
10       return arr;
11   }
```

3.3 Ricerca dei geni mutati

In seguito all'intervento fatto sui file nel passo precedente, è stato applicato un algoritmo che ha permesso di ricercare le corrispondenze tra i geni appartenti alle singole pathway e i DMGS precedentemente estratti.

Questo algoritmo, ciclando su tutte le pathway prese in considerazione, **associa** per tutti i geni contenuti nella pathway un valore booleano che indica se è stata trovata per un dato gene una corrispondenza nel DMGS.

Il primo passo è stato definire, mediante una struct, il tipo *nodes* il quale mantiene due variabili, una stringa e un intero, che rappresentano rispettivamente il **nome del gene** e il **valore booleano** che gli verrà assegnato. Questa struct diventa necessaria in quanto per ogni gene appartenente ad una pathway possiamo creare una coppia *(chiave, valore)* che sarà molto utile durante le fasi successive.

```
1     struct nodes {
2       string node;
3       int weight;
4
5       nodes() {
6         weight = 0;
7       }
8     };
```

Il metodo che svolge il compito di effettuare l'associazione tra i geni di una pathway e il DMGS relativo ad un determinato gene driver è *createOutputWeights*. Esso prende in input tre parametri, ovvero un array di *nodes*, una stringa ed un intero, e ritorna in output un array di *nodes* nel quale sono già state effettuate le associazioni e pertanto è pronto per essere riportato in output.

```
1     nodes* createOutputWeights(nodes* g, string driver, int n) {
2       ifstream driver_genes_results(("input/" + driver + ".txt").
    c_str());
3
4       set<string> pat;
5
6       while(!driver_genes_results.eof()) {
7         string a;
8         driver_genes_results >> a;
9         pat.insert(a);
```

```
10      }
11
12      for(int i = 1; i < n; i++){
13        for(set<string>::iterator it = pat.begin(); it != pat.end()
    ; ++it) {
14              string a = *it;
15              if(a == g[i].node){
16              g[i].weight = 1;
17            }
18        }
19      }
20
21      return g;
22    }
```

Gli output generati verranno poi scritti in file di testo, rinominati secondo lo schema *genedriver_pathway_out.txt*, e svolgeranno un ruolo essenziale durante la fase di plotting.

4

Visualizzazione delle reti biologiche

In questo capitolo verrà illustrata la fase di visualizzazione delle reti biologiche, che rappresenta un punto cruciale nel progetto di tesi. Basandoci sui risultati precedentemente ottenuti sarà possibile eseguire i plot delle singole pathway evidenziando i geni che le compongono e quelli che hanno subito una mutazione. Prima di spiegare i passi che sono stati necessari per eseguire tale fase, è importante fare una piccola introduzione a ***Pathview***, ovvero il tool di R che ha permesso di poter eseguire il plotting.

4.1 Pathview

4.1.1 Panoramica del tool

Pathview è un package che mette a disposizione un set di strumenti per l'integrazione dei dati basati sulle pathway e per la loro visualizzazione. Esso effettua la **mappatura** e il **rendering** dei dati degli utenti sui plot delle pathway. Tutto che gli utenti devono fare è fornire i dati relativi al gene o al composto e specificare la pathway su cui effettuare l'analisi; *Pathview*, infatti, scarica automaticamente il grafico della pathway richiesta e il relativo file con i metadati associati, mappa i dati dell'utente su essa e ne esegue il rendering grafico.

4.1.2 Analisi dei parametri

Il metodo principale è l'omonimo *pathview*, il quale è capace di effettuare le quattro fasi principali del package: **Download**, **Parser**, **Mapper** e **Viewer**. Tale metodo prende in input diversi parametri ma, in questa sede, analizzeremo solo quelli che sono stati essenziali per il plotting delle pathway di nostro interesse.

- **gene.data** rappresenta il database di tutti i geni sul quale effettuare l'analisi dei dati;
- **pathway.id** è la pathway per la quale si vuole effettuare l'analisi e il successivo plotting;
- **species** rappresenta la specie dei geni per cui si è effettuata l'analisi, di default è impostata a *hsa* cioè la specie dei geni relativa all'*homo sapiens*;
- **kegg.dir** serve per indicare la directory su cui salvare i file .png e .xml relativi alla pathway;
- **out.suffix** serve ad aggiungere un suffisso al nome del file che è stato generato;
- **discrete** è una lista di due elementi booleani, rispettivamente *gene* e *cpd*, che indica se trattare i geni e/o i composti come elementi discreti o continui;
- **bins** è una lista di due valori interi, rispettivamente *gene* e *cpd*, che serve a specificare il numero di livelli per i geni e/o i composti in fase di conversione in pseudo-colori;
- **low**, **mid**, **high** sono dei parametri formati da una lista di due colori, rispettivamente *gene* e *cpd*, che servono a specificare lo spettro dei colori da applicare ai geni e/o ai composti;

4.2 Generazione dei plot con R

La fase di generazione dei plot è essenziale per lo sviluppo del progetto, in quanto rappresenta il punto di convergenza di tutti i dati finora raccolti.

Il linguaggio di programmazione su cui è stato fatto affidamento è **R** che, integrando perfettamente il package *Pathview*, ha permesso di sviluppare uno script utile ai fini della generazione dei singoli plot.

In questo paragrafo analizzeremo step by step tutto ciò che è stato fatto in R per la manipolazione del database dei geni e la successiva visualizzazione delle reti biologiche.

4.2.1 Impostazione degli assets e dei cicli

Come si può notare dal codice sottostante, il primo step necessario è stato l'importazione delle librerie e l'impostazione dei percorsi di lavoro. Oltre alla libreria di *Pathview*, di cui abbiamo parlato nei paragrafi precedenti, spicca la libreria ***org.Hs.eg.db*** che ha svolto un ruolo essenziale per tradurre i nomi dei geni nei relativi **EntrezID** in KEGG per permettere così la manipolazione del database ***gse16873.d*** ai fini del plotting.

Successivamente sono state lette le liste relative ai nomi delle pathway e dei geni driver per poter effettuare delle iterazioni che hanno permesso di associare ad ogni gene driver tutte le pathway prese in considerazione. Infatti, iniziando a ciclare sul file dei nomi di tutti i geni driver ed effettuando un seconda iterazione sui nomi delle pathway, è stato possibile costruire tutte le possibili combinazioni *gene driver - pathway*.

```
1    library(pathview)
2    library(org.Hs.eg.db)
3    library(data.table)
4
5    directory <- "C:/Users/simon/Desktop/Tesi/3.PlottingPathview/"
6
7    #Imposto la directory nella quale salvare i plot
8    setwd("C:/Users/simon/Desktop/Tesi/3.PlottingPathview/plots")
9
10   #Inizializzo il database per la traduzione dei nomi dei geni in
     EntrezID
11   hs <- org.Hs.eg.db
12
13   #Leggo la lista di tutte le pathway
14   pathway_name_list <- scan(paste(directory, "data/PathwaysName.
     txt", sep = ""),
15                             what = "",
```

```
                                sep = "\n")

 #Leggo la lista di tutti i geni driver
 gene_driver_list <- scan(paste(directory, "data/geneList.txt",
sep = ""),
                          what = "",
                          sep = "\n")

 for(i in 1:length(gene_driver_list)) {
   current_gene <- gene_driver_list[i]

   for(j in 1:length(pathway_name_list)) {
     #Inizializzo il database da manipolare per effettuare il
plot finale
     data("gse16873.d")

     current_pathway <- pathway_name_list[j]

     ...
   }
 }
```

4.2.2 Manipolazione del database finale per il plotting

In questo step si è proceduto con la manipolazione del dataset **gse16837.d** al fine di arrivare ad una struttura composta da un unico campo per ogni singolo gene, appartenente alla pathway corrente, contenente il valore booleano che è stato associato dal codice descritto nel Capitolo 3, in funzione del gene driver attualmente fissato dal ciclo.

Dopo aver tradotto i nomi dei geni appartenenti alla pathway in *EntrezID*, effettuando una **select** sul database ***hs***, e estratto gli *EntrezID* comuni tra il risultato ottenuto e il dataset *gse16837.d*, si è potuto procedere con l'assegnazione del *valore di mutazione/non-mutazione* ad ogni singolo gene trovato, preparando così il dataset finale su cui effettuare la fase di plotting.

```
 #Leggo la tabella con i record [gene - (mutazione/non-mutazione
)] relativa al gene driver e al pathway correnti
     genePath <- read.table(paste(directory, "input/", current_
gene, "_", current_pathway, "_out.txt", sep = ""),
                            header = FALSE,
                            sep = "\t")

     #Estraggo solo la colonna relativa ai nomi dei geni per
procedere alla traduzione in EntrezID
```

```
7         geneLabels <- as.vector(genePath[, 1])
8
9         #Effettuo la traduzione dal nome del gene al EntrezID del
    gene
10        geneEntrezID <- select(hs,
11                               keys = geneLabels,
12                               columns = c("ENTREZID"),
13                               keytype = "SYMBOL")
14
15        listEntrezID <- as.array(geneEntrezID$ENTREZID)
16        listDataBase <- as.array(rownames(gse16873.d))
17
18        #Estraggo i geni comuni tra il database e il gene corrente
    per evitare problemi con il plot
19        geneIntersection <- Reduce(intersect, list(listDataBase,
    listEntrezID))
20
21        #Estraggo il database finale su cui effettuare le modifiche
     per i colori
22        finalDataBase <- gse16873.d[geneIntersection, ]
23
24        #Se ho problemi con la fase di plotting la salto e passo al
     ciclo successivo
25        tryCatch({
26          #Imposto i colori ciclando sul database finale
27          for(k in 1:length(finalDataBase)) {
28            #Seleziono il label del gene relativo all'EntrezID
    corrente
29            entry <- rownames(finalDataBase)[k]
30
31            if(is.na(entry) || is.null(entry)) next
32
33            label <- select(hs,
34                            keys = entry,
35                            columns = c("SYMBOL"),
36                            keytype = "ENTREZID")
37
38            #Seleziono il valore da passare per la codifica del
    colore nel plot
39            val <- genePath[genePath$V1 == label$SYMBOL, ]$V2
40
41            #Assegno il valore trovato
42            finalDataBase[k, ] <- val
43          }
44          ...
45        })
```

4.2.3 Esecuzione del plotting

L'ultimo step è stato eseguire il plot delle pathway attraverso il metodo *pathview*.

Come mostra il codice sottostante sono stati utilizzati tutti i metodi mostrati nel paragrafo precedente; il valore che è stato passato in input al parametro *gene.data* è il dataset ***finalDataBase***, ottenuto mediante la manipolazione del dataset *gse16837.d* precedentemente effettuata.

Riguardo la colorazione dei geni si è scelto di evidenziare tutti i geni appartenenti alla pathway in funzione della mutazione del gene driver fissato. In particolare, i geni che non hanno subito delle mutazioni sono stati colorati in **verde** mentre quelli che ne hanno subite in **rosso**.

```
#Plot finale
pathview(gene.data = finalDataBase[, 1],
         pathway.id = substring(current_pathway, 4),
         species = "hsa",
         kegg.dir = "C:/Users/simon/Desktop/Tesi/3.PlottingPathview
    /kegg/",
         out.suffix = current_gene,
         discrete = list(gene = TRUE, cpd = TRUE),
         bins = list(gene = 2, cpd = 1),
         mid = list(gene = "#00FF00", cpd = "#00FF00"),
         high = list(gene = "#FF0000", cpd = "#00FF00"))
```

Qui di seguito si mostrano alcuni esempi di plot generati da *Pathview*.

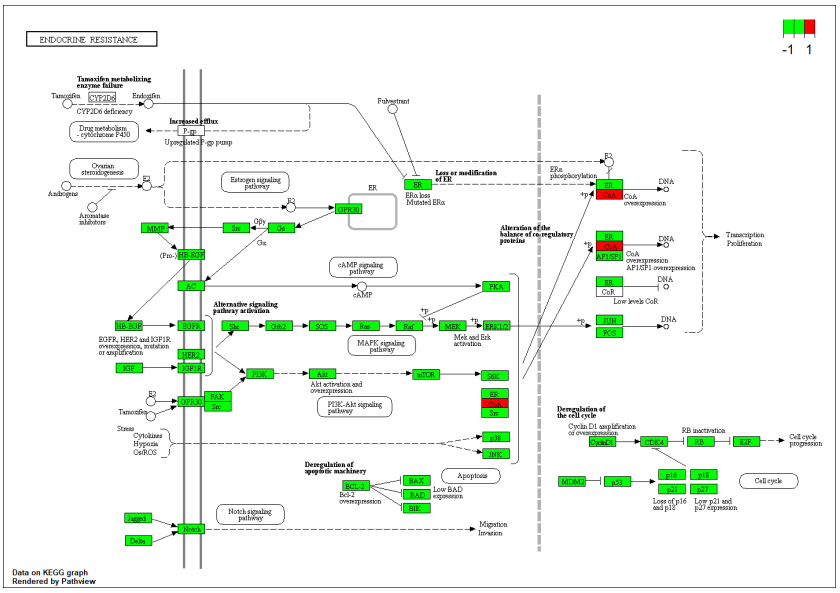

Figura 4.1: Plot della pathway hsa01522 fissata la mutazione del gene driver ERBB3

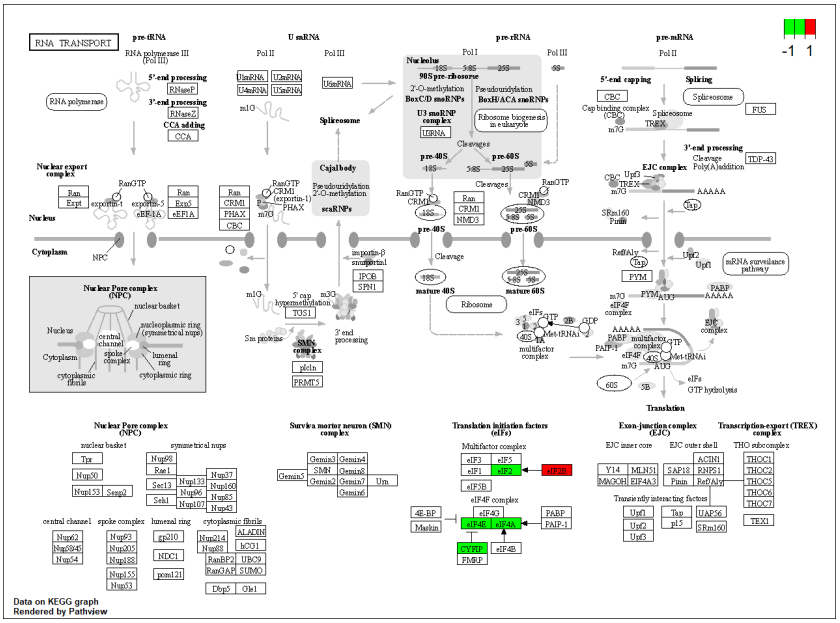

Figura 4.2: Plot della pathway hsa03013 fissata la mutazione del gene driver BRAF

5

Estrapolazione delle misure di centralità

In questo capitolo verranno illustrati i processi che hanno portato alla ricerca e l'estrapolazione degli **endpoint** nelle pathway prese in esame nel progetto e il calcolo delle misure di centralità che sono state descritte nel Capitolo 1.

I risultati ottenuti, combinati con i plot calcolati nel capitolo precedente, costituiscono i due elementi essenziali per procedere con la creazione dell'interfaccia web che sarà descritta nel Capitolo 6.

5.1 Preparazione e creazione dei file di input

Prima di iniziare con i passi descritti poc'anzi, è stato necessario effettuare una prima fase di preparazione e creazione dei file necessari per il raggiungimento degli obiettivi prefissati.

5.1.1 Estrazione dei nodi dalla MetaPathway

Il primo passo è stato manipolare il file relativo alla **MetaPathway** formato dai soli collegamenti tra i geni che la compongono; tali collegamenti possono essere di tipo ***activation*** oppure ***inhibition***. La MetaPathway rappresenta un enorme grafo che racchiude tutte le pathway che sono state prese in considerazione in questo progetto; essa ci permette di avere una visione più completa su come i geni delle

diverse pathway interagiscono tra loro.

Per permettere a R di leggerla sotto forma di grafo è necessaria però anche la lista dei nodi che la compongono, per questo motivo è stato scritto un semplicissimo script C++ che, prendendo in input la lista degli archi, ha estratto tutti i nodi che la costituiscono, generandone un file di testo in output.

```
#include <bits/stdc++.h>

using namespace std;

ifstream in("input/edgesMetaPathway.txt");
ofstream out("input/nodesMetaPathway.txt");

int main() {
  set<string> nodes;
  string tmp;

  while(!in.eof()) {
    in >> tmp;
    nodes.insert(tmp);
  }

  for(set<string>::iterator i = nodes.begin(); i != nodes.end(); ++
   i) {
    out << *i << endl;
  }
}
```

L'operazione appena descritta per la MetaPathay, è stata effettuata anche su tutte le altre pathway considerate nel progetto.

5.1.2 Ricerca degli endpoint sulle pathway

Il secondo step necessario è stato la ricerca degli **endpoint** nelle singole pathway.

Gli endpoint di una pathway sono tutti quei *nodi* (geni) i quali non hanno *archi uscenti*, ovvero non attivano o inibiscono altri geni della pathway. Potremmo pertanto considerarli come i prodotti finali delle pathway.

Al fine di estrarre quest'ultimi è stato applicato un semplicissimo algoritmo che, letto in input il file degli archi relativi ad una pathway, restituisce in output un file di testo con tutti gli endpoint che sono stati trovati.

L'algoritmo divide le coppie *gene di partenza - gene di arrivo* in due insiemi (**set**)

separati, ***starts*** e ***ends***; successivamente iterando su tutti i geni di arrivo controlla se viene trovata una corrispondenza nella lista dei geni di partenza. Se non viene trovata alcuna corrispondenza allora è stato trovato un endpoint, viceversa passa al gene di arrivo successivo.

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  ifstream pathways("input/PathwaysName.txt");

  while(!pathways.eof()) {
    string current_pathway;
    pathways >> current_pathway;

    cout << "Processing pathway " << current_pathway << "..." << endl;

    ifstream hsa(("input/Pathways/" + current_pathway + ".txt").c_str());
    ofstream out(("endpoints/" + current_pathway + "_endpoints.txt").c_str());

    set<string> starts, ends, endpoints;

    while(!hsa.eof()) {
      string tmp;

      hsa >> tmp;
      starts.insert(tmp);

      hsa >> tmp;
      ends.insert(tmp);

      hsa >> tmp;
    }

    for(set<string>::iterator i = ends.begin(); i != ends.end(); ++i) {
      set<string>::iterator j;
        for(j = starts.begin(); j != starts.end(); ++j) {
          if (*i == *j) break;
      }
      if(j == starts.end()) endpoints.insert(*i);
    }

    for(set<string>::iterator i = endpoints.begin(); i != endpoints.end(); ++i) {
```

```
40        out << *i << endl;
41      }
42
43    }
44
45    cout << "Operation complete... Output generated in endpoints
       directory." << endl;
46    return 0;
47  }
```

5.2 Calcolo delle misure di centralità

Dopo aver concluso la fase di preparazione dei dati, si è proceduto con il calcolo delle misure di centralità sui geni che compongono le pathway.

Il linguaggio di programmazione utilizzato è stato R, in quanto include nativamente la libreria ***iGraph*** che mette a disposizione una serie di strumenti e metodi che hanno permesso in tutta semplicità di effettuare le misure di centralità sulle pathway.

Dopo aver importato correttamente tutti i file di input, mediante delle iterazioni sui files delle pathway, è stato possibile effettuare il calcolo delle misure di centralità e riportare i risultati in output.

```
between_centrality <- betweenness(metaPathway,
                                  v = current_pathway_nodes)
  closness_centrality <- closeness(metaPathway,
                                   v = current_pathway_nodes)
  degree_centrality <- degree(metaPathway,
                              v = current_pathway_nodes)
  page_rank <- page.rank(metaPathway,
                         vids = current_pathway_endpoints)
```

Come si può notare dal codice sovrastante, solamente le prime tre misure di centralità, ovvero *betweennes*, *closeness* e *degree*, sono state applicate sulla pathway correntemente visitata dal ciclo; la ***page.rank***, invece, è stata applicata sulla MetaPathway e ai soli endpoints della pathway attualmente visitata. Questo è stato fatto per capire meglio l'importanza di un particolare gene nella rete, infatti un gene, che è un endpoint per una data pathway, potrebbe essere un punto di partenza di un'altra, e pertanto sarebbe stato riduttivo, se non inutile, vederne l'importanza

limitatamente alla pathway in cui è stato scoperto come endpoint, piuttosto diventa molto più interessante vederne l'importanza relativamente MetaPathway per capire meglio in che modo interagisce con gli altri geni.

6

Interfaccia di visualizzazione

Questo capitolo rappresenta il punto di arrivo di tutti gli step che sono stati effettuati nei capitoli precedenti, permettendo mediante un'interfaccia web di visualizzare i plot delle pathway e le tabelle contenenti i dati relativi alle misure di centralità precedentemente calcolate.

Prima di mostrare i passi fatti per la realizzazione dell'interfaccia web e il risultato finale, introduciamo brevemente il tool che ha permesso tutto ciò: **Shiny**.

6.1 Breve introduzione a Shiny

Shiny è un package di R che permette in maniera semplice di costruire web app interattive direttamente da R. Si possono ospitare app autonome su una pagina web o incorporarle in altri progetti di R. È possibile anche estendere le app di Shiny con temi *CSS*, *htmlwidgets* e azioni *JavaScript*.
Shiny riesce a combinare il potere computazionale di R con l'interattività del web moderno.

6.2 Realizzazione dell'interfaccia web

Per poter realizzare una web application con Shiny è necessario definire due strutture essenziali: ***ui*** e ***server***. La prima serve per definire la struttura grafica della nostra

applicazione, la seconda per definire le logiche del server cioè i "comportamenti" che l'applicazione deve rispettare a seguito di una determinata azione dell'utente.

6.2.1 Analisi della User Interface

In merito allo sviluppo della UI, è stata costruita una *sidebar* con il compito di permettere all'utente finale la selezione progressiva del *gene driver*, per il quale si vuole effettuare la ricerca, e successivamente la *pathway* che si vuole visualizzare.

Dopo aver cliccato sul pulsante ***Plot***, verranno visualizzati i plot e le tabelle relative alle misure di centralità della pathway che è stata selezionata, in funzione del gene driver scelto.

Di seguito si illustra il codice relativo allo sviluppo della UI che è stata descritta.

```
ui <- fluidPage(
  br(),

  titlePanel(
    h1("Association Rules for Genetic Mutations and their
    visualization on Biological Networks", align = "center")
  ),

  br(), br(),

  sidebarLayout(
    position = "right",

    sidebarPanel(
      selectInput(
        "gene",
        h4("Select a driver gene"),
        choices = gene_driver_list,
        selected = ""
      ),

      conditionalPanel(
        condition = "input.gene != ''",
        selectInput(
          "pathway",
          h4("Select a pathway"),
          choices = pathway_name_list,
          selected = ""
        )
      ),

      conditionalPanel(
```

```
32          condition = "input.pathway != ''",
33          actionButton(
34            "submit",
35            "Plot"
36          )
37        ),
38
39        width = 2
40      ),
41
42      mainPanel(
43        fluidRow(
44          column(1),
45
46          column(10,
47                 imageOutput("plot")
48                 ),
49
50          column(1)
51        ),
52
53        br(), br(), br(), br(), br(), br(), br(), br(), br(), br(),
    br(),
54        br(), br(), br(), br(), br(), br(), br(), br(), br(), br(),
    br(),
55        br(), br(), br(), br(), br(), br(), br(), br(), br(), br(),
    br(),
56
57        fluidRow(
58          column(7,
59            DT::dataTableOutput("centrality")
60          ),
61
62          column(1),
63
64          column(4,
65            DT::dataTableOutput("pagerank")
66          )
67        ),
68
69        br(), br(),
70
71        width = 10
72      )
73    )
74  )
```

6.2.2 Descrizione delle logiche del server

Il lato server dell'applicazione deve fondamentalmente raccogliere tutte le informazioni che gli vengono passate in input dalla UI e recuperare dalle directory, nelle quali sono salvati i plot e i dati relativi alle misure di centralità, i dati corretti da visualizzare in output.

Di seguito è mostrato il codice relativo alle logiche del server che sono state brevemente descritte.

```
server <- function(input, output) {
  observeEvent(input$submit, {
    output$plot <- renderImage({
      filename <- normalizePath(file.path(paste(directory, "output/
   plots/", input$pathway, ".", input$gene, ".png", sep = "")))

      list(src = filename,
           width = 1333,
           height = 972)
    })

    centrality <- read.table(paste(directory, "output/
   centralityValues/", input$pathway, "_centralityValues.txt", sep
   = ""),
                             col.names = c("gene", "betweenness", "
   closness", "degree"),
                             dec = ".",
                             sep = "\t")

    output$centrality <- DT::renderDataTable({centrality},
                                             server = F,
                                             rownames = F,
                                             selection = "single")

    pagerank <- read.table(paste(directory, "output/
   endpointsPageRank/", input$pathway, "_endpointsPageRank.txt",
   sep = ""),
                           col.names = c("endpoint", "page_rank"),
                           dec = ".",
                           sep = "\t")

    output$pagerank <- DT::renderDataTable({pagerank},
                                           server = F,
                                           rownames = F,
                                           selection = "single")
  })
}
```

6.3 Risultato finale

La web application che è stata costruita è estremamente semplice, ma possiede tutto ciò che è necessario per una completa visione dei dati che sono stati raccolti.

L'immagine sottostante mostra un esempio del risultato finale che è stato ottenuto.

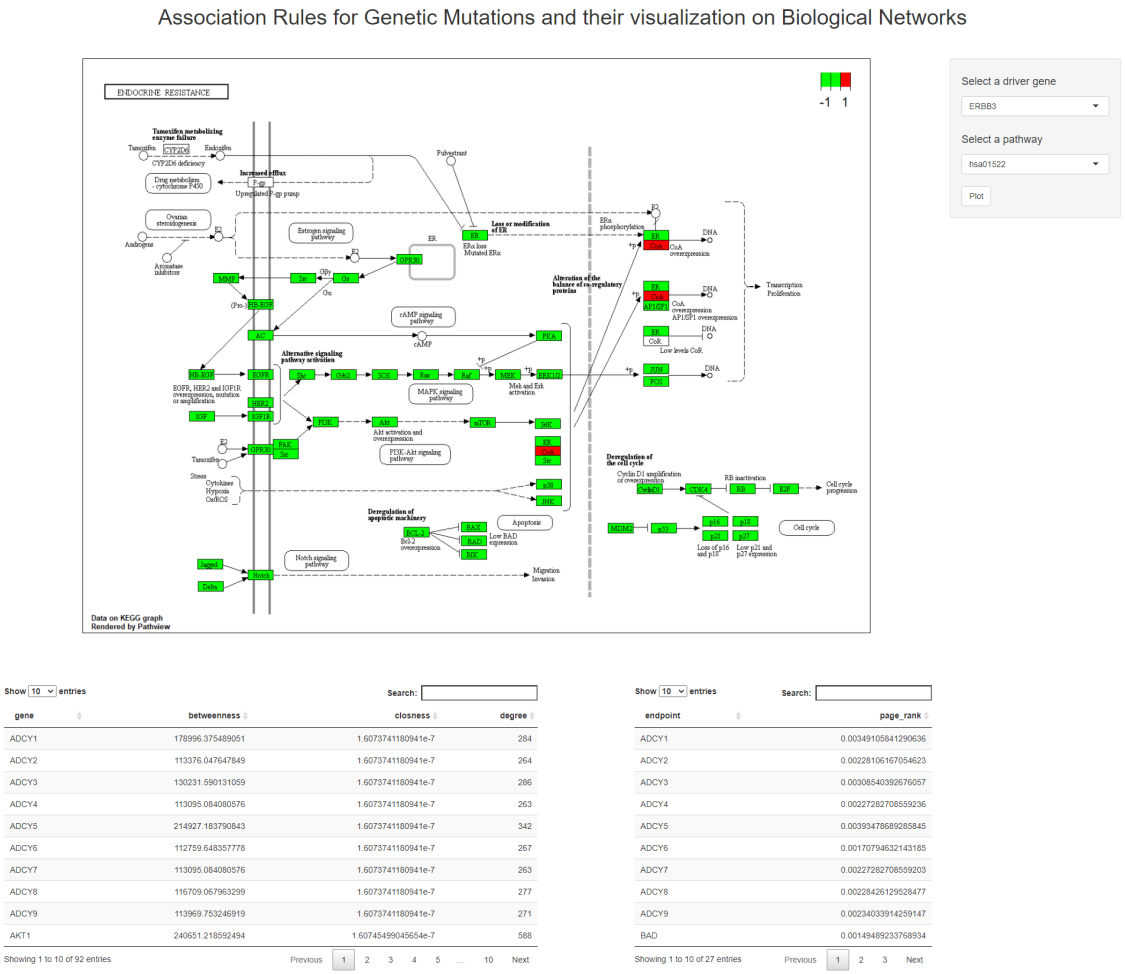

Figura 6.1: Interfaccia della web application che è stata sviluppata con Shiny

7

La pathway Breast Cancer (hsa05224)

In questo capitolo conclusivo verrà analizzata la pathway relativa al tumore al seno e verranno prese alcune considerazioni in merito ai risultati che sono stai ottenuti dal progetto.

7.1 Sottotipi molecolari di cancro al seno

L'obiettivo finale del progetto di tesi che è stato illustrato in questa trattazione era quello di sviluppare un'apposita interfaccia web capace di permettere la visualizzazione delle pathway e dei dati relativi a quest'ultime ponendo un focus sulla pathway hsa05224, ovvero quella relativa al **tumore al seno**.

Esistono principalmente 4 sottotipi intrinsechi o molecolari di cancro al seno che si basano sui geni espressi da un cancro:

- **Luminal A**, il cancro al seno è positivo al recettore dell'ormone, HER2 negativo, e ha bassi livelli della proteina Ki-67, che aiuta a controllare la velocità di crescita delle cellule tumorali. I tumori luminali A sono di basso grado, tendono a crescere lentamente e hanno la migliore prognosi.

- **Luminal B**, il cancro al seno è positivo al recettore dell'ormone, HER2 positivo o HER2 negativo con alti livelli di Ki-67. Generalmente cresce leggermente

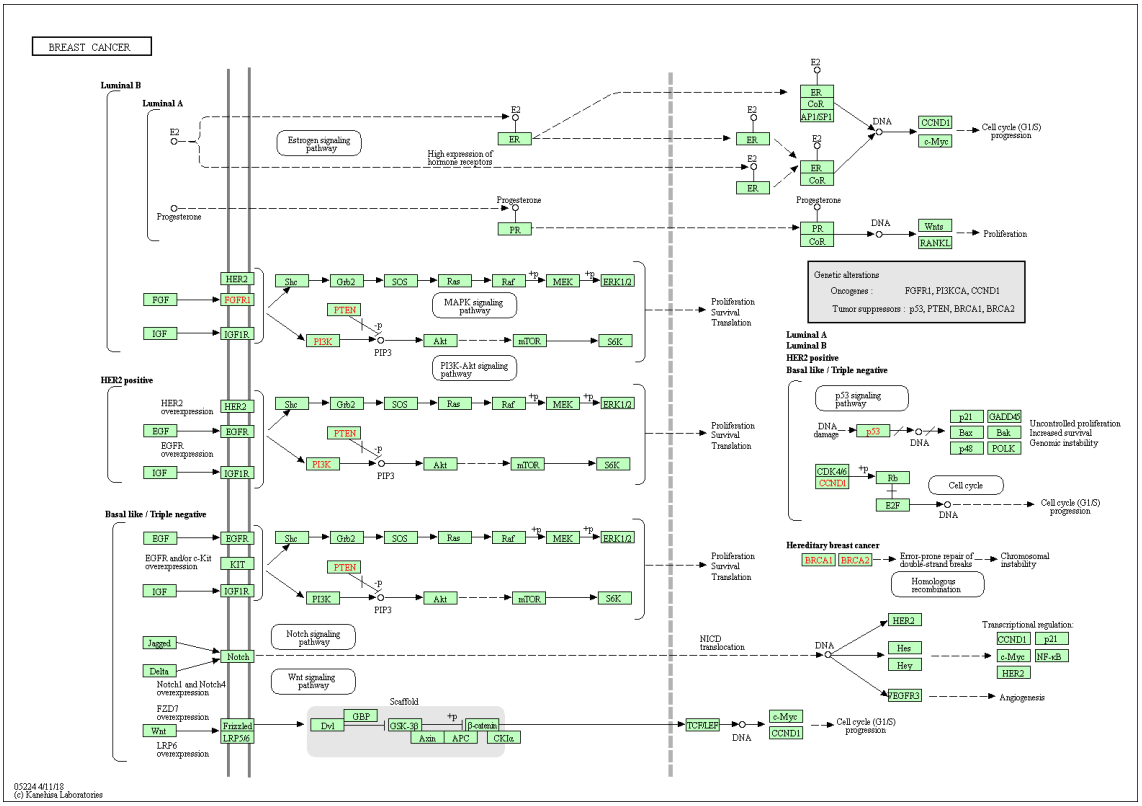

Figura 7.1: Pathway biologica relativa al tumore al seno (hsa05224)

in modo più veloce del tumore luminale A e la loro prognosi è leggermente peggiore.

- **Triple-negative/basal-like**, il cancro al seno è negativo per il recettore dell'ormone e per l'HER2 negativo. Questo tipo di cancro è più comune nelle donne con mutazioni del gene BRCA1. I ricercatori non sono sicuri del perché, ma questo tipo di cancro è molto più comune tra le donne giovani e di colore.

- **HER2-enriched**, il cancro al seno è negativo per il recettore dell'ormone e positivo per l'HER2. Tale sottotipo tende a crescere più velocemente dei cancri luminali precedenti e può avere una prognosi peggiore, ma è spesso trattato con successo con terapie mirate alla proteina HER2, come Enhertu, Herceptin, Perjeta, Tykerb, Nerlynx e Kadcyla.

7.2 Analisi dei risultati ottenuti

Analizzando alcuni dei risultati ottenuti nella pathway hsa05224, si può notare che la mutazione del gene driver **ABCB1** porti alla mutazione dei geni LRP5 e LRP6. Pertanto la mutazione di tali geni potrebbe potenzialmente essere causa di tumore al seno.

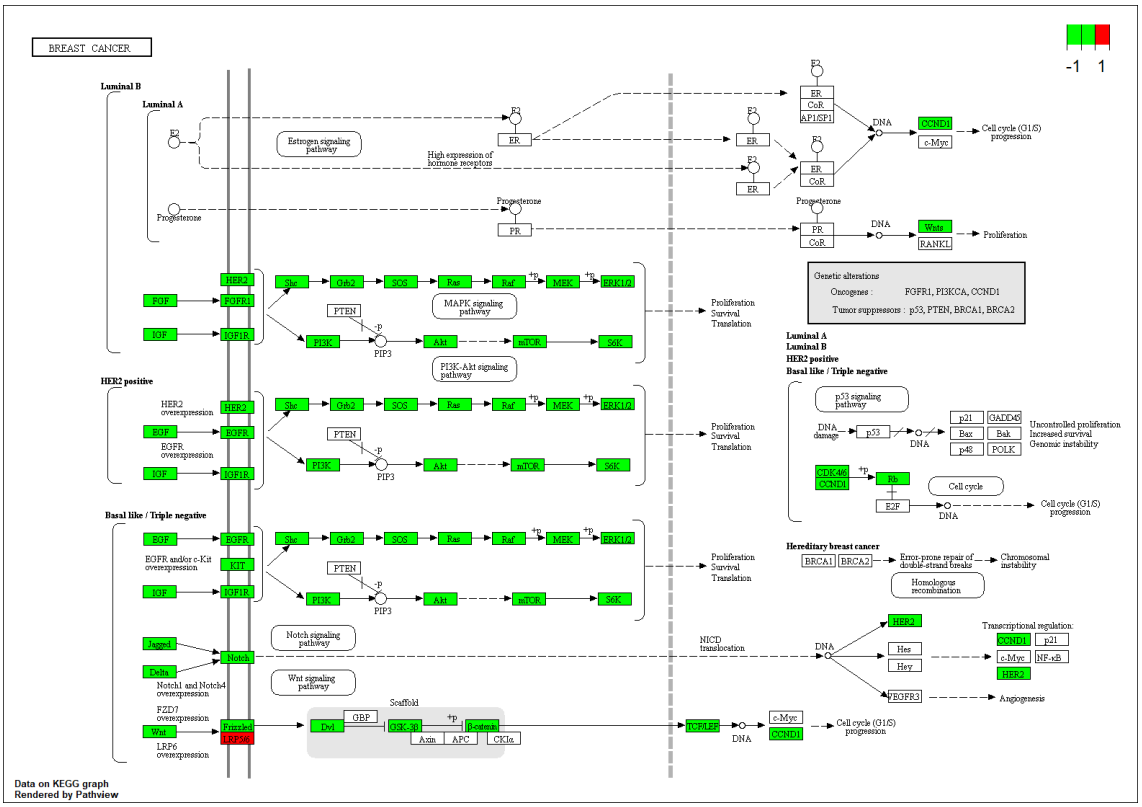

Figura 7.2: Plot della pathway hsa05224 fissata la mutazione del gene driver ABCB1

A sostegno di questo risultato è possibile citare l'articolo *LRP5/6 in Wnt signaling and tumorigenesis*. Secondo quanto riportato:

> *La **pathway di segnalazione Wnt** è spesso iperattivata in diversi tipi di cancro umano. Recenti studi hanno scoperto che la proteina correlata al recettore delle lipoproteine a bassa densità (LRP) 5 e 6 sono corecettori Wnt essenziali e interagiscono con diversi componenti chiave della via*

di segnalazione Wnt. Inoltre, è stato dimostrato che LRP5 e LRP6 sono potenziali proteine oncogene.

Un altro articolo che è possibile citare è *LRP5 regulates the expression of STK40, a new potential target in triple-negative breast cancers.*

Secondo quest'articolo i *triple-negative breast cancer* (TNBC) sono correlati con la sovraespressione dei geni LRP5 e LRP6, inoltre vi è una correlazione tra l'esaurimento del LRP5 e la deplezione del STK40, la quale riduce la vitalità cellulare e la formazione di colonie inducendo l'apoptosi delle cellule TNBC.

Conclusioni

Nonostante tutto ciò fosse nato come un progetto per la materia ***Introduzione al Data Mining***, questo progetto ha colpito così tanto il mio interesse al punto da diventare oggetto della mia tesi.

Dalla sua prima realizzazione, il progetto mostrato ha subìto una serie di aggiunte e migliorie: dal plotting mediante ***iGraph*** al più ordinato, ma complesso nella realizzazione, plotting mediante ***Pathview***; la realizzazione di un'interfaccia web mediante il tool ***Shiny***; la creazione di codici ad-hoc mediante **C++** e **R**... tutti questi sono stati piccoli passi che hanno permesso di migliorare il progetto originario fino ad arrivare a quello mostrato in questa trattazione.

Sicuramente il lavoro che è stato svolto pone le basi per ulteriori aggiornamenti futuri come la gestione di una maggiore mole di percorsi biologici fino ad arrivare a miglioramenti anche grafici nell'ambito dell'interfaccia web sviluppata. Può ricoprire anche un ruolo importante la possibilità di hostare il tutto all'interno di un server in modo tale da permettere la fruizione dei dati anche su Internet.

Ringraziamenti

Per quanto brevi, questi tre anni di università sono stati intensi. Nonostante i momenti di gioia e felicità, non sono mancati i momenti difficili ed è per questo che approfitto di quest'ultimo paragrafo per ringraziare tutti coloro che mi sono stati vicini.

Ai miei relatori che, grazie alla loro immensa disponibilità, mi hanno permesso di raggiungere questo obiettivo.

Ai miei colleghi di lavoro che, nella loro pazienza, mi hanno supportato e sopportato nei momenti critici di studio.

A tutti i miei amici, vecchi e nuovi, che sono stati essenziali per la mia crescita personale e con i quali ho condiviso bellissimi momenti.

Alle mie nonne, a mia zia e a tutti i miei parenti che hanno seguito, anche se in misure differenti, il percorso da me affrontato.

Alla mia **famiglia** per tutto il supporto, l'amore e il coraggio che mi avete sempre dato; rappresentate la colonna portante della mia vita.

A mio **padre** che, nonostante non sarà essere presente questo giorno, mi ha dato la forza e il coraggio di continuare. Tutto questo è dedicato a te!

Riferimenti bibliografici

1. Giovanni Micale, unreleased work, DMGS Finder, Catania, Italy, 2019
2. Jeremy Sapienza, *Ego-network genetiche di co-espressione e co-mutazione in pazienti di tumore al seno*, Catania, 2020
3. iGraph, *Library open source for R - iGraph*
4. Shiny, *Library open source for R - Shiny*
5. Weijun Luo, *Pathview: pathway based data integration and visualization*, r-forge.r-project, 2013
6. Giorgio Ausiello, Fabrizio d'Amore, Giorgio Gambosi, *Linguaggi, Modelli, Complessità*, Franco Angeli, 2014
7. Wikipedia, *Biologia - Wikipedia, L'enciclopedia libera*, Wikipedia, 2020
8. Wikipedia, *Biomedicina - Wikipedia, L'enciclopedia libera*, Wikipedia, 2020
9. Wikipedia, *DNA - Wikipedia, L'enciclopedia libera*, Wikipedia, 2020
10. Wikipedia, *Gene - Wikipedia, L'enciclopedia libera*, Wikipedia, 2020
11. chimica-online.it, *Cellula*, chimica-online.it
12. AIRC, *Tumore del seno*, AIRC, 2018
13. BreastCancer.org, *Molecular Subtypes of Breast Cancer - Breastcancer.org*
14. Yonghe Li, Guojun Bu, *LRP5/6 in Wnt signaling and tumorigenesis*, Future Medicine, 2005
15. Sylvie Maubant, Tania Tahtouh, *LRP5 regulates the expression of STK40, a new potential target in triple-negative breast cancers*, Oncotarget, 2018