UNIVERSITÀ DEGLI STUDI DI CATANIA

FACOLTÀ DI MEDICINA E CHIRURGIA

DOTTORATO IN BIOLOGIA, GENETICA UMANA E BIOINFORMATICA: BASI CELLULARI E MOLECOLARI DEL FENOTIPO-XXII CICLO

DIPARTIMENTO DI SCIENZE BIOMEDICHE

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Dott. Raffaele Di Natale

ALGORITMI DI GRAPH QUERYING PER LA RICERCA DI SOTTOSTRUTTURE IN GRANDI RETI BIOLOGICHE

Tesi di Dottorato

Coordinatore:
Chiar.mo Prof. MICHELE PURRELLO
Tutor:
Chiar.mo Prof. ALFREDO FERRO

Anno Accademico 2010 / 2011

Indice

Elenco delle figure

Elenco delle tabelle

ELENCO DELLE TABELLE

1

Introduzione

La ricerca nell'ambito del graph querying sulle grandi reti biologiche ha ovviamente origine nella più generica ricerca sui grandi grafi. In effetti, sono prima di tutte la bioinformatica e le scienze sociali o umane a fornire gli spunti (o problemi) più interessanti e complessi sui quali la comunità scientifica si è maggiormente concentrata. Ciò che ha contraddistinto la ricerca in questi settori è stata la considerazione che la grande mole di dati rappresentata mediante grafi avrebbe potuto rappresentare realmente uno strumento di scoperta ed innovazione se e solo fosse stato possibile sviluppare dei tool specifici in grado di interagire con tali dati in maniera performante. Il problema di base, di per sé semplice ed intuitivo, è quello di individuare tutti i grafi che contengano o siano contenuti, per intero o parzialmente, in un grafo query . Nel tempo sono state individuate tre possibili tipologie di ricerca o querying su database di grafi: 1) Ricerca esatta di sottografi (subgraph querying); 2) Ricerca esatta di supergrafi (supergraph querying); 3) Ricerca per similarità o inesatta (inexact querying). Nella ricerca di sottografi il problema affrontato è quello di individuare tutti i grafi del database che contengono un grafo query. Questa tipologia di interrogazione trova applicazione in svariati ambiti come ad esempio l'analisi dell'interazione tra proteine, la progettazione di nuovi farmaci, la correlazione fra composti molecolari. Nella ricerca di supergrafi si vogliono individuare, invece, tutti i grafi di un database che sono contenuti all'interno di un grafo query. L'ambito principale di applicazione è senz'altro il riconoscimento di oggetti, ma anche l'identificazione di descrittori chimici. Nell'ultima tipologia di ricerca, ovvero la ricerca per similarità, si parte dal presupposto che i grafi trattati non siano esenti da errore. Quindi piuttosto che tentare di individuare solo sottostrutture che combacino esattamente con quella cercata, si estende la ricerca anche a sottostrutture approssimate, poiché potrebbe essere molto probabile che le eventuali lievi differenze

siano dovute ad un errore sperimentale e non ad una differenza reale, garantendo in tal modo la possibilità di effettuare, ad esempio, un confronto tra specie differenti o molecole simili. Nei capitoli seguenti saranno affrontati il primo ed il terzo tipo di ricerca, in quanto molto più legati all'ambito biologico e per i quali saranno proposte soluzioni innovative.

1.1 Lo stato dell'arte dell'ambito del Graph Querying

Per le tre tipologie di ricerca o querying su database di grafi possiamo distribuire i tool attualmente più diffusi in base alla Tabella 1.1.

Tabella 1.1: Stato dell'arte nel Graph Querying

Tipo di Query	Struttura Completa	Sottografi	Supergrafi
Esatta		GraphGrep (7) gIndex (2) CTreee (3) FG-Index (10) GString (11) Tree+δ (12) GCoding (4) QuickSI (13)	cIndex (8) GPTree (9)
Inesatta	RASCAL (14)	Grafil (15) SAGA (17)	SG-Enum (16)

1.2 I limiti del problema

L'idea di ricercare il grafo query in maniera esaustiva su tutti i grafi del database è apparsa da subito computazionalmente costosa se non addirittura impraticabile e pertanto tutti gli sforzi si sono rivolti nell'individuare una strategia ottimale che consentisse di ottenere delle indicazioni più o meno granulari sull'effettiva presenza della query all'interno di uno o più grafi contenuti all'interno del database. Inizialmente, le soluzioni proposte erano pensate per grafi di piccole dimensioni e la loro applicazione su grafi reali, composti in molti casi da migliaia di archi e migliaia di nodi risultava impraticabile se non addirittura impossibile. Dovendo effettuare una ricerca in grafi di grandissima dimensione risulta essenziale disporre di un meccanismo capace di scartare

con sicurezza ed in tempi rapidissimi i grafi che non possono contenere (o essere contenuti) nel grafo query, spendendo altresì risorse computazionali per affinare la ricerca nei restanti grafi.

1.3 Le risposte fornite dai tool proposti

I tool qui proposti, superano gli ostacoli precedentemente descritti nei rispettivi problemi affrontati (subgraph querying esatto e subgraph querying inesatto), cercando di fornire alla comunità scientifica degli strumenti realmente utilizzabili in situazioni reali. Per questo motivo, partendo dalla ricerca esatta, per la quale è stato sviluppato un innovativo algoritmo, si è giunti alla realizzazione di una altrettanto innovativa tecnica nella ricerca inesatta che trova probabilmente più applicazioni in un contesto quale quello biologico.

2

Preliminari

2.1 Teoria e terminologia dei grafi

Si definisce *grafo* un insieme di punti e di linee che connettono coppie di punti. Tali punti assumono il nome di *nodi* o *vertici*, mentre le linee sono chiamate *archi* o *frecce*. Normalmente un grafo è rappresentato come $G(V,E)$, dove G è il nome del grafo, V il suo insieme di nodi, E di archi, che può essere espresso come $E \subseteq V \times V$. Si usa rappresentare il numero di nodi come n ed il numero di archi come e che corrispondono rispettivamente a $|V|$ e $|E|$. Con $V(G)$ indichiamo l'insieme di nodi del grafo G e con $E(G)$ l'insieme degli archi di G. Si dice che un grafo è *completo* quando esiste un arco per ogni coppia di nodi; in questo caso si parla anche di *clique*. Un arco si dice *diretto* o *orientato* quando, quando E altrimenti si definisce indiretto o *non orientato*. Un grafo si dice *indiretto* quando l'insieme degli archi E è costituito da coppie non ordinate di vertici, viceversa si dirà *diretto*. Nel primo caso gli archi si diranno *indiretti*, nel secondo *diretti* I nodi connessi mediante un arco si dicono *adiacenti*, mentre un *vicino* di un nodo v è un nodo adiacente a v. Pertanto, denotiamo con $N(v)$ l'insieme dei vicini di un nodo v. Per *grado* di un nodo si intende il numero di archi che incidono su quel nodo. Nel caso in cui il grafo sia diretto si definisce il grado entrante di un nodo (il numero di archi che arrivano a quel nodo) ed il grado uscente (il numero di archi che partono da quel nodo). Un *path* o *cammino* è rappresentato da una sequenza di nodi ed archi in maniera che un nodo appartiene all'arco che lo precede e lo segue nella sequenza e non sono presenti nodi ripetuti. Un path formato da k nodi è definito come P_k. La *length* o *lunghezza* di un path è costituita dal numero di archi che compongono il path. Un *subgraph* o *sottografo* di G è un grafo nel quale tutti i nodi e gli archi appartengono a G. Un *induced subgraph* o *sottografo indotto* H di G è un sottografo di G sui nodi

$V(H)$, tale che $E(H)$ è formato da tutti gli archi di G che connettono nodi di $V(H)$.

2.1.1 Isomorfismo tra grafi

Definiamo $g = (V, E, \Sigma, l)$ *grafo indiretto etichettato* in cui V rappresenta l'insieme dei vertici, $E \subseteq V \times V$ l'insieme degli archi, Σ l'alfabeto delle etichette e $l : V \rightarrow \Sigma$ la una funzione che assegna ad ogni vertice del grafo una etichetta dell'alfabeto Σ. Se $e = (v_1, v_2)$ è un arco, allora v_1 e v_2 sono definiti i suoi *estremi*. Definiamo $dimensione(g) = |E|$ ed indichiamo con $\mathcal{G}$ l'insieme di tutti i possibili grafi. Un grafo $g_1 = (V_1, E_1, \Sigma, l_1)$ è definito *sottografo* di un secondo grafo $g_2 = (V_2, E_2, \Sigma, l_2)$ se e solo se $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. Dati due grafi $g_1 = (V_1, E_1, \Sigma, l_1)$, $g_2 = (V_2, E_2, \Sigma, l_2)$, un *isomorfismo* o *isomorphism* tra g_1 e g_2 è una funzione $\phi : V_1 \rightarrow V_2$ tale che:

- $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$
- $l_1(u) = l_2(f(u)) \forall u \in V_1$

Pertanto, possiamo affermare che due grafi sono detti isomorfi se esiste un isomorfismo tra di loro.

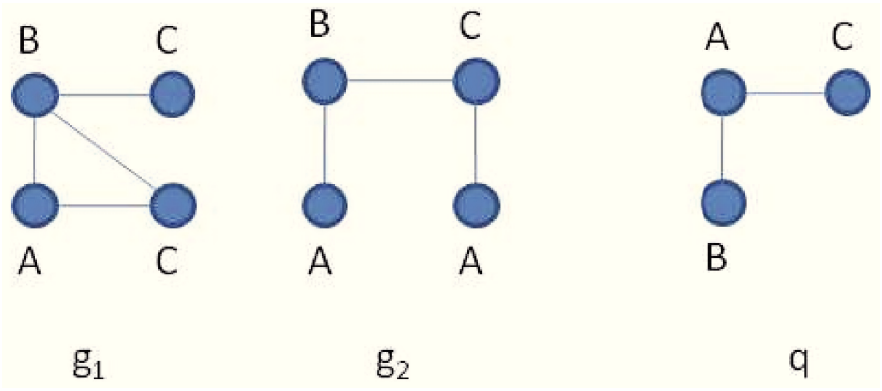


Figura 2.1: Esempio di database formato da due grafi g_1, g_2 a da un grafo query q. - q è isomorfo ad un sottografo di g_1, ma non di g_2.

Non esistono algoritmi che risolvono il problema dell'isomorfismo tra grafi in tempo polinomiale (19), ciò comporta che, nel caso peggiore, la complessità computazionale degli algoritmi disponibili risulta esponenziale rispetto al numero di nodi dei due grafi. In pratica la presenza e la tipologia delle etichette ai nodi e/o agli archi consente di ridurre sensibilmente la complessità della ricerca rendendo in tal modo il problema gestibile dal punto di vista computazionale. Sfruttando le caratteristiche proprie di

particolari tipologie di grafi, è stato possibile definire degli algoritmi che risolvono il problema dell'isomorfismo tra grafi in tempo addirittura polinomiale. In genere, le gli algoritmi utilizzati per verificare che due grafi siano isomorfi o meno si basano essenzialmente su tecniche di ricerca in profondità con strutture dati di tipo albero ed il supporto del *backtracking* (5) . Questi algoritmi iniziano individuando un match parziale, assegnando nodi del primo grafo a quelli del secondo, che poi è iterativamente esteso aggiungendo nuove corrispondenze nodo-nodo. L'inserimento di nuove coppie di nodi compatibili si arresta non appena risulta incompatibile la struttura topologica ovvero l'etichetta ai nodi o agli archi o ulteriori parametri di compatibilità tra nodi definiti dall'utente. Non appena rilevata l'incompatibilità, l'algoritmo avvia una procedura di backtracking mediante la quale, iterativamente, l'ultima coppia di nodi aggiunta al match parziale viene rimossa fino a quando non viene individuata una ulteriore possibile alternativa. Se per caso non risulta possibile estendere ulteriormente il match parziale, l'algoritmo terminerà restituendo in output che i due grafi risultano non isomorfo; in caso contrario sono ovviamente isomorfi. Un algoritmo più recente e molto simile al precedente è VF2 (5) che è stato ampiamente utilizzato per le sue caratteristiche di efficienza negli gli algoritmi di graph querying qui proposti. In effetti il problema pratico più interessante non soltanto nell'ambito della Bioinformatica è l'isomorfismo tra sottografi o *subgraph isomorphism*, strettamente connesso al problema dell'isomorfismo tra grafi. Un'interessante variante dell'isomorfismo tra grafi è rappresentata dall'isomorfismo tra sottografi (dei quali il primo è più piccolo del secondo), che quanto non richiede l'esatta corrispondenza tra il primo grafo ed il secondo grafo, ma solo l'isomorfismo tra il primo grafo ed un qualsiasi sottografo del secondo grafo. Un esempio di isomorfismo tra sottografi è illustrato in Figura 2.1. Nel caso descritto, il grafo q è un sottografo isomorfo di $g1$ ma non di $g2$. Da un punto di vista formale possiamo dire quindi che un *isomorfismo tra sottografi* tra g_1 e g_2 è un isomorfismo tra g_1 e un sottografo di g_2. Si dice che il grafo g_1 sia isomorfo ad un secondo grafo g_2 se è presente un isomorfismo tra g_1 e g_2. Semplificando possiamo dire che g_1 è equivalente a g_2 denotando questa relazione come $g_1 \approx g_2$. Notiamo che $\approx$ costituisce una relazione di equivalenza su $\mathcal{G}$. Un grafo g_1 è definito come sottografo isomorfo di un secondo grafo g_2 se è presente un subgraph isomorphism tra g_1 e g_2. In tal caso diremo che g_1 è *contenuto* in g_2 e scriviamo $g_1 \precsim g_2$. Nell'esempio illustrato in Figura 2.1 sono presenti due grafi denominati g_1, g_2 ed un grafo q. In questo caso $q \precsim g_1$ ma $q \not\precsim g_2$ Il concetto di isomorfismo e di conseguenza anche quello di isomorfismo tra sottografi può essere esteso facilmente anche a grafi diretti e/o etichettati agli archi. In conclusione possiamo dire pur essendo il *subgraph querying esatto* il primo problema in ordine di tempo

affrontato, nel tempo si è cominciato ad affrontare un problema che probabilmente, in molteplici ambiti compreso quello biologico, risulta più utile: il *subgraph querying inesatto*.

2.2 Reti *Scale Free*

Una delle proprietà di un grafo che possono essere misurate direttamente è rappresentata dal grado della distribuzione, ovvero la porzione di nodi $P(k)$ che possiedono k connessioni (o che hanno un grado pari a k). Una rete *scale free* o *ad invarianza di scala* è un grafo per il quale P(k) $\sim ck^{-}\lambda$, dove c rappresenta un fattore di normalizzazione e λ varia nel range $2 < \lambda < 3$ In queste reti sono presenti alcuni nodi, denominati *hub*, caratterizzati da un grado molto più grande del grado medio riscontrato nella rete. Alcuni esempi di rete scale free sono: internet, le social network e le network biologiche.

3

Le reti biologiche

3.1 Terminologia biologica

Le proteine costituiscono un elemento fondamentale nelle attività del cellula. Svolgono molteplici funzioni che vanno dal trasferimento dei segnali al controllo degli enzimi. Per raggiungere questi obiettivi le proteine interagiscono tra di loro, con il DNA e con altre molecole. Una particolarità di queste interazione è rappresentata dal fatto che esistono interazioni persistenti e temporanee (transienti). Inoltre, esistono gruppi di proteine che eseguono una particolare attività cellulare e per questo motivo sono chiamati *protein complex* o *complessi proteici*. Una proteina può essere costituita da uno o più *domini*. I domini sono definiti come una catena polipeptidica o parte di essa che si ripiega indipendentemente in una struttura stabile. I domini costituiscono unità funzionali e spesso a domini diversi di una proteina sono associate funzioni diverse. Esempi di tali funzioni possono essere la crescita della cellula, la trasduzione del segnale, la trascrizione, etc. La *Gene Ontology* (20) si occupa di descrivere in maniera sistematica tali funzioni. L'aspetto interessante da un punto di vista computazionale è che esiste una relazione *molti-a-molti* tra proteine e domini, cosi che una proteina può essere caratterizzata da differenti domini, mentre uno specifico dominio può essere presente in più proteine.

3.2 Introduzione alle reti biologiche

Con il termine di rete biologica ci si riferisce ad un insieme eterogeneo di elementi modellati direttamente o indirettamente attraverso dati sperimentali. Alla prima categoria appartengono: le reti di regolazione, le reti di interazione proteina-proteina,

le reti metaboliche, le reti di trasduzione del segnale. Alla seconda: le reti di correlazione e le reti filogenetiche. Le reti di regolazione della trascrizione si occupano di controllare l'espressione genica all'interno della cellula. Sono rappresentate mediante un grafo diretto, in cui il nodo descrive il gene e l'arco il controllo che un gene applica al nodo/gene regolato. Una particolarità riscontrata in questo tipo di reti è costituita dalla presenza di particolari sotto-strutture che prendono il nome di *Motif* e che saranno descritte in seguito perché oggetto di specifiche applicazioni mediante il tool denominato SING (21) alla sezione 7.4. Le reti di interazione proteina-proteina sono realizzate utilizzando molteplici approcci sperimentali e computazionali. Questo significa che per una data specie non è corretto parlare semplicemente di rete di interazione proteina-proteina senza associarla al particolare metodo utilizzato. Le reti metaboliche sono costituite da un insieme di elementi e rispettive relazioni, tra cui: metaboliti, reazioni biochimiche, enzimi, geni e cofattori. In questo contesto, uno degli aspetti chiave delle ricerca è costituito dallo studio e dalla predizione dei *pathway* metabolici (22). Le reti di trasduzione del segnale possono essere viste come delle reti di regolazione della trascrizione con catene di segnali che comprendono diverse tipologie di nodi ed in cui gli archi possono essere interazioni proteina-proteina e fosforilasi. Di recente sono stati presentati diversi lavori su *graph mining* nell'ambito delle reti di trasduzione del segnale (23). Le reti filogenetiche descrivono le relazioni evolutive tra diversi organismi. In effetti, sarebbe meglio parlare di alberi filogenetici, visto che la struttura utilizzata è un albero binario, in cui ogni ramificazione all'interno dell'albero rappresenta la separazione evolutiva tra due specie. In ogni caso, pur essendo alberi binari possono sempre essere manipolati come reti e pertanto su di essi possono essere applicati gli algoritmi di graph querying alla pari delle altre tipologie di reti (24).

3.3 Database di reti biologiche

La ricerca nell'ambito della Bioinformatica si è avvalsa della collaborazioni di molteplici tecnologie computazionali e, com'è accaduto in altre aree della Bioinformatica, anche per le reti biologiche sono stati prodotti nel tempo ingenti quantitativi di dati strutturati e non, che hanno consentito di aggiungere nuovi tasselli nella costruzione del grande mosaico dei processi cellulari. In modo particolare, le reti biologiche non sono semplicemente un agglomerato di oggetti, come possono essere i composti biochimici, le proteine e il materiale genetico, ma anche e soprattutto caratterizzate dalle relazioni esistenti tra gli elementi di cui sono composti. Pertanto, i database relativi alle reti biologiche presentano molteplici riferimenti incrociati con altri database, i qua-

li possiedono informazioni circa i composti chimici, le proteine ed i materiali genetici. Bisogna però osservare che la maggior parte di questi database, alcuni dei quali sono forniti come esempio in in Tabella 3.1, pur essendo continuamente aggiornati non sono completamente esaustivi e molto spesso è necessario estrarre ulteriori informazioni provenienti da altri database per riuscire ad avere un quadro completo e specifico della rete che si sta studiando. Inoltre, questi database presentano e distribuiscono queste reti mediante differenti formati. Ciò comporta che nella maggior parte dei casi per poter operare con dati provenienti da differenti database è necessario un lavoro preliminare per uniformarne al rappresentazione con la possibilità di introdurre degli errori e la necessità di dover ripetere questa procedura ad ogni aggiornamento delle reti utilizzate.

Tabella 3.1: Database di Reti Biologiche

Database	URL
KEGG	`http://www.genome.jp/kegg/pathway.html`
PathCase	`http://nashua.cwru.edu/pathways`
BIND	`http://bind.ca`
DIP	`http://dip.doe-mbi.ucla.edu`
BioCyc	`http://www.biocyc.org`

3.4 Obiettivi computazionali nell'ambito delle reti biologiche

Una rappresentazione efficiente di una rete biologica rappresenta la base per una corretta analisi computazionale della stessa. Come abbiamo anticipato in precedenza, la modellazione di una rete biologica mediante un grafo riesce a sintetizzare in maniera efficiente le entità e le rispettive relazioni all'interno di una rete biologica ed è pertanto divenuta in breve tempo uno standard *de facto*. Una rappresentazione di grafi mediante DBMS (Data base management system) è supportata dai moderni RDBMS (Relational DBMS), mediante i quali all'efficienza della rappresentazione si aggiunge il valore aggiunto delle funzionalità del DBMS e del linguaggio SQL. Un esempio di implementazione di questa soluzione orientata alle reti è quello fornito da Stephens et al.(25). L'utilizzo del database KEGG PATHWAY (22) rappresenta un'ottima opportunità per individuare in tempi rapidissimi i *pattern* frequenti delle reti supportate. Un ulteriore esempio di possibile obiettivo computazionale basato sulle reti biologiche è rappresentato dalla individuazione di porzioni di reti di specie diverse che si sono conservate nel

tempo (26). L'ultimo esempio, che rappresenta l'obiettivo primario degli algoritmi che saranno trattati in seguito, è costituito dalla ricerca di particolari sottostrutture all'interno delle grandi reti biologiche ovvero all'interno di grandi database di reti biologiche (21, 27). Per completezza di informazione, dobbiamo aggiungere che la rappresentazione basata su grafi non è la sola utilizzata nell'ambito dell'analisi computazionale delle reti biologiche. Infatti, a questa vanno aggiunte ad esempio:

- la rappresentazione delle reti di pathway mediante notazione grafica e diagramma di processo (28);
- circuiti logici per descrivere le reti di regolazione della trascrizione (29);
- rappresentazioni di espressioni geniche e polimorfismi genici(30).

3.5 Applicazione degli algoritmi di graph querying alle reti di trascrizione della regolazione e alle reti di interazione proteina-proteina

I due sistemi di graph querying descritti nei capitoli successivi sono stati principalmente applicati a due tipologie di reti biologiche: le reti di trascrizione della regolazione e le reti di interazione proteina-proteina e per tale motivo, nei capitoli successivi saranno descritte evidenziando gli obiettivi computazionali affrontati. I sistemi di graph querying esatto ed inesatto, possono tuttavia essere applicati sotto particolari precondizioni anche ad altre reti biologiche. I motivi per i quali ci si è focalizzati su queste due tipologie di reti sono molteplici. Per quanto riguarda le reti di interazione proteina-proteina, il confronto tra complessi proteici di specie diverse rappresenta ancora adesso un problema aperto nell'ottica del riconoscimento sia dei complessi sia delle relative funzioni cellulari. Inoltre, le notevoli dimensioni di tali reti hanno consentito di verificare la scalabilità dei sistemi proposti anche mediante il confronto con i tool che rappresentano lo stato dell'arte nello specifico contesto applicativo. Analogamente per le reti di regolazione della trascrizione, il lavoro presentato da (6) ci ha fornito diversi spunti di riflessione in merito ai Motif e soprattutto sulla capacità dei moderni sistemi di individuarli all'interno delle reti di trascrizione.

4

Reti di interazione proteina proteina

4.1 Introduzione

Tra i più recenti e interessanti sviluppi nell'ambito della proteomica si colloca sicuramente lo studio delle interazioni proteine-molecole. Tali ricerche si basano sull'assunto che l'interpretazione delle possibili relazioni tra genoma e proteoma possa consentire la comprensione delle malattie ad esse correlate e quindi mettere a punto nuove terapie per la loro cura. Dagli inizi, l'identificazione delle interazioni tra proteina e proteina ha rappresentato un problema tutt'altro che banale, sia dal punto di vista biochimico sia da quello computazionale. Nonostante la scarsa accuratezza della maggior parte dei metodi sviluppati, col tempo c'è stata un'attenzione sempre maggiore verso tali reti di interazione. Il modello che da subito si è affermato come uno standard de facto per la rappresentazioni delle interazioni delle proteine è quello basato su grafi, in cui i nodi rappresentano le proteine e gli archi le *possibili* interazioni tra di esse. Parliamo di possibili interazioni perché in effetti si riscontrano molti falsi negativi dovuti ai limiti dell'approccio utilizzato, ma anche molti falsi positivi per via del fatto che due proteine che interagiscono soltanto indirettamente, spesso sono rilevate come direttamente interagenti. Come accennato in precedenza, quando si descrive la rete di interazione proteina-proteina di una particolare specie è più corretto associarla al particolare approccio sperimentale/computazionale che l'ha prodotto: infatti, approcci differenti producono reti differenti(31). Ciò spiega le notevoli differenze nelle rappresentazioni di tali reti anche all'interno di una stessa specie, sia per numero di nodi/proteine sia per numero di archi/interazioni. A questo va aggiunto che in tali modelli di rappresentazio-

ne non sono contemplate informazioni fondamentali come quelle temporali, spaziali e le condizioni sperimentali. Sotto questo punto di vista un passaggio fondamentale è rappresentato dall'introduzione del *livello di confidenza*, illustrato in (31) che ci permette di riconoscere quanto *vera* possa essere una data interazione.

4.2 Identificazione delle reti di interazione proteina-proteina

Le due tecniche più usate nel passato per l'identificazione delle interazioni proteina-proteina sono state la cromatografia di affinità (32) e la coimmunoprecipitazione (33). La qualità e la quantità del materiale a disposizione per questi due approcci sono di fondamentale importanza per ottenere risultati più o meno alterati da errori. La Co-immunoprocipitazione (Co-IP) rappresenta una tecnica molto popolare mediante la quale è possibile identificare le interazioni proteina-proteina fisiologicamente rilevanti utilizzando anticorpi specifici della proteina target per catturare indirettamente le proteine che sono legate ad essa. Tali complessi proteici possono essere analizzati per identificare nuove affinità tra proteine nonché la funzione della proteina target. La purificazione tramite cromatografia d'affinità, inizialmente sviluppata per la purificazione degli enzimi, si basa sulle interazioni altamente specifiche delle molecole biologiche che si vogliono separare, piuttosto che sulle differenze nelle proprietà fisiche di tali molecole. Pertanto, potrebbe raggiungere teoricamente una purificazione completa in una singola tappa, anche partendo da miscele complesse. La prima applicazione di tale metodologia è avvenuta sugli enzimi e successivamente è stata utilizzata su altri composti, quali ad esempio acidi nucleici o immunoglobuline. Essa prevede che il composto da purificare si leghi reversibilmente a un ligando specifico, immobilizzato su una matrice insolubile. Per tale ragione, al fine della sua applicazione è necessario conoscere in maniera dettagliata la struttura e le specificità del composto da purificare. Solo così sarà possibile allestire le condizioni di separazione che consentano di ottenere la resa più elevata. E' necessario sottolineare, però, che soltanto grazie alla recente affermazione dei metodi basati sulla spettrometria di massa è stato possibile ottenere finalmente informazioni molto più precise sulle interazioni proteina-proteina e quindi sull'identificazione di complessi proteici. Attualmente sono due gli approcci utilizzati al fine di ottenere una mappa delle interazioni proteiche ad alta produttività. Il primo metodo è denominato *yeast two-hybrid* mentre il secondo, *AP-MS*, è basato sull'analisi dei complessi proteici mediate *affinity purification* accoppiata alla spettrometria di massa (34, 35, 36, 37). Possiamo affermare che i due metodi pur differendosi per modalità e obiettivi sono assolutamente complementari. Infatti, il primo è caratterizzato da una elevata capa-

città nell'individuare le interazioni dirette tra proteina e proteina, mentre il secondo riconosce agevolmente i complessi proteici stabili. Negli ultimi anni, parallelamente all'evoluzione di tali tecniche in-vitro, si sono affermate molteplici tecniche in-silico. Per ovvie ragioni esse risultano particolarmente veloci e convenienti se confrontate con le prime e continuano a diffondersi con il sequenziamento di nuovi organismi. E' necessario osservare, però, che queste tecniche richiedono che le proteine siano ortologhe e falliscono nel momento in cui tali relazioni di ortologia non risultano ben definite. Le tecniche in-silico sono caratterizzate dal fatto che tendono a produrre un gran numero di falsi positivi poiché privilegiano un'alta sensibilità al costo di una bassa specificità.

4.3 Database pubblici

Così com'è avvenuto in altri campi della Bioinformatica, anche per le reti di interazione proteina-proteina si sono resi disponibili database per la loro consultazione. Un aspetto che è apparso da subito palese è stata la notevole difformità dovuta principalmente ai particolari approcci utilizzati per l'estrazione di tali reti. Pertanto, già a partire dal 2004 ci sono stati tentativi (42) di standardizzazione, ma in genere, la flessibilità insita dei dati modellati, costringe la comunità scientifica a continue operazioni di integrazione a aggiornamento. Attività che risulta tutt'altro che banale. Nella tabella 4.1 sono descritti i principali database che ospitano informazioni relative alle reti di interazione proteina-proteina con una breve descrizione delle rispettive caratteristiche.

4.4 Strutture biologiche all'interno di una rete di interazione proteina-proteina

Lo studio delle reti di interazione proteina-proteina a livello biochimico si è da sempre rivolto al riconoscimento delle sotto strutture in esse contenute con grande dispendio di energie e con risultati spesso approssimativi. Oggi, grazie all'avvento della biologia computazionale questo tipo di ricerche non solo sono possibili in tempi rapidissimi, ma inoltre utilizzando le enormi basi di dati disponibili è possibile effettuare delle analisi prima impensabili. Bisogna notare che pur essendo disponibili moltissimi tool che operano in generale su grafi, pochissimi sono orientati direttamente alla grandi reti di interazione proteina-proteina.

4.4.1 Complessi proteici

I processi cellulari rappresentano un meccanismo fondamentale per la vita della cellula e sono possibili grazie all'interazione diretta di gruppi di proteine. In queste condizioni, tale insieme di proteine prende il nome di *complesso proteico*. Un primo aspetto fondamentale che contraddistingue queste sottostrutture è rappresentato dal fatto che pur esistendo delle proteine che costituiscono una sorta di nucleo del complesso, durante i vari processi cellulari in cui il complesso è coinvolto possono aggiungersi altre proteine per brevi periodi di tempo. Un secondo aspetto è costituito dal fatto che una o più proteine di un complesso possono prendere parte ad altri complessi perché coinvolti anche in altri processi. Con queste premesse, è facile intuire come la vera grande battaglia che tutti i metodi *in vitro* e *in silico* devono affrontare è costituita dalla capacità di individuare correttamente quei gruppi di proteine che rappresentano davvero un complesso. Grazie agli studi basati sulla spettrometria di massa è stato possibile individuare i complessi proteici in *S. Cerevisiae* in maniera più efficace di quanto non avvenuto con il metodo *yeast 2-hybrid*. Alla base degli studi dei complessi proteici c'è senz'altro il confronto tra specie differenti. Mediante questo approccio (34) è stato possibile ad esempio mostrare che proteine ortologhe tendono ad interagire con complessi in cui sono presenti altre proteine ortologhe. Questo significa che quelle interazioni che si conservano durante l'evoluzione della specie potrebbero rappresentare un nucleo di funzioni comuni alla cellule eucariote (43). Il confronto tra reti di interazione proteina-proteina e complessi proteici rappresenta il principale obiettivo degli esperimenti che sono stati eseguiti per la validazione degli algoritmi di ricerca esatta ed inesatta qui presentati. Da quanto descritto in precedenza si evidenzia una caratteristica fondamentale valida per tutte le specie:

- esistono molteplici rappresentazioni delle reti di interazione e spesso sono discordanti per numero di proteine e di interazioni coinvolte;
- esistono complessi ben studiati e ben definiti;
- esitono complessi la cui rappresentazione è spesso discordante;
- esistono complessi non ancora individuati.

L'idea che sta quindi alla base dei tool di graph querying descritti nei capitoli successivi è quella di sfruttare le conoscenze relative ai complessi ben studiati per verificarne la possibile corrispondenza a porzioni della rete di interazione proteina-protiena di altre specie. Ovviamente una eventuale risposta affermativa non significa

immediatamente l'avere individuato un nuovo complesso, ma pone le basi per una ricerca più specializzata.

4.4.2 Pathway molecolari

I *pathway* molecolari rappresentano una lunga catena di reazioni molecolari grazie alle quali è possibile la sopravvivenza dell'organismo. Molteplici sono i processi cellulari che vedono coinvolti i pathway: per questa ragione parliamo di pathway metabolici, pathway dell'apoptosi e *signaling* pathway. L'alterazione di un pahtway può causare malattie gravi come ad esempio il cancro. Ciò spiega in maniera efficace come la conoscenza di patway rappresenti senza alcun dubbio un passo fondamentale nello studio e nell'interpretazione dei complessi processi cellulari. Per tale ragione, l'individuazione di pathway a partire da reti di interazione proteina-proteina è divenuta nel tempo una importantissima area della ricerca della Bioinformatica.

4.5 Proprietà delle reti di interazione proteina-proteina

L'analisi sistematica in una rete di interazione proteina-proteina di una data specie permette di conoscere il significato funzionale di una data proteina o di gruppi di proteine. Dal confronto di due o più reti appartenenti a specie diverse è possibile individuare le porzioni di sottoreti comuni che si sono conservate durante il processo evolutivo delle specie. Su questi gruppi di proteine e sulle loro interazioni ci si può senz'altro focalizzare per individuare quelle funzioni biologiche che si sono conservate nel tempo e che ricoprono quindi un ruolo chiave nel complesso sistema dei processi cellulari delle specie analizzate.

4.5.1 Reti con topologia *scale-free*

Uno studio basato sul database WIT (44) ha dimostrato che le reti metaboliche di 43 organismi censiti all'interno di questo database presentano una topologia scale free con $(Pk) \approx k^{-}2.2$ sia per gli archi in entrata sia per quelli in uscita. Il diametro delle reti metaboliche risulta identico per tutte le specie prese in considerazione, dimostrando che con l'aumento di complessità della specie tende a crescere la connessione dei nodi. Queste reti sono caratterizzate dalla presenza di pochi hub connessi con la quasi totalità dei nodi della rete: la loro rimozione produce infatti un aumento nella dimensione del diametro. La percentuale dei nodi/proteine comuni e tutte le reti è pari al 4%: questi nodi presentano la caratteristica di essere altamente connessi in tutte le reti

dei diversi organismi. In contrapposizione a questo aspetto se ne evidenzia un altro: le sottostrutture specifiche di un sola specie, presentano poche connessione ai nodi. Infine, la rimozione casuale di un nodo all'interno della rete non altera la media dei cammini minimi, dimostrando che la rete ha una sua capacità intrinseca di sopperire ad errori casuali. Recenti studi (45) hanno dimostrato che è presente una selezione evolutiva su larga scala di un struttura comune per le reti biologiche e che la vera impresa per la ricerca sarà quella di individuarla nelle specie ancora meno studiate.

4.5.2 Reti con topologia gerarchica

Da un'analisi effettuata in (44) si è dimostrato che le reti di interazione proteina-proteina, oltre ad essere scale-free, presentano un coefficiente di clustering medio che risulta essere più grande di un ordine di grandezza se paragonato a quello di generiche reti scale free della stessa dimensione. Questo suggerisce che le reti di interazione proteina-proteina siano caratterizzate da una elevata modularità, siano composte da molteplici moduli di piccola dimensione e altamente connessi, organizzati gerarchicamente all'interno di unità più ampie.

4.5.3 *Network Motif*

Si definiscono *Network Motif* i pattern di interconnessioni che ricorrono in molteplici porzioni di una data rete, con una frequenza maggiore rispetto a quella che si potrebbe trovare in una rete generata casualmente. Utilizzando i network motif per analizzare la rete di regolazione della trascrizione di E. Coli si è dimostrato (6) che la quasi totalità di tale rete può essere descritta mediante la composizione di un numero definito di motif, ognuno dei quali svolge una specifica azione nella determinazione dell'espressione genica.

4.5.4 Reti con topologia geometrica

Nel modello geometrico casuale, i nodi corrispondono a punti distribuiti in maniera indipendente ed uniforme all'interno dello spazio metrico e due nodi sono collegati da un arco se la loro distanza è inferiore o uguale ad un prefissato raggio r. E' stato utilizzato il termine di *graphlet* per indicare una piccola rete costituita da un numero limitato di nodi (3,4,5). Tutti i possibili grafi o reti che possono essere creati in questo modo, sono stati ricercati all'interno di molteplici reti di interazione proteina-proteina, all'interno di reti con le stesse caratteristiche, ma costruite in maniera casuale e all'interno di reti geometriche casuali. E' stato dimostrato che la frequenza della distribuzione dei

graphlet maggiormente presenti in una rete PPI è molto vicina a quella rilevata nelle reti geometriche casuali. Un lavoro molto interessante in questa direzione è stato svolto in (46)

4.6 Relazione funzione-struttura in una rete di interazione proteina-proteina

L'organizzazione di una rete biologica non è casuale e sono le sue proprietà a caratterizzarne la funzione. E' stato dimostrato che:

- le reti complesse sono costituite da semplici blocchi fondamentali;
- le differenti classi funzionali delle proteine sono legate a reti con differenti caratteristiche.

Utilizzando la classificazione adottata nel database MIPS (47) si possono determinare le proprietà delle reti relative ai diversi gruppi funzionali (48). E' stato dimostrato che le proteine coinvolte nella traduzione tendono ad essere caratterizzate da un elevato grado medio, mentre quelle coinvolte del transporto sono caratterizzate da un grado medio molto basso. Sulla base di queste osservazioni è possibile effettuare delle predizioni sulla particolare funzione biologica svolta da una data sottorete analizzandone quindi le caratterisitche topologiche. Infatti, nel momento in cui si passa da una piccola rete ad una grande rete di interazione proteina-proteina, nasce l'esigenza di studiarla in sottoparti, ovvero in moduli. Un modulo può essere infatti definito come una collezione di proteine che eseguono una specifica attività cellulare. I moduli funzionali sono molto utili per annotare una proteina non ancora caratterizzate ovvero per studiare l'evoluzione di sistemi interagenti. L'esistenza di tali moduli è stata dimostrata sperimentalmente ed ha permesso di individuare alcuni moduli, ad es. la glicolisi, che sono stati ricostruiti *in vitro*.

Nome	Descrizione	Organismi	Riferimento
Database of Interacting Proteins (DIP)	Reti di interazione PP determinate sperimentalmente	372	(38)
Munich Information Center for Protein Sequences (MIPS)	Reti di interazione proteina--proteina, complessi, pathway	Mammiferi	(39)
Human Reference Protein Database (HRPD)	Reti di interazione proteina-proteina correlate a malattie	*Homo Sapiens*	(40)
The Drosophila Interactions Database (DroID)	Reti di interazione proteina-interazioni genetiche	*D. Melanogaster*	(41)

Tabella 4.1: Database pubblici per reti di interazione proteina-proteina

5

Reti di regolazione della trascrizione

5.1 Introduzione

Le reti di regolazione della trascrizione svolgono un ruolo fondamentale all'interno del complesso meccanismo cellulare che controlla l'espressione del genica. Un interessante studio nell'ambito di questo particolare tipo di reti è rappresentato da (49) L'azione di questo meccanismo è resa possibile grazie ai fattori di trascrizione, rappresentati da quelle proteine in grado di legarsi a specifiche sequenze di DNA all'interno delle regioni di regolazione dei geni. Normalmente avviene che più fattori di trascrizione regolano un singolo gene, ma può capitare anche che più fattori di trascrizione possono controllare più di un gene. Negli eucarioti superiori, i fattori di trascrizione possono anche cooperare. Può accadere infatti che un insieme di differenti fattori di trascrizione sia necessario per iniziare la trascrizione di un particolare gene. Un gene può essere attivato per mezzo di differenti insiemi di fattori di trascrizione, abilitando in tal modo la trascrizione di un gene sotto differenti circostanze Ogni fattore di trascrizione può prendere parte nella regolazione di molteplici geni. I geni che che codificano fattori di trascrizione, sono essi stessi soggetti al controllo da parte di altri fattori, ma anche dei fattori codificati. Da questo insieme di complesse relazioni che variano nel tempo può essere ricostruita una rete di trascrizione che complessivamente si occuperà di controllare i processi biologici essenziali come la morfogenesi, la proliferazione cellulare, la differenziazione... Un primo importante aspetto è rappresentato dal fatto che i fattori di trascrizione tendono ad evolvere molto più velocemente dei dei geni che controllano. Un secondo aspetto riguarda invece la topologia della maggior parte delle reti di re-

golazione della trascrizione, che ancora una volta si presentano come reti di tipo scale free.

5.2 Identificazione di Motif all'interno di una rete di regolazione della trascrizione

I moderni dispositivi elettronici sono costruiti utilizzando una combinazione di circuiti elementari di base. E' stato dimostrato che anche nel caso di reti biochimiche e reti neuronali è possibile riscontrare una simile proprietà. I Motif sono dei particolari sottografi presenti in una rete un numero di volte di gran lunga superiore rispetto ad una rete costruita interamente in modo casuale(50). Come descritto in (29) i Motif svolgono un ruolo importante nel processo dell'informazione all'interno della rete. Infatti, uno dei primi Motif individuati, il Motif di tipo *feed-forward-loop*, ha la capacità di eseguire particolari processi informativi quali il *sign-sensitive filtering*, la *response acceleration* e la *pulse-generation* (51). Col tempo sono stati individuati nuovi ed importanti Motif e piano piano si è reso necessario un meccanismo per individuarli in maniera automatica (o quasi) a partire da una rete di regolazione della trascrizione nota. L'idea alla base di questa procedura è stata quindi quella di individuare tutti quei sottografi all'interno della rete di regolazione che appaiono un numero di volte superiore rispetto alla corrispondente rete costruita in maniera casuale.

5.3 Reti di regolazione della trascrizione, Motif e sistemi di Graph Querying

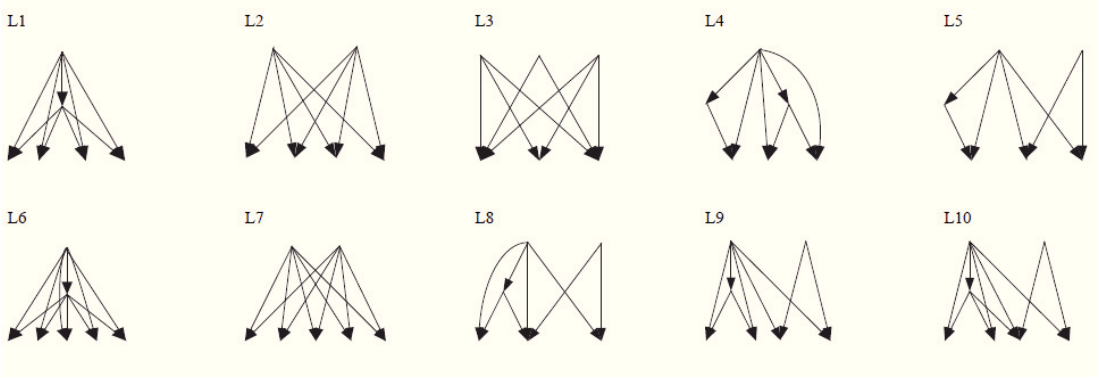


Figura 5.1: Motif in una rete di regolazione della trascrizione. - Un esempio di possibili motif di ordine superiore presenti all'interno della rete di E. Coli

In Figura 5.1 sono illustrati alcuni dei Motif individuati da (6) e che sono stati utilizzati come grafi query negli esperimenti eseguiti mediante il tool SING (21). L'idea che è stata alla base degli esperimenti realizzati su questa tipologia di reti è stata quella di utilizzare i 20 Motif individuati da (6) etichettati con differenti valori discretizzati dell'espressione genica ed utilizzarli come grafi query sulla più estesa componente connessa della rete di regolazione della trascrizione di *E. Coli*. In questo modo è stato possibile individuare facilmente la presenza di tali Motif all'interno della rete di trascrizione, oltre a poter verificare anche la particolare tipologia sulla base della specifica espressione genica assegnata ai singoli nodi del Motif.

6

Graph querying in grandi reti biologiche

6.1 Introduzione

La disponibilità di grandi database di grafi che modellano dati strutturati in molteplici campi di applicazione rappresenta senz'altro una grande *chance* per la ricerca. La crescita esponenziale dei dati disponibili in questi database ha spinto la comunità scientifica a sviluppare strumenti sempre più efficienti per manipolare tali database e per rispondere in maniera performante alla problema della ricerca di un grafo in un database di grafi. Dato un database di grafi ed un grafo *query* si vogliono conoscere tutti i grafi del database che sono uguali o semplicemente *contengono* il grafo query passando attraverso un meccanismo di indicizzazione. Pur partendo da premesse differenti ed usando anche strumenti differenti, l'elemento comune della maggior parte dei tool che in contesti diversi risolvono questo problema è rappresentato dalla presenza di un indice inverso dei grafi del database. Come accade ad esempio, nell'ambito dell'elaborazione di immagini (52), della Bioinformatica (53), database XML-based (54). Una possibile alternativa all'utilizzo di un indice potrebbe essere una ricerca sequenziale all'interno del database di grafi e quindi del singolo grafo. Ma, per ovvie ragioni, questa operazione è computazionalmente costosa ed inefficiente. Infatti, il problema dell'isomorfismo fra grafi è un problema NP-completo (55) e pertanto le uniche soluzioni al problema proposto che rispondono in maniera efficiente effettuano una indicizzazione dei grafi per filtrare il database iniziale, escludendo solo quei grafi che in maniera più o meno evidente differiscono dal grafo query.

6.2 Ricerca di sotto-strutture in grandi reti biologiche

Come descritto in precedenza, in molteplici domini biologici i dati sono rappresentati mediante grafi. La ricerca di sotto-strutture all'interno di un grafo o di un database di grafi costituisce un'attività chiave per riuscire ad identificare la similarità tra molecole, reti di molecole, reti di interazione proteina-proteina etc. Per esempio, la ricerca di pathway proteici all'interno di una collezione di reti di interazione proteina-proteina consente di identificare i pathway che si sono conservati nell'evoluzione e può inoltre fornire un valido supporto nell'annotazione funzionale delle proteine e nell'ancor più interessante predizione delle interazioni. Il dominio della Bioinformatica comprende una moltitudine di dati strutturati quali genomi, proteine, reti biologiche ed in breve tempo il grafo è divenuta la struttura dati destinata alla loro rappresentazione e manipolazione grafica e computazionale. Il data mining su grafi applicato alla Bioinformatica si è rivolto principalmente alla ricerca esatta ed inesatta di sottostrutture (21, 27). La ricerca di sottostrutture nelle grandi reti biologiche si avvale delle strutture e degli algoritmi propri della ricerca dei match esatti ed inesatti nell'ambito dei grafi.

6.2.1 Matching esatto tra grafi

Il matching tra grafi trova applicazione in un numero sempre crescente di ambiti, non solo nella Bioinformatica, ma anche in sociologia, elettronica, chimica fino ad arrivare a contesti border-line come la prevenzione di attività terroristiche. Effettuare il matching tra due grafi significa ricercare i possibili appaiamenti tra i nodi dei due grafi in modo tale che vi sia corrispondenza tra gli archi e/o le etichette. In effetti, il problema del matching fra grafi si riconduce a un problema ben noto in letteratura: il subgraph isomorphism fra due grafi. Si tratta, come anticipato in precedenza, di un problema NP-Completo (55). Una variante particolarmente interessante dal punto di vista biologico è rappresentata dal match inesatto, mediante il quale un match è possibile anche mediante l'inserimento/cancellazione di nodi/archi. Lo scopo del matching esatto tra grafi o *exact graph mathing* è quello di determinare se due grafi o una loro parte è identica sia dal punto di vista strutturale sia da quello delle etichette presenti ai nodi e/o agli archi. Le possibili strutture dati che sono utilizzate per la rappresentazione del grafo sono le liste di adiacenza e le matrici di adiacenza. Pertanto, si potrebbe pensare che il confronto tra due grafi si possa ridurre ad un confronto tra due liste di adiacenza o due matrici di adiacenza. Escludendo le prime per ovvie ragioni, potrebbe apparire plausibile utilizzare le seconde per potere verificarne l'identicità. In effetti, in un grafo non esiste alcun ordine tra nodi e archi: ciò significa che per un grafo formato da n

nodi vi sono n! possibilità di ordinare i suoi nodi e quindi è possibile definire fino a n! differenti matrici di adiacenza. Ciò spiega il motivo per cui il confronto diretto tra le due matrici di adiacenza non può risolvere il problema. Per capire se esiste davvero una identità tra due grafi si utilizza una particolare funzione che prende il nome appunto di *isomorfismo tra grafi*, che verifica l'eventuale coincidenza del primo grafo rispetto al secondo. Senza perdita di generalità, nei capitoli seguenti, si utilizzeranno grafi indiretti etichettati ai nodi e non agli archi. In tutti gli altri casi sarà sempre possibile ricondurre il grafo a questa tipologia.

6.2.2 Match inesatto tra sottografi

In precedenza, descrivendo le reti di interazione proteina-proteina, è stato evidenziato come il processo che ha per output il grafo che rappresenta la vera e propria rete di interazione, è tutt'altro che esente da errori e particolarmente suscettibile alle condizioni a contorno. Ciò significa che spesso è più corretto parlare di una collezione di grafi che descrivono un certa struttura biologica, piuttosto che di un singolo grafo. Quindi appare evidente il paradosso che potrebbe accadere che persino due grafi che teoricamente rappresenterebbero la medesima struttura biologica (una rete di interazione proteina-proteina, proseguendo nell'esempio) potrebbero risultare non isomorfi, ma, piuttosto, potrebbero condividere soltanto moltissimi e grandi sottografi. Un secondo aspetto particolarmente importante quando si parla del confronto tra due grafi è rappresentato dall'alfabeto che è utilizzato per etichettare tali grafi. Accade spesso che i nodi di un grafo siano etichettati con una etichetta che si riferisce a valori non discreti (si veda per esempio il paragrafo 7.4), generando quel *rumore* che rende di fatto non isomorfi grafi che in effetti lo sono. Quanto evidenziato sinora rivela i notevoli limiti di questa tipologia di ricerca in applicazioni reali. Per ovviare a tale limite è necessario che un algoritmo progettato per risolvere una simile tipologia di problema ed in un contesto reale, sia in grado di gestire le differenze rilevate tra i due grafi con una certa tolleranza. Sono stati proposti nel tempo diverse tipologie di algoritmi che risolvono questo problema. Questi sono: distanza di editing su grafi; Reti neuronali; Rilassamento dell'etichettatura (Relaxation Labelling) e Spectral Methods.

6.3 Indicizzazione di grafi

6.3.1 Introduzione

Ricercare in maniera efficiente un grafo all'interno di un altro grafo o, più in generale, un grafo all'interno di un database di grafi, rappresenta senz'altro l'obiettivo di tutti i tool che utilizzano grafi per rappresentare la loro base di informazione. Dato un database di grafi $D = G_1, G_2, ..., G_n$ e un grafo query Q la *graph search* restituisce un insieme $D_Q = G....$, dove M rappresenta una funzione booleana che risponde ai tre principali quesiti di questa tipologia di ricerca:

- ricerca della struttura completa che corrisponde al graph isomorphism
- ricerca della sottostruttura, che corrisponde al subgraph isomorphism
- ricerca per similarità della struttura completa, che corrisponde al match approssimato e
- ricerca per similarità della sottostruttura, che corrisponde al subgraph match approssimato.

La ricerche esatta ed inesatta devono quindi essere risolte in maniera efficiente se davvero si vuole che l'algoritmo proposto abbia un'applicazione a problemi reali. In altre parole, ciò che si richiede è che dato un grafo query si riescano ad ottenere in tempo quanto più rapido possibile tutti i grafi appartenenti ad un grande database che possono contenere il grafo query, mediante l'utilizzo di un indice. I sistemi di indicizzazione di grafi si basano su uno schema di filtro-e-verifica che include due fasi principali: (1) proprocessing, durante la quale il sistema costruisce l'indice mediante la scansione del database di grafi; (2) query processing, attraverso la quale l'indice è sfruttato per rispondere in maniera efficiente alla query. A sua volta la fase di query processing può essere suddivisa in sotto-fasi: filtering e matching. La prima sotto-fase ha il compito di scartare tutti i grafi che non possono contenere il grafo query, generando un insieme di grafi candidati. La seconda sotto-fase esegue un subgraph isomorphism su tutti i grafi candidati. La Tabella 6.1 sintetizza le caratteristiche principali dei più diffusi sistemi di indicizzazione utilizzati nei capitoli successivi.

6.3.2 Indicizzazione basata su feature

Dato un database $D = \{G_1, G_2, ..., G_n\}$ di grafi, la ricerca esatta di uno o più match relativi alla query Q rispetto al database D comporta la ricerca di tutti i grafi G in

Tabella 6.1: Sistemi di indicizzazione di grafi

System	Features	Data mining	All matches
GraphGrep (7)	Path	No	Si
gIndex (2)&Grafil (15)	Si	No	
FGIndex (10)	Grafi	Si	No
GDIndex (56)	Grafi	No	Si
TreePi (57)	Tree	Si	No
Tree+δ(12)	Tree+Grafi	Si	No
CTree (3)	-	No	No
GCoding (4)	-	No	Si
SING (21)	Path	No	Si

D tali $Q \subseteq G$. Un sistema di indicizzazione basato su feature utilizza un framework per la selezione e verifica che permette di filtrare quei grafi del database che non contengono certamente la query. Per feature si intende un piccolo grafo che permette di discriminare, verificandone appunto la sua presenza, che che il grafo target contenga o meno il grafo query. Denominiamo $\mathcal{F}$ l'insieme di tutte le possibili feature. La scelta di $\mathcal{F}$ dipende dal particolare sistema utilizzato. Un sistema di indicizzazione di grafi che fa uso del concetto di feature, si basa sull'osservazione che affinché una query Q ammetta un match con il grafo G, è necessario che ogni feature di $\mathcal{F}$ contenuta in Q risulta contenuta anche in G. Più precisamente, quando affermiamo che una feature f è contenuta in G significa che esiste un graph isomorphism tra f ed un sottografo di G. Il pruning è realizzato eseguendo le seguenti tre fasi:

- **Preprocessing:** è una fase che avviene *off-line* ed è totalmente indipendente dalla query che sarà eseguita successivamente. L'algoritmo di indicizzazione esamina ogni grafo del database al fine di estrarre tutte le feature di $\mathcal{F}$ che sono contenute all'interno del grafo. L'insieme costituito da tutte le feature di tutti i grafi del database è memorizzato in una struttura che prende appunto il nome di *indice dei grafi*.

- **Filtering:** la query Q è esaminata al fine di estrarre tutte le feature che sono contenute in Q. L'algoritmo calcola l'insieme dei grafi candidati confrontando le feature estratte da Q rispetto a quelle contentue all'interno dell'indice.

- **Matching:** l'algoritmo esamina ciascun grafo candidato per poter verificare che esista realmente un match tra il grafo query ed il grafo target.

Le strutture tipiche utilizzate come feature per i più efficienti sistemi di indicizzazione noti sono:

- path;
- alberi;
- grafi.

Di seguito, sono descritte queste tre strutture in relazione al sistema di indicizzazione. Si rimanda a (12) per un'analisi più approfondita delle caratteristiche in termini di efficacia ed efficienza delle tre varianti di seguito descritte.

6.3.2.1 Feature basate su path

I path di lunghezza predefinita rappresentano una delle possibili strutture da utilizzare in qualità di feature per consentire l'indicizzazione dei grafi di un database. Per la loro semplicità e dinamicità sono stati adottati dagli algoritmi di graph querying descritti all'interno dei capitoli conclusivi. L'idea che sta alla base della indicizzazione basata su feature di tipo path è quella di calcolare, per ogni grafo del database, tutti i possibili path di lunghezza massima $maxL$. Questo produrrà un indice inverso, in cui per ogni feature conservata è noto l'indice del grafo di appartenenza. Pertanto una feature sarà una sequenza di vertici (o meglio etichette dei vertici) del tipo v_1, v_2, v_k, tale che $\forall 1 \leq i \leq k-1$ e (v_i, v_i+1) rappresenta un arco. I vantaggi fondamentali attribuibili all'indicizzazione basata su path si legano principalmente alla semplicità computazionale di tali strutture se paragonate agli alberi ovvero ai sottografi. Inoltre, dato un grafo o un database di grafi e nota la $maxL$ è possibili conoscere anticipatamente la dimensione massima dell'indice è nota. Due sono i possibili svantaggi di questo approccio. La decomposizione di un grafo nelle sue feature può comportare nella fase di filtering l'individuazione di falsi positivi con conseguente appesantimento della successiva fase di matching. Inoltre, nel caso in cui l'alfabeto delle etichette risulti particolarmente limitato, l'adozione di un sistema di indicizzazione potrebbe risultare inappropriato. Viceversa, la disponibilità di un alfabeto delle etichette molto vasto potrebbe rendere il sistema meno performante rispetto ad sistema diretto di graph/subgraph isomorphism.

6.3.2.2 Feature basate su alberi

Al fine di sopperire alle lacune mostrate da un sistema di feature basato su path, è stato proposto un sistema di indicizzazione basato su feature di tipo albero (12). La

struttura ad albero riesce a catturare meglio le informazioni topologiche relative ai grafi che si vogliono indicizzare rispetto al semplice path. Inoltre le caratteristiche di questa struttura dati la rendono senz'altro più performante rispetto al grafo anche se la sua manipolazione è più complessa rispetto a quella del path. Infatti, a differenza del graph isomorphism, il tree-isomorphism può essere risolto in tempo polinomiale. TreePi (57) rappresenta il primo tentativo di utilizzo degli alberi come feature. Gli autori descrivono un algoritmo lineare nel tempo per il calcolo dell'etichettatura canonica di un albero e dimostrano sperimentalmente che le feature di tipo albero riescono a catturare la struttura topologica molto bene. Inoltre, rappresentano un buon compromesso tra efficienza ed efficacia nella fase di filtering. Come mostrato dagli autori, è sempre possibile definire un centro unico per un dato albero. Di conseguenza, è possibile calcolare la distanza (inteso come cammino minimo) tra coppie di feature in un grafo. TreePi utilizza una ulteriore regola di pruning basata sulla distanza tra feature al fine di migliorare la qualità del match. Più precisamente, tale regola si basa sull'osservazione che affinché un grafo sia isomorfo ad un grafo target, la distanza tra ogni coppia di vertici nel grafo query non può essere inferiore alla distanza tra i vertici corrispondenti nel grafo target. Tree+δ (12) utilizza invece sia feature basate su alberi, sia feature basate su una ristretta classe di piccoli grafi anche in questo caso per migliorare la capacità di filtering. Esso effettua inizialmente l'estrazione e l'indicizzazione degli alberi frequenti, e successivamente, seleziona all'interno della query un insieme di grafi di piccole dimensioni dall'elevato potere discriminante, che dovrebbe consentire di filtrare meglio i grafi. La selezione di questi piccoli grafi è realizzata appositamente per ogni data query. Per far ciò, il pruning power di un grafo è calcolato approssimativamente mediante le sua feature con un limite superiore ed inferiore. Data una query, Tree+δ individua tutti i sottoalberi frequenti di Q fino alla dimensione massima $maxL$. Utilizzando questo insieme di feature che denotiamo con $T(Q)$), il tool calcola l'insieme dei grafi candidati $C(Q)$, intersecando l'insieme dei grafi che supportano t, $\forall$ t $\in T(Q)$. Se Q è un non-tree cyclic graph, esso dà luogo ad un insieme F di feature discriminanti non basate su alberi. Tali feature f possono essere memorizzate nella cache già dalla precedente ricerca. In caso contrario, Tree+δ effettuerà la scansione di tutti i grafi del database per generare l'indice inverso tra f ed i grafi di D. Quindi l'insieme $C(Q)$ è intersecato con l'insieme di grafi di supporto a $D(f)$.

6.3.2.3 Feature basate su grafi

gIndex (2), GDIndex (56) e FGIndex (10) rappresentano tre esempi di sistemi che fanno uso di feature basate su grafi. Questi tool considerano un insieme di feature $\mathcal{F} = \mathcal{G}_{/\approx}$,

dove $\mathcal{G}$ è l' universo dei grafi e $\mathcal{G}_{/\approx}$ rappresenta la $\mathcal{G}$ indotta dal graph isomorphism. Tutti i grafi isomorfi sono considerati come una singola feature rappresentata dalla loro classe di equivalenza. Il principale vantaggio nell'utilizzo di feature di tipo grafo è costituito dal fatto che sono più adatte a catturare la struttura topologica dei grafi e ovviamente tendono a produrre pochi candidati. Il principale svantaggio è invece legato al fatto che il numero di feature cresce in maniera esponenziale con la dimensione del grafo: come conseguenza si avrà un indice di grandi dimensioni che provocherà un degradamento delle prestazioni sia nella fase di preprocessing sia di filtering. Al fine di risolvere questo problema, gIndex (2) and FGIndex scelgono ad esempio feature basate su insiemi di sottografi frequenti.

6.3.2.4 Feature basate su sottostrutture frequenti

Le soluzioni alternative all'utilizzo di feature di tipo path, si sono concentrate su strutture più o meno complesse al fine di sopperire alla perdita di informazioni dovuta alla semplicità della prima. Come conseguenza nell'utilizzo di un simile approccio si ha una esplosione nel numero di feature che possono essere estratte, rendendo pressoché inapplicabile un tale sistema di indicizzazione per casi reali. Per sopperire a tale lacuna si è pensato di considerare non più tutte le feature ma soltanto quelle caratterizzate da una soglia prefissata. Questo compromesso consente di sfruttare appieno le caratteristiche di feature più complesse e allo stesso tempo di ridurre sensibilmente le dimensioni dell'indice. gIndex (2) considera anche il concetto di *sottografi discriminanti* per riuscire a ridurre ulteriormente il numero di feature. Tutti questi approcci richiedono l'esecuzione all'interno della fase di preprocessing di un passo di mining molto costoso, con conseguente perdita di efficienza. Inoltre, quando questi approcci sono applicati a grafi di grandi dimensioni, questo step di mining può risultare impraticabile. FGIndex utilizza un piccolo indice residente nella memoria principale e memorizza la parte restante dell'indice in memoria secondaria. Gli autori di FGIndex utilizzano il concetto innovativo di *δ-tolerance closed frequent subgraph* per distinguere tra features residenti nella memoria primaria e feature residenti nella memoria secondaria. Quando la query non può essere eseguita utilizzando soltanto l'indice residente nella memoria primaria, tale indice è utilizzato per identificare i blocchi della memoria secondaria da caricare. Per impedire costosi accessi al disco, è caricato in memoria un insieme ridotto di feature massimali che copre l'intero grafo query. GDIndex elenca tutti i sottografi indotti contenuti in ogni grafo del database. Il tool organizza tutte le feature mediante una rappresentazione di tipo DAG con una relazione d'ordine parziale $\preceq$ tra feature. La dimensione dell'indice è ridotta impedendo la ridondanza. Ogni feature è associata ad

un insieme di grafi che la contengono e che non contengono la feature antenata nel DAG. Durante la fase di filtering, l'insieme dei grafi che contengono una feature può essere dedotto da feature-DAG. Elencando tutti i sottografi di un grafo può risultare un'operazione dispendiosa. Pertanto, questo approccio può essere utilizzato soltanto su database di grafi di piccole dimensioni.

6.3.3 Algoritmi di indicizzazione non basati su feature

Non tutti i sistemi di indicizzazione di grafi si basano sul concetto di feature. Alcuni di questi tool come (3, 4) forniscono performance migliori se paragonati a taluni sistemi di indicizzazione basati su feature. In alcuni casi le prestazioni migliori sono giustificate dalla capacità di questi sistemi di recepire in maniera ottimale la struttura dei grafi coinvolti nella indicizzazione e ricerca. Tra questi tool annoveriamo CTree (3) e GCoding(4). CTree (3) organizza i grafi del database in una struttura dati di tipo R-tree. Le foglie rappresentano il singolo grafo ed i nodi interni gli insiemi di grafi sintetizzati mediante una struttura denominata *closure graphs*. Il closure graph di un insieme di grafi si ottiene nel seguente modo. Tutti i grafi dell'insieme sono allineati mediante un veloce algoritmo di approssimazione denominato *Neighbor Biased Mapping*. I vertici del closure graph sono etichettati mediante gli insieme di etichette che corrispondono ai vertici allineati. Allo stesso modo, gli archi del closure graphs sono gestiti come l'unione dei archi allineati. Dato un grafo, viene eseguito, in modalità *top-down*, un algoritmo di match approssimato sui closure graph che non produce falsi negativi. Non appena un closure graph di un nodo produce una risposta negativa, tutti i sottoalberi che hanno radice in quel nodo sono scartati così come tutte le sue foglie-grafo. I grafi rimanenti costituiscono i candidati che saranno successivamente verificati mediante un algorimo di exact matching. Nonostante la flessibilità ed il filtering power che lo contraddistinguono, CTree (3) evidenzia una efficienza di filtering limitata poichè l'esecuzione dell'algoritmo di match approssimato è molto costoso e necessita di essere applicato alla maggior parte dei closure graph. GCoding (4) utilizza le proprietà degli autovalori della matrice di adiacenza per scartare i grafi. In modo particolare, esso utilizza l'*Interlacing theorem* che lega gli autovalori della matrice di adiacenza dei grafi che fanno match. Nella fase di preprocessing, sono analizzati tutti i grafi del database. Per ogni vertice v appartenente ad un dato grafo, è calcolata la *vertex signature*. Questo calcolo coinvolge l'etichetta del nodo, le etichette dei nodi vicini ed il più alto autovalore della matrice relativa all'albero che ha radice in v e che rappresenta tutti i path di lunghezza n che partono da v. Le vertex signature di un grafo sono quindi unite per formare la cosiddetta *graph signature*. Infine, la graph signature sono orga-

nizzate in una struttura simile ad un B-tree al fine di renderne efficiente la ricerca. Nel momento in cui è fornita la query q, il sistema ne calcola sia la vertex signature sia la graph signature. La graph signature è utilizzata per individuare un primo insieme dei grafi candidati all'interno del B-tree. Successivamente, viene selezionato un secondo insieme di grafi candidati a partire dal primo scartando tutti i grafi le cui vertex signature non coincidono con quelle della query. La corrispondenza tra graph signature e vertex signature è definita mediante l'applicazione dell'Interlacing theorem. Mediante questa strategia di codifica basata sugli autovalori, GCoding (4) consente una rappresentazione compatta dell'indice. Allo stesso tempo, però, si rileva che il calcolo degli autovalori risulta costoso, determinando in tal modo una fase di preprocessing molto lenta. Infine, la perdita di informazioni introdotta dalla codifica scelta, produce una capacità di pruning meno efficace se paragonata a quella di CTree (3).

6.3.3.1 Concusioni

Da quanto descritto in precedenza si evince facilmente la particolare importanza dell'indicizzazione di database di grafi nei limiti dei casi sovramensionati. Infatti, non è corretto parlare di sistema di indicizzazione in generale, in quanto ogni sistema presenta dei punti di forza e di debolezza, ma ciò che è evidente è che in tutte le applicazioni reali in cui è possibile modellare i dati del problema mediante una collezione di grafi, allora sarà necessario adoperare il sistema di indicizzazione più appropriato. Nei capitoli successivi, saranno proposte due diverse tipologie di sistemi di indicizzazione, il primo orientato al match esatto, il secondo a quello inesatto, e saranno analizzate le possibili applicazioni in ambito biologico.

7

Un algoritmo di ricerca esatta di sottostrutture in grandi reti biologiche: SING

7.1 Introduzione a SING

SING (Subgraph search In Non-homogeneous Graphs) (21), è un sistema di indicizzazione capace di operare su grandi database di grafi. L'algoritmo di indicizzazione ed il relativo algoritmo di *filtering* è basato sul concetto di *feature*, che in questo caso è rappresentato da un path di lunghezza predefinita. Tutti i grafi presenti all'interno del database sono annotati dalle rispettive *feature*. L'idea innovativa che sta alla base dell'algoritmo proposto è costituita dall'utilizzo delle informazioni associate alla posizione delle *feature* all'interno del grafo. Nelle sezioni sperimentali è evidenziato il considerevole miglioramento delle prestazioni dovuto a tale accorgimento. I test, eseguiti su grafi reali e sintetici, sono stati focalizzati sulle reti di biologiche ed è stato dimostrato come il tool proposto garantisca prestazioni migliori rispetto ai più noti tool nel caso di database formati da grafi di dimensioni medio/grandi.

7.2 Descrizione dell'algoritmo

Il problema della ricerca esatta di una sottostruttura all'interno di un database di grafi può essere ricondotto alla risoluzione di due problemi ad esso direttamente connessi che sono: Problema *First_query_occurrence*: Dato un database di n grafi $D = \{g_1,g_2,...,g_n\}$ ed un grafo query q, eseguire la query q su D equivale a cercare tutti i grafi g di D

tali che q risulti un sottografo isomorfo a g. Senza perdita di generalità, assumiamo che tutti i grafi di D e il grafo query condividano lo stesso alfabeto Σ. Problema *All_query_occurrences*: Dato un database di n grafi $D = \{g_1,g_2,...,g_n\}$ ed un grafo query q, eseguire la query q su D equivale e cercare tutti i grafi isomorfi tra q e gli elementi of D. Poiché si farà un uso frequente della nozione di feature, introduciamo la seguente definizione:

Definizione 1 *Sia $\mathcal{G}$ l'insieme di tutti i possibili grafi in un dato alfabeto di etichette. Un insieme $\mathcal{F}$ rappresenta un* insieme di feature su $\mathcal{G}$ *se e solo se esiste una relazione binaria $is\_a\_feature \subseteq \mathcal{F} \times \mathcal{G}$ tale che la seguente proprietà risulta vera (*graph upward monotonicity*):*

$\forall f \in \mathcal{F}, q, g \in \mathcal{G}$,
$is\_a\_feature(f,q) \wedge q \precsim g \rightarrow is\_a\_feature(f,g)$

Nel seguito, $is\_a\_feature(f,g)$ si esprime dicendo che g *contiene* f. Ogni insieme di feature definisce una *pruning rule* per il problema del subgraph isomorphism:

Regola di pruning 1 *Se $is\_a\_feature(f,q)$ e $\neg$ $is\_a\_feature(f,g)$ allora q non può essere un sottografo isomorfo a g.*

Per insieme di feature intendiamo l'insieme $Paths_{\leq k}$ di tutti i path etichettati di lunghezza $\leq k$. Per path etichettato si intende una sequenza di etichette. Consideriamo l'insieme di feature $Paths\_occ_{\leq k}$ delle coppie (p,n), in cui p è un path etichettato di lunghezza $\leq k$ e n rappresenta un *lower bound* sul numero di occorrenze di p nel dato grafo. La corrispondente proprietà di pruning asserisce che se il grafo query q contiene almeno n occorrenze di un dato path etichettato p e g non contiene almeno n occorrenze di p, allora q non può essere un sottografo isomorfo di g e g può essere filtrato. Si noti che, in tutti gli esempi seguenti, se una feature f è una sotto-feature di una data feature f' di g allora anche f' è una feature di g. Formalizziamo tale definizione mediante la seguente nozione: Un insieme di feature *monotono decrescente* è un insieme di feature parzialmente ordinato $(\mathcal{F}, \preceq)$ tale che:

$\forall f, f' \in \mathcal{F}, g \in \mathcal{G}$,
$f \preceq f' \wedge is\_a\_feature(f',g) \rightarrow is\_a\_feature(f,g)$

Per esempio $Paths_{\leq k}$ è un insieme di feature monotono decrescente rispetto alla sottostante relazione tra path etichettati. $Paths\_occ_{\leq k}$ è monotono decrescente rispetto al numero di occorrenze. Non si tratta comunque di un insieme monotono decrescente rispetto alla sottostante relazione. Infatti in Figura 2.1, (ABC,2) è una feature di g_1 ma (AB,2) non è una feature di g_1.

Un insieme monotono decrescente di feature permette ulteriori ottimizzazioni nel corso della fase di selezione: la regola di selezione può essere ristretta soltanto alla massima feature f nella query. Ciò significa che nessun'altra feature f' nella query può essere strettamente più grande di f nell'ordine parziale di feature. Nel paragrafo successivo si descriverà in maniera dettagliata il sistema di indicizzazione proposto. Tale sistema si basa sulla classica regola di selezione 1 insieme ad una una nuova regola. I tre passi relativi allo schema selezione-verifica saranno discussi separatamente. La fase di preprocessing definisce due strutture dati (si veda Figura 7.1 per maggiori dettagli.

Approccio utilizzato

L'approccio utilizzato nell'algoritmo proposto si basa su una nuova regola di pruning basata sulla localizzazione delle feature che consente di ridurre il numero dei candidati provenienti dall'applicazione della Pruning rule 1. Questa nuova regola consente di catturare la struttura dei grafi consentendo di ridurre sensibilmente il numero dei candidati. L'informazione sulla localizzazione è utilizzata per ridurre lo spazio di ricerca durante la fase di verifica. L'idea che sta alla base di questa regola si ispira a Treepi (57), che utilizza il concetto di distanza tra le feature, che richiede però il calcolo di tutte le distanza tra coppie. In alcuni casi, in modo particolare quando deve essere applicato a grandi grafi, l'approccio di Treepi risulta computazionalmente costoso. Infatti, enumerare tutti gli alberi produce una esplosione nel numero di feature che deve essere successivamente ridotto mediante un opportuno step di data mining. Questo produce un incremento nel tempo di preprocessing a vantaggio della fase di filtering. Per il fatto che Treepi richiede il calcolo della distanza tra coppie di feature, soltanto un numero limitato di feature deve essere selezionato, generando in tal modo un elevato numero di candidati. In contrapposizione a questo approccio, SING considera tutti i path che iniziano da un dato nodo. Ciò richiede un tempo di calcolo inferiore dando luogo ad un tempo di preprocessing e di filtering molto basso. Utilizzando semplicemente feature basate su path, SING è in tal modo in grado di catturare la topologia dell'albero indotto da un nodo e, conseguentemente, richiede un numero molto basso di feature e permette di evitare un costoso processo di selezione delle feature. Si consideri il grafo in Figura 2.1.Si può facilmente verificare che q è un sottografo isomorfo di g_1 ma non di g_2. q contiene la feature (A, B) e (A, C) che sono anche contenute sia in g_1 sia in g_2. Utilizzando queste feature il grafo non può essere scartato. Si noti infine che entrambe le feature in q partono dallo stesso vertice. Questa stessa situazione si verifica in g_1 ma non in g_2. Più precisamente in in g_2 non vi è alcun vertice dal quale partono entrambe le feature. Conseguentemente, il vertice etichettato con A di q non può fare match con

Preprocessing

Input:
database of graphs $D = \{g_1, g_2, ..., g_n\}$
Output:
a global index GI
a local index for each graph $LI[g]$
Algorithm:
$GI \leftarrow \phi$
for each $g = (V, E, \Sigma, l) \in D$ do
  $LI[g] \leftarrow \phi$
  for each $v \in V$ do
    $mark(v) \leftarrow unmarked$
  for each $v \in V$ do
    $p \leftarrow (l(v))$
    $mark(v) \leftarrow marked$
    dfs_preprocessing(g,v,p,GI,$LI[g]$)
    $mark(v) \leftarrow unmarked$

dfs_preprocessing

Input:
a graph $g = (V, E, \Sigma, l)$
a vertex $v \in V$
a path feature $p = (a_1, a_2, ..., a_{|p|})$
a global index GI
a local index $LI[g]$
Output:
the updated global index GI
the updated local index $LI[g]$
Algorithm:
$GI[p][g] \leftarrow GI[p][g] + 1$
$LI[g][p][v] = 1$
if $|p| = l_p$ than
  return
for each $(v, u) \in E | mark(u) = unmarked$ do
  $p \leftarrow (a_1, a_2, ..., a_{|p|}, l(u))$
  $mark(u) \leftarrow marked$
  dfs_preprocessing(g,v,p,GI,$LI[g]$)
  $mark(u) \leftarrow unmarked$

Figura 7.1: Algoritmo di preprocessing -

alcun vertice di g_2, che può essere quindi scartato. Le seguenti affermazioni formalizzano il concetto appena espresso. Esse costituiscono una immediata conseguenza della definizione di subgraph isomorhism. Sia $start(f, g)$ l'insieme di vertici v tale che una occorrenza di f parte da v in g.

Teorema 1 *Dati due grafi* $q = (V_q, E_q, \Sigma, l_q)$ *e* $g = (V_g, E_g, \Sigma, l_g)$*, let* $\phi : V_q \to V_g$ *sia un subgraph isomorphism tra* q *e* g*. Per ogni vertice* $v \in V_q$ *vale la seguente:*
$\{f \in \mathcal{F} | v \in start(f, q)\} \subseteq \{f \in \mathcal{F} | \phi(v) \in start(f, g)\}$.

Teorema 2 *Dati due grafi* q*,* g*. If* $q \precsim g$ *allora per ogni vertice* v *di* q *deve esistere almeno un vertice* u *in* g *tale che* $\{f \in \mathcal{F} | v \in start(f, q)\} \subseteq \{f \in \mathcal{F} | u \in start(f, g)\}$.

L'affermazione 2 suggerisce un modo più efficace di filtrare il database di grafi. Dato un grafo candidato g, per ogni vertice v del grafo query q, deve esistere un vertice u di g tale che ogni feature che parte da v deve partire anche da u. Di conseguenza, se per qualche vertice di q non esiste questo vertice corrispondente u, allora g può essere scartato. L'affermazione 1 fornisce un metodo per ridurre lo spazio di ricerca da parte dell'algoritmo di matching. In altre parole, introduce una condizione più restrittiva sulle coppie di vertici abbinate.

7.2.1 Preprocessing

In questa fase avviene la creazione di un indice globale ($GI[f]$) che associa ogni feature al set di grafi che la contiene e il numero di occorrenze di ogni feature di ogni singolo grafo. Un secondo indice associa ogni feature all'insieme dei nodi di partenza di tutte le sue occorrenze. $LI[g][f][v] = 1$ se una occorrenza della feature f nel grafo g parte dal vertice v. Altrimenti $LI[g][f][v] = 0$. In questa fase sono visitati tutti i grafi presenti all'interno del database mediante una strategia *depth first* e sono elencate tutte le feature contenute all'interno di ogni grafo. Quindi, partendo da ogni vertice, sono visitati tutti i path di lunghezza massima l_p, dove l_p rappresenta una soglia prefissata, tipicamente $l_p \leq 10$, per default $l_p = 4$ Le strutture GI e LI possono essere implementate mediante delle *hash table*. Pertanto la complessità media dell'algoritmo precedentemente descritto relativamente al database D è $O(d_m^{(l_p-1)} \cdot v_a \cdot |D|)$ dove d_m rappresenta il grado massimo dei vertici e v_a è il numero medio di vertici nei grafi di D. Nella nostra implementazione le strutture GI e LI sono alberi binari, pertanto la complessità è calcolata come $O(d_m^{(l_p-1)} \cdot v_a \cdot |D| \cdot log(|D|) \cdot log(|\Sigma^{l_p}|))$ nel caso peggiore. Nell'implementazione, $LI[g][p]$ è rappresentata mediante un *bit array* la cui dimensione è pari la numero di nodi di g. Le soluzioni adottate riducono lo spazio dell'indice favorendo inoltre una fase di filtering più veloce (per maggiori dettagli si veda 7.2.2).

Filtering

Input:
a query graph $q = (V_q, E_q, \Sigma, l_q)$
a database of graphs D
a global index GI
a local index for each graph $LI[g]$
Output:
a set of candidates C_2
a map of vertices compatibility for each candidate graph $M[g]$
Algorithm:
// extract query features
$FQ \leftarrow \phi$
$FVQ \leftarrow \phi$
for each $v \in V_q$ do
  $mark(v) \leftarrow unmarked$
for each $v \in V_q$ do
  $p \leftarrow (l(v))$
  $mark(v) \leftarrow marked$
  dfs_filtering(g,v,p,FQ,FVQ)
  $mark(v) \leftarrow unmarked$

// first step filtering
$C_1 \leftarrow D$
for each $f \in FQ$ do
  $S \leftarrow \phi$
  for each $g \in GI[f]$ do
    if $GI[f][g] \geq FQ[f]$ then
      $S \leftarrow S \cup \{g\}$
  $C_1 \leftarrow C_1 \cap S$

// second step filtering
$C_2 \leftarrow C_1$
for each $g \in C_1$ do
  for each $v \in V_q$ do
    $M[g][v] = \bigcap_{f \in FVQ[v]} LI[g][f]$
    if $M[g][v] = \phi$ than
      $C_2 \leftarrow C_2 - \{g\}$

dfs_filtering

Input:
a query graph $q = (V_q, E_q, \Sigma, l_q)$
a vertex $v \in V$
a path feature $p = (a_1, a_2, ..., a_{|p|})$
a global query index GIQ
a local query index LIQ
Output:
the updated global query index GIQ

7.2.2 Filtering

La fase di filtering applica la regola di pruning 1 e l'affermazione 2 per scartare i grafi del database che non contengono la query (vedi Figura 7.2). La struttura FQ rappresenta l'insieme di feature contenute all'intero della query, estratte mediante una ricerca di tipo depth-first simile a quella utilizzata nella fase di preprocessing. Tale procedura esamina soltanto i path massimali all'interno della query, scartando tutti i path che risultano prefissi di path più grandi. La struttura FVQ associa ad ogni vertice l'insieme delle feature che partono da tale vertice. Tale struttura è utilizzata durante il secondo step della fase di filtering. Il primo passo della procedura di pruning calcola l'insieme

$$C_1 = \bigcap_{f \in FQ} \{g \in GI[f] : GI[f][g] \geq FQ[f]\} \tag{7.1}$$

che mantiene soltanto i grafi che hanno tutte le feature contenute nella query. Inoltre, se un grafo non possiede almeno lo stesso numero di occorrenze di ogni feature come nella query, allora sarà scartato. Se FQ e FVQ sono rappresentati mediante tabelle hash, la complessità media nell'estrazione delle feature della queryè $O(d_m^{(l_p-1)} \cdot |V_q|)$, dove d_m rappresenta il grado massimo dei vertici e V_q l'insieme dei vertici della query. La complessità del primo step della fase di filtering è $O(|\Sigma^{l_p}| \cdot |D|)$ utilizzando una *hash table*. In pratica la complessità risulta minore in quanto l'insieme dei grafi associati ad una feature è più basso di $|D|$ e non tutte le possibili feature sono presenti all'interno del database.

Second step filtering

Il primo passo della fase di filtering tiene in considerazione soltanto le occorrenze di una feature all'interno del database di grafi. Il secondo passo utilizza le informazioni sulla posizione per filtrare meglio il database. Per ogni grafo g che supera il test relativo al primo step di filtering, si definisce una associazione tra i vertici della query ed i vertici di g mediante la seguente procedura. Sia v un vertice della query e $FVQ[v]$ l'insieme di feature che partono da v che è stato precedentemente calcolato. L'algoritmo calcola $M[g][v] = \bigcap_{f \in FVQ[v]} start(f, g)$ come l'insieme dei vertici del grafo g compatibili con v. Se per qualche vertice v abbiamo $M[g][v] = \emptyset$ allora il grafo g è scartato. L'affermazione 2 garantisce la correttezza di questa secondo passo di filtering. Poiché LI è implementata mediante bit array, l'insieme $M[g][v]$ può essere calcolato in maniera efficiente mediante operazioni di AND logico. La complessità è $O(|C_1| \cdot |V_q| \cdot |\Sigma^{l_p}| \cdot v_a)$ se

M risulta implementato sia con vettori sia con hash table. Ovviamente questa tipologia di operazioni risulta assai banale per i moderni calcolatori.

7.2.3 Matching

L'algoritmo per il subgraph matching VF2 (5) è applicato ad ogni grafo candidato che ha superato quindi sia il primo sia il secondo test della fase di filtering. $VF2$ è un algoritmo di ricerca combinatoria che genera un albero di ricerca mediante branching states e che utilizza regole di compatibilità per filtrare la ricerca. Ogni stato di VF2 consiste in un match parziale tra la query ed il grafo target. Partendo da uno stato iniziale che consiste in un match vuoto, VF2 produce una sequenza di stati incrementando la dimensione del match parziale. Ad ogni passo è generato un nuovo stato aggiungendo al match parziale una coppia di vertici corrispondenti. Nel momento in cui non sia più possibile estendere il match parziale, allora l'algoritmo effettua un backtracking. Per decidere se due vertici possono coincidere, VF2 utilizza un insieme di regole di compatibilità topologica e di compatibilità semantica, delle quali la più utilizzata è senz'altro quella basata sul confronto di etichetta. SING utilizza un nuovo ed efficiente criterio per la compatibilità semantica dei nodi che consente di ridurre l'ampiezza dell'albero di ricerca. Come descritto nel paragrafo relativo al secondo passo della fase di filtering 7.2.2 per ogni vertice v della query, è calcolato un insieme $M[g][v]$ del grafo g. $M[g][v]$ rappresenta l'insieme dei vertici di g che possono fare match con v. Infatti, per definizione, se un vertice v' del grafo g non appartiene a $M[g][v]$, esiste almeno una feature di f tale che $v \in start(f, q)$ e $v' \notin start(f, g)$. Dall'affermazione 1 segue che non esiste alcun subgraph isomorphism ϕ tra q e g che può associare v a v'. Ciò significa inoltre che la coppia (v, v') non può essere coinvola in alcun match. SING è in grado di risolvere sia il problema della First_query_occurrence sia quello della All_query_occurrences. Nel caso del First_query_occurrence, l'algoritmo di match si arresta nel momento in cui è individuato il primo match del grafo.

7.3 Applicazione su dati molecolari

Gli esperimenti sui dati molecolari sono stati eseguiti utilizzando il *DTP AIDS Antiviral Screen dataset* forniti dal the National Cancer Institute (58) contenente circa 42000 composti chimici. Gli esperimenti sono stati realizzati sulla base di tre differenti sottoinsiemi di grafi formati rispettivamente da 8000, 24000 e 40000 grafi, tale che ognuno di questi grafi corrisponde a un grafo i cui nodi rappresentano gli atomi etichettati con il relativo simbolo atomico. Ogni legame semplice o multiplo tra due

atomi è rappresentato mediante un arco singolo. Per ognuno dei tre database è stato inoltre generato un insieme di query secondo la seguente procedura. Si sceglie casualmente un grafo g all'interno del database e uno dei suoi vertici v. A partire da v, si esegue in maniera casuale una ricerca di tipo *breadth-first* fino a quando sia raggiunto un numero totale prefissato di archi. In tal modo sono stati ottenuti 4 differenti gruppi formati da 100 query con dimensioni rispettivamente di 4,6,16 e 32. La Tabella 7.1 e la Tabella 7.2 mostrano rispettivamente i risultati relativi al confronto tra SING, CTree (3) e GCoding (4) rispetto al tempo di preprocessing ed alla dimensione dell'indice. Da tali tabelle si evince che SING genera l'indice più rapidamente di GCoding. Poichè CTree (3) non opera alcuna estrazione di feature dai grafi, è ovvio che il suo tempo di preprocessing risulti di gran lunga inferiore a quello degli altri tool. La Figura 7.4 mostra il numero medio di candidati che superano la fase di filtering rispetto ad un dato gruppo con data dimensione. Le Figure 7.3 e 7.5 confrontano rispettivamente SING in termini di tempo medio di processo della query con CTree (3) e GCoding (4). Come per il test precedente, anche questa tipologia di test è stata realizzata su gruppo di query di una data dimensione. CTree (3) e GCoding considerano rispettivamente la First_query_occurrence e la All_query_occurrences. Di conseguenza, la Figura 7.3 mostra il confronto tra il problema della First_query_occurrence mentre la Figura 7.5 si riferisce al problema delle All_query_occurrences. SING supera tutti gli altri tool nei test effettuati, ad eccezione del test relativo a query di dimensione 4. In questo caso, CTree (3) risulta il migliore tra gli altri tool, anche se i suoi step di filtering e matching scalano meno bene al crescere della dimensione delle query. Scendendo nei dettagli, CTree (3) utilizza un processo di filtering particolarmente dispendioso poiché è basato su un un algoritmo approssimato di subgraph isomorphism. L'algoritmo di matching utilizzato da CTree (3) è una variante dell'algoritmo di Ullmann (59) le cui prestazioni si dimostra essere inferiori a VF2 (5). Il vantaggio di CTree (3) nella fase di preprocessing suggerisce un suo impiego nelle situazioni in cui è richiesta l'esecuzione di un esiguo numero di query. In caso contrario, cioè nelle situazioni in cui è richiesta l'esecuzione di un elevato numero, risulterebbe più appropriato l'uso di SING. La Tabella 7.3 illustra le migliori performance del tool in relazione al numero ed alla dimensione delle query. All'interno della tabella sono riportati i tempi di preprocessing ed il tempo di esecuzione della query.

Il tool gIndex (2) esegue uno step di data mining che risulta generalmente oneroso, caratteristica che si amplifica quando lo si utilizza con grafi di grandi dimensioni: in questo caso non è in grado di eseguirlo. Negli esempi riportati in (2) sono stati eliminati tutti gli atomi H insieme ai loro legami. Per poter confrontare SING con gIndex, è stato

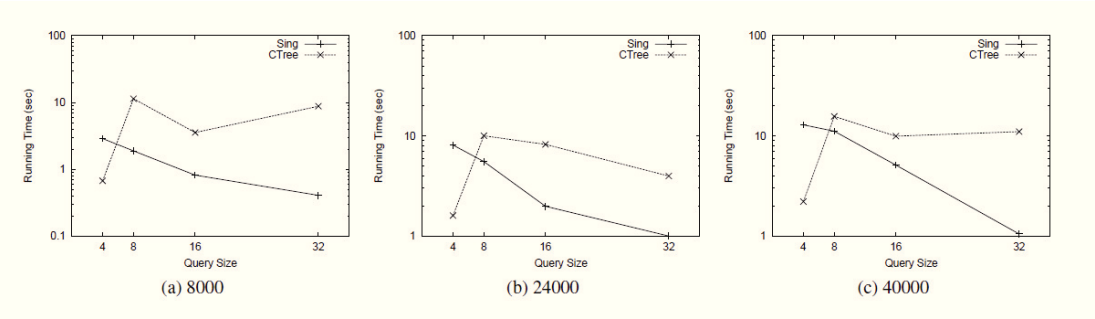


Figura 7.3: Query time relativo al problema denominato First_query_occurrence. - Query time totale su database di composti molecolari. I tool risolvono il problema denominato First_query_occurrence problem.

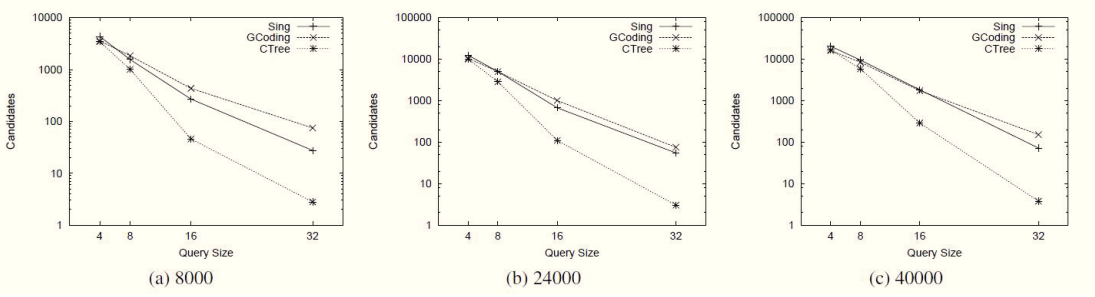


Figura 7.4: Candidati - Numero di candidati relativi ai database di composti molecolari

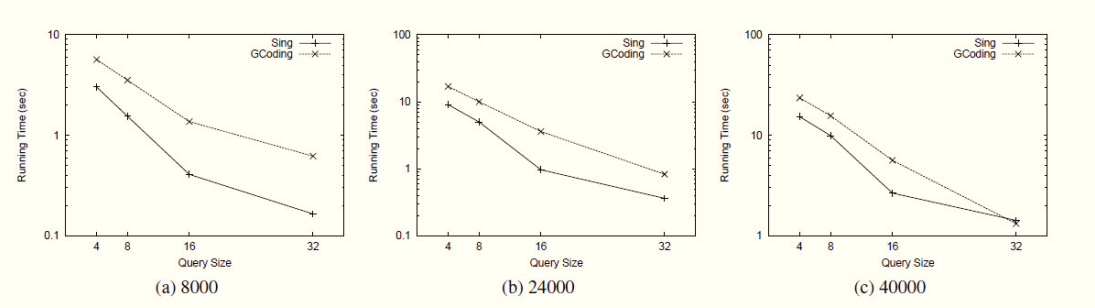


Figura 7.5: Query time relativo al problema All_query_occurrences - Query time totale relativo ai database di composti molecolari. I tool risolvono il problema All_query_occurrences.

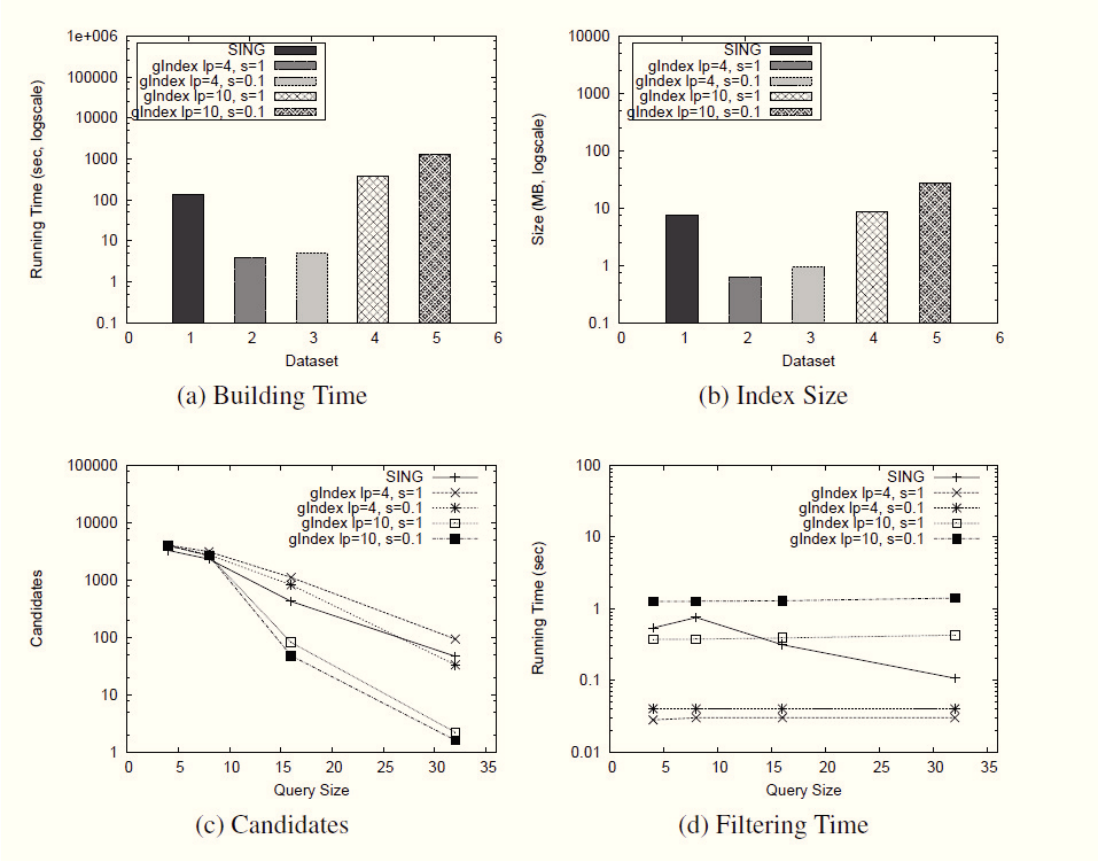


Figura 7.6: Confronto con gIndex - Confronto con gIndex su un dataset di 8000 piccoli composti molecolari.

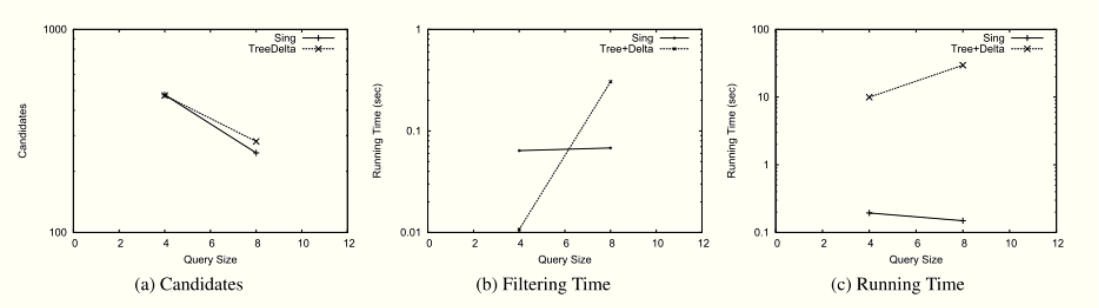


Figura 7.7: Confronto con Tree+Delta - Confronto con Tree+Delta su un dataset di 800 piccoli composti molecolari.

Tabella 7.1: Tempo di Preprocessing su composti molecolari AIDS. I tempi sono espressi in secondi.

Dimensione del database(kb)	CTree	GCoding	SING
8000	8	642	149
24000	25	1948	452
40000	42	2960	755

Tabella 7.2: Dimensione dell'indice relativo al database di composti molecolari di AIDS. La dimensione è espressa in KB.

Dimensione del database	CTree	GCoding	SING
8000	13844	6687	8445
24000	41372	20088	25279
40000	70208	30651	42830

costruito un dataset di piccoli composti molecolari mediante la seguente procedura. Partendo dall'intero database di composti molecolari di AIDS, sono stati scartati 43 grafi aventi più di 250 nodi. Da questo dataset sono stati selezionati 8000 grafi in maniera casuale. gIndex (2) è stato eseguito utilizzando 4 diverse configurazioni. La dimensione massima delle feature è stata fissata a 4 e 10. gIndex (2) utilizza un soglia di supporto che cresce col crescere della dimensione delle feature. La soglia di supporto massima è stata fissata a 0.1 e 1. La Figura 7.6 descrive i risultati ottenuti dove lp descrive la massima dimensione delle feature e s la soglia di supporto massima. In figura non è riportato i tempo totale di query time poichè gIndex (2) non esegue lo step di matching. Fissando $lp = 4$, il building time, la dimensione dell'indice ed il filtering time sono decisamente inferiori ai corrispondenti valori di SING grazie al fatto che gIndex (2) gestisce un numero esiguo di feature. Le performance di filtering decadono moltissimo fissando $lp = 10$ (il più alto numero di candidati) e, fatta eccezione per le query di dimensione 32 con $s = 0.1$, il numero di candidati è sempre più alto rispetto a SING. Con $lp = 10$ e $s = 0.1$, la dimensione dell'indice e confrontabile (7.5 MB di SING contro 8.5 MB di gIndex), ma gIndex (2) richiede un tempo 3 volte maggiore per generarlo (391 secondi contro 135 di SING). Il filtering time di gIndex (2) tende ad essere costante rispetto alla dimensione della query. Rispetto a gIndex (2), SING ha un filtering time più alto su query di piccola dimensione, mentre in SING il filtering time diminuisce col crescere della dimensione delle query. Il filtering power di SING

Tabella 7.3: Il sistema più performante dipende dal numero di query di una data dimensione. Eseprimenti eseguiti sul database AIDS formato da 40.000 composti molecolari. Confronto tra SING, CTree eGCoding.

# di queries	dimensione 4	dimensione 8	dimensione 16	dimensione 32
≤ 70	CTree	CTree	CTree	CTree
71-147	CTree	CTree	CTree	**SING**
148-157	CTree	CTree	**SING**	**SING**
≥ 158	CTree	**SING**	**SING**	**SING**

e gIndex (2) risulta confrontabile su piccole queries. gIndex (2) genera un numero di candidati più piccolo su query più grandi al costo di un tempo di preprocessing e di filtering più elevato. Con $lp = 10$ e $s = 0.1$, gIndex (2) mostra un lieve miglioramento nella pruning power, ma le performance di preprocessing decadono ed il filtering time è più alto: ciò è principalmente dovuto all'elevato numero di feature individuate. E' stato effettuato un confronto anche con Tree+Delta (12), un sistema recentemente proposoto che usa feature basate su trees ed un piccolo insieme di grafi selezionati. Poiché Tree+Delta (12) ha performance migliori rispetto a TreePi è stato deciso di non confrontare SING con TreePi. In maniera analoga a quanto osservato per gIndex (2), Tree+Delta (12) non è in grado di eseguire l'intero dataset di AIDS. Pertanto, al fine di realizzare il confronto con SING, abbiamo generato casualmente un piccolo dataset di 800 composti molecolari a partire dal database AIDS. La dimensione dei grafi presenti all'interno di questo dataset varia da 3 nodi e 4 archi fino a 276 nodi e 284 archi. Il confronto è stato eseguito utilizzando 100 query di dimensione (intesa come numero di archi) pari a 4 e 100 query di dimensione 8. I parametri di Tree+Delta (12) sono stati fissati in accordo con (12). Più precisamente la dimensione massima delle feature, maxL, è stata fissata a 10. La dimensione dell'indice generato da Tree+Delta (12) è 114kb (74kb per SING); il tempo di pre-processing è 11 secondi (stesso risultato per SING). Come riportato in Figura 7.7 il numero di candidati di Tree+Delta (12) è inferiore a quello individuato da SING, fatta eccezione per le query di dimensione 4, per le quali si ottiene lo stesso numero di candidati. Inoltre, SING ha sempre prestazioni migliori di Tree+Delta (12) sul processing time della query. Il filtering time di Tree+Delta (12) è inferiore rispetto a SING per query di dimensione pari 4 e più elevato rispetto a SING per query di dimensione 8. In contrasto a ciò, il filtering time di SING è quasi sempre constante e ciò conferma la buona scalabilità di SING rispetto a Tree+Delta (12). Relativamente alle query di dimensione maggiore a 8, il confronto

non è stato riportato in quanto Tree+Delta (12) richiede in media un elevato tempo di esecuzione.

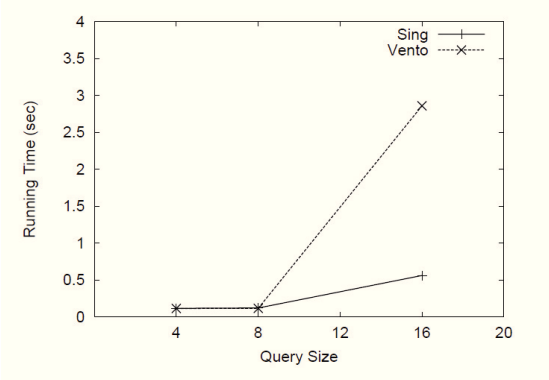


Figura 7.8: Query time relativo ad un singolo grande grafo -

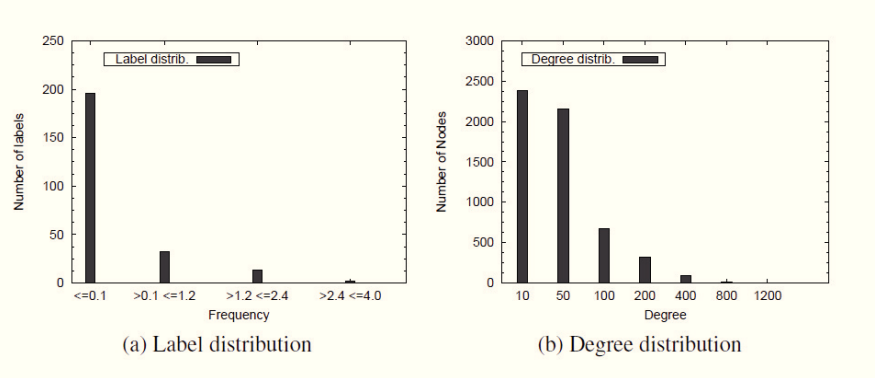


Figura 7.9: Distribuzione etichette e grado in una rete di H. Sapiens - Distribuzione delle etichette nella rete H.Sapiens (clusters) e distribuzione del grado nella rete S.Caeravice.

7.4 Applicazione a reti di regolazione della trascrizione

Al fine di valutare le performance di SING sulle grandi reti biologiche, abbiamo generato un database di reti geniche etichettate con valori discreti delle espressioni geniche, mediante una rete di regolazione della trascrizione di Escherichia Coli annotata con espressioni geniche. Abbiamo estratto la più larga componente connessa presente all'interno dell'intera rete fornita insieme al materiale supplementare da (6). Abbiamo usato i profili dell'espressione genica di 22 esempi dell'esperimento GDS2825 (libera-

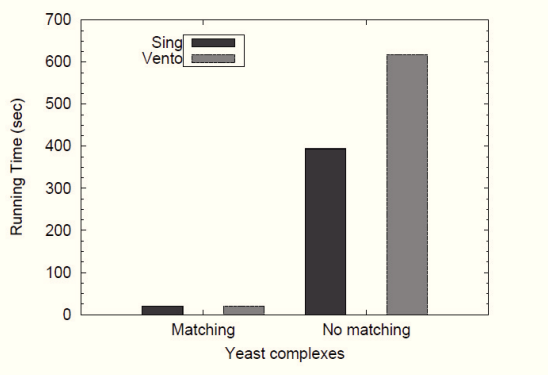


Figura 7.10: Query Time S. Cereavise VS H. Sapiens - Risultati relativi alla ricerca di un insieme di complessi di rispetto alla rete di interazione proteina-proteina di H. Sapiens.

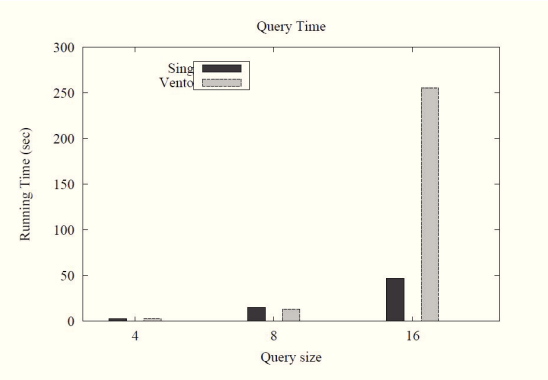


Figura 7.11: Risultati relativi alla rete di S. Cereavise. - Query time relativo alla rete di S. Cereavise (casuale, 8 etichette)

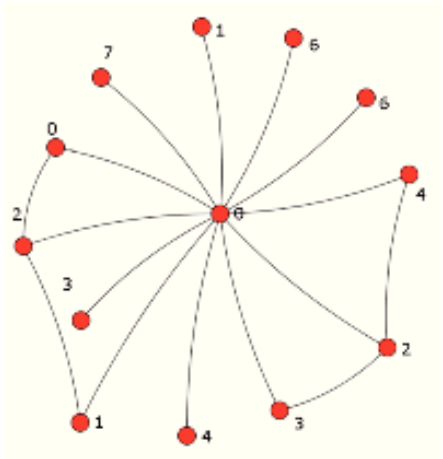


Figura 7.12: Un esempio di query senza match per VF2 -

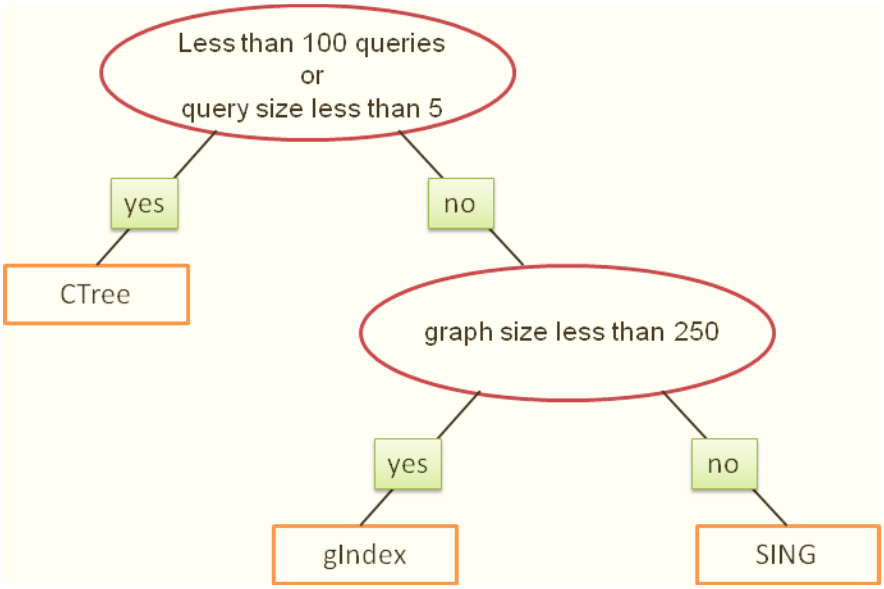
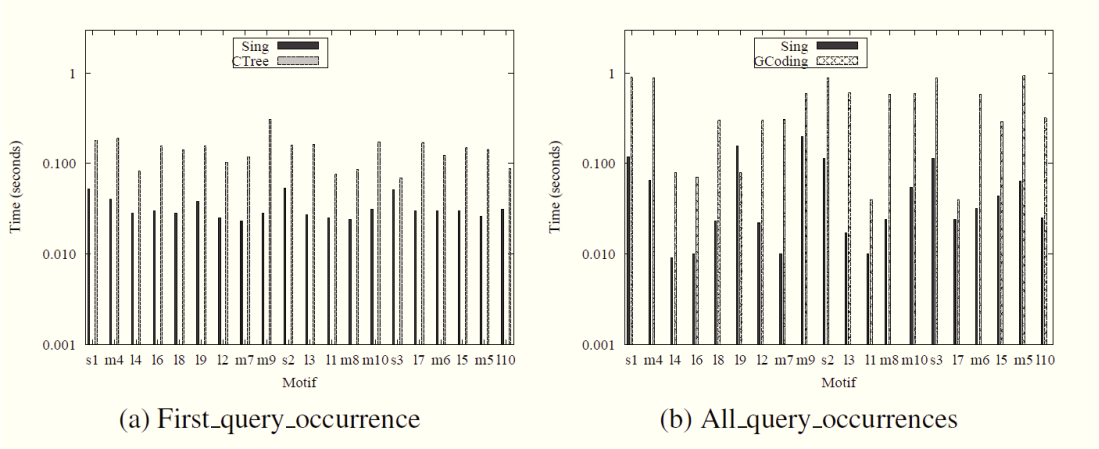


Figura 7.13: Abero descisionale del sistema - L'albero decisionale permette di individuare il sistema più efficiente in base a differenti scenari. SING rappresenta la scelta migliore nel caso in cui sia richiesta l'esecuzione di un elevato numero di query su database di grafi di media e grande dimensione.

mente disponibile in NCBI (60)). Infine, abbiamo discretizzato ogni valore relativo all'espressione genica mappandolo all'interno di un insieme di 5 differenti livelli, etichettati come very low, low, medium, high, very high. Tali livelli sono stati assegnati come etichette ai rispettivi nodi della rete di regolazione. Seguendo le indicazioni di Alon et. Al. (6), abbiamo tentato di identificare i gruppi di nodi connessi mediante una specifica topologia ed annotati con particolari profili del livello dell'espressione genica. Si potrebbe utilizzare questo approccio per comprendere il meccanismo di regolazione di un gene, verificando se un dato pattern sia presente in un dato insieme di esempi in cui esso appare e quali geni coinvolge. Abbiamo interrogato il database con un insieme di motif etichettati con il livello di espressione genica utilizzando i motif individuati da U. Alon et. al. (6). Ogni vertice è stato etichettato con un livello di espressione genica *very high*. Il tempo di preprocessing e la dimensione dell'indice sono riportati in Tabella 7.4. La Figura 7.6 mostra il tempo totale per il processamento di ogni query, denominate come in (6). SING supera tutti gli altri metodi relativamente al tempo di processamento della query.



(a) First_query_occurrence (b) All_query_occurrences

Figura 7.14: Query time relativo a reti biologiche -

Tabella 7.4: Tempo di Preprocessing e dimensione dell'indice su grandi reti biologiche

Tool	Dimensione dell'indice(KB)	Tempo di Preprocessing (sec)
GraphFind	1252	23
CTree	292	1.3
GCoding	85	101

7.5 Applicazione a reti di interazione proteina-proteina

La maggior parte dei tool di graph mining incontrano non poche difficoltà a manipolare le reti di interazione proteina-proteina (2, 10). Al fine di illustrare le performance di SING su questa tipologia di reti, abbiamo utilizzato l'intera rete di interazione proteina-proteina di Homo Sapiens formata da 7824 nodi e 28303 archi. L'insieme di query utilizzate è stato ottenuto a partire dai 282 complessi proteici di S. Cereavice. I nodi delle proteine sono stati ottenuti mediante SGD(61) e gli archi da BioGRID (62). Il primo passo è stato quello di eseguire una all-pair-BLAST sull'insieme di proteine di S. Cereavice e Homo Sapiens per poter eseguire successivamente una loro clusterizzazione mediante un clustering di tipo *single-linkage*. Al fine di evitare che proteine dissimili possano essere raggruppate all'interno dello stesso cluster, abbiamo applicato una *score cutoff* di 40 bit. Infine è stato fissato a 100 la dimensione massima di un cluster. La rete di H.Sapiens ed i complessi di S.Cereavice sono stati quindi etichettati con l'identificativo del cluster di appartenenza, ottenendo in tal modo 1251 differenti etichette. La distribuzione delle etichette in questa rete è descritta in Figura 7.9b.Si noti che, approssimativamente 200 etichette mostrano una frequenza pari a circa 0.1%:

ciò significa che il 20% delle etichette appare soltanto 5 volte circa nella rete). Sono stati rimossi tutti i complessi con una numero di archi inferiore a 2 e si è ottenuto in tal modo un insieme di 209 complessi di S.Cereavice la cui dimensione varia tra i 2 ed i 1290 archi (che corrisponde al complesso denomianto *transcription factor*). I tool basati su feature come gIndex (2), Tree+Delta (12) non riescono ad operare su grafi così grandi poiché non sono stati progettati per tale scopo. Pertanto, in questo caso, il confronto è stato realizzato direnttamete con VF2 (5). La rete di Homo Sapiens è stata pre-processata da sing in 613 secondi producendo un indice di 9556 kb. Successivamente sono state eseguite le query sulla rete target utilizzando i complessi di S. Cereavice. Sono stati individuati dal tool 7 complessi di S. Cereavice caratterizzati da almeno un match nella rete di H.Sapiens. La Tabella 7.5 descrive il tempo totale di processo della query eseguita da SING e VF2 per l'esecuzione dell'insieme delle 7 query e dell'insieme delle query rimanenti (i complessi che non hanno un match in H. Sapiens). Facendo seguito alle attese, SING è migliore di VF2 quando è applicato a query che non sono presenti nella rete target. Nel caso in cui è applicato a query caratterizzate da almeno un match, SING e VF2 sono quasi equivalenti in termini di tempo di esecuzione della query. La Figura 7.10 illustra il tempo totale di processo della query richiesto da SING e VF2 sia per un insieme di query per le quali è individuato un match sia un insieme di query per le quali non è individuato alcun match. Anche in questo caso SING è migliore di VF2 quando risulta applicato a query che non hanno match nella rete target, mentre hanno un comportamento simile in caso contrario. Ciò si spiega perchè il numero di etichette (1251) confrontato al numero di nodi (7824) rendono la localizzazione banale sia per VF2 sia per SING. Al fine di confrontare gli algoritmi in un ambito più complesso, abbiamo considerato l'intera network di S.Cereavice (scaricata da BioGRID (62)). Tale rete contiene 5589 nodi and 92835 archi. La Figura 7.9a descrive la distribuzione del grado per questa rete. Sono state infine assegnate ai nodi della rete 8 differenti etichette in maniera casuale per migliorarne la distribuzione. L'insieme di query utilizzato è formato da 100 query di dimensione 4 e 8 e 50 query di dimensione 16 estratte in modo casuale dalla rete. A causa dell'ovvio elevato numero di match si è deciso di arrestare l'esecuzione del tool al raggiungimento di 100 match. La Figura 7.11 descrive la performance di SING e VF2 in relazione alle query di dimensioni e 4, 8 e 16. SING e VF2 sono confrontabili per le prime due dimensioni di query, mentre per le restanti SING supera VF2. Analizzando i risultati più in dettaglio, troviamo un caso interessante che mette in evidenza il differente comportamento dei due tool confrontati. Per una query di dimensione 16, VF2 necessita di circa 2,3 ore per eseguirla, mentre SING è in grado di processarla in appena 8 secondi. Questa

Tabella 7.5: Query time (in sec.) dei complessi che fanno match nell'esperimento H.Sapiens - S. Ceravisiae

QueryName	VF2	SING
CAF	2,928	2,982
Elg1 RFC like	2,984	2,996
Piccolo NuA4 histone acetyltransferase	2,959	2,980
SNARE 31201	2,952	2,983
methionyl glutamyl tRNA synthetase	2,960	3,032
nascent polypeptide associated	2,952	2,976
tubulin	3,034	2,994

query è rapprensentata in Figura 7.12.Questa query è composta da appena 13 nodi e 16 archi, ma vi è un nodo hub centrale collegato mediante 12 archi agli altri nodi del grafo. Quando VF2 cerca di trovare un match tra il nodo centrale della query con un hub nella rete target (definiamola H), l'esplorazione combinatoriale dei suoi vicini può risultare molto costosa. Viceversa, in molti casi SING può verificare in anticipo che i due hub non possono coincidere. Per esempio, se per il path 0-4-2-3-, contenuto nella query partendo dal nodo centrale, non è possibile individuare un path corrispondente nel grafo target che comincia da H, SING scarta il match e continua, impedendo la costosa verifica dei suoi vicini.

Applicazione su dati sintetici

Per verificare le performance di Sing su un grande grafo, è stata generata una rete scale free di 2000 nodi e 4000 archi. La procedura utilizzata per creare questa rete è la seguente: partendo da un arco iniziale, ogni nuovo arco è connesso ad uno esistente con una probabilità proporzionale al grado del nodo. Ciò garantisce che la rete prodotta goda di una distribuzione del grado che segue la *power low* e che sia quindi *Scale Free*. Otto differenti etichette sono stati assegnate ai nodi della rete in maniera casuale su distribuzione uniforme. Sono stati generati 3 differenti insiemi di query rispettivamente di 4, 8 e 16 archi, generati casualmente seguendo la stessa procedura descritta in precedenza. In questo caso SING è stato confrontato solo con VF2 (5) in quanto la fase di filtering non ha senso essendo presente un solo grafo nel database ed inoltre la fase di verifica è eseguita proprio utilizzando VF2 (gli altri algoritmi che non usano VF2 hanno ovviamente performance peggiori) SING ha generato un indice di 1.2 MB in 35 second ed il query time è riportato in Figura 7.8.Possiamo osservare come per query

di piccole dimensioni SING and VF2 hanno un andamento simile mentre per query di dimensione 16, SING supera VF2.

7.6 Efficienza ed efficacia del metodo

In questo capitolo abbiamo discusso delle problematiche relative al match esatto e descritto un nuovo algoritmo per la ricerca all'interno di grafi denominato SING. Confrontandolo con i tool più accreditati nello stesso ambito di ricerca si è rilevato che SING, durante la fase di ricerca, è in grado di scartare in maniera ottimale i grafi che non contengono il grafo query. Abbiamo dimostrato che CTree (3) dovrebbe essere usato solo nel caso in cui si esegue un numero ridotto di query (ad esempio 100); oltre questa soglia è conveniente utilizzare SING. Inoltre su database di grafi di piccole dimensioni (al di sotto di 250 nodi), si evince che gIndex (2) ha prestazioni migliori anche se a scapito delle dimensione dell'indice. Infine, SING è senz'altro il miglior tool nel caso si esegua un elevato numero di query su database formati da grafi di grandi dimensioni. La Figura 7.13 descrive un albero decisionale che suggerisce quale sistema utilizzare in corrispondenza del particolare scenario fornito.

8

Un algoritmo per la ricerca inesatta di sottostrutture in grandi reti biologiche: SIGMA

8.1 Introduzione a SIGMA

Nel capitolo precedente, trattando il problema del match esatto, è stata evidenziata la significativa disponibilità di tool atti a risolvere tale problema. Per quanto riguarda, invece, il problema della ricerca inesatta di sottografi, possiamo affermare che sono disponibili pochissimi tool che lo risolvono (15). Definita *indel* la generica operazione di inserimento o rimozione di un arco in un dato grafo, il problema della ricerca inesatta può essere altresì tradotto come il problema di individuare i grafi target che sono isomorfi al grafo query a meno di un numero massimo predefinito di indel. In realtà questo problema può essere ristretto alla gestione esclusiva della rimozione degli archi. Infatti, gli inserimenti, ovvero gli archi aggiuntivi nel grafo target che non sono presenti nella query, possono essere scartati poichè apportano soltanto un miglioramento esclusivamente qualitativo del match. Ulteriori varianti al problema proposto sono rappresentate dalla possibilità di consentire *mismatch* a livello di etichette, l'inserimento e la rimozione dei nodi. Un ambito di applicazione della ricerca inesatta è rappresentato dall'identificazione di farmaci che curano particolari patologie o che possono presentare effetti collaterali conosciuti, nel caso in cui la struttura molecolare responsabile di una particolare attività o effetto collaterale sia nota. La Figura 8.1 mostra come le molecole antidepressive come l' L-triptofano condividono composti con alcaloidi, ammine isolate dalle piante, che includono veleni come la stricnina e potenti allucinogeni come

l'LSD. Sono evidenziate le porzioni condivise. Rimuovendo 7 archi dall'L-triptofano, il composto rimanente presenta un match con la Stricnina e 5 delezioni sono necessarie per trovare un match con l'LSD. Infatti, in (63) è spiegato che sia l'L-triptofano sia l'LSD sono coinvolti nella sindrome da serotonina e che la stricnina produce simili sintomi in quanto coinvolta nella diagnosi differenziale. SIGMA (Set-cover-based

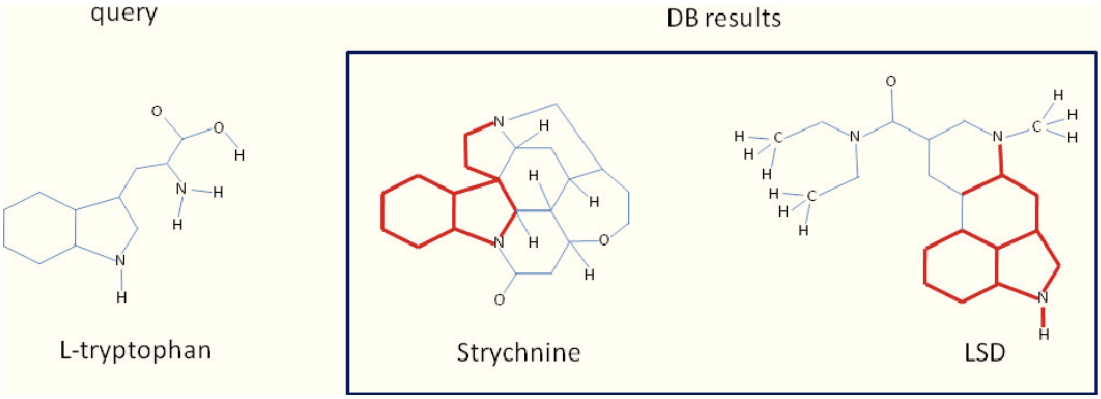


Figura 8.1: Un esempio di match inesatto su composti molecolari. - I composti sono rappresentati come grafi nei quali i vertici sono gli atomi etichettati con il simbolo del rispettivo elemento. I vertici privi di etichetta corrispondono ad atomi C, e gli archi sono legami (i legami doppi sono rappresentati mediante un singolo arco). La parte colorata in rosso della stricnina e dell'LSD presentano un match con una part della struttura del'L-triptofano. Pertanto, la ricerca di match equivale ad identificare le molecole che presentano proprietà chimiche condivise.

Inexact Graph Matching Algorithm) è un algoritmo efficiente di filtering basato su feature per la ricerca inesatta su grafi. L'algoritmo si basa sull'associazione di un insieme di feature ad ogni arco della query e sulla ricerca della collezione di tali insieme la cui rimozione permette un match esatto della query sul dato grafo. Pertanto il problema proposto può essere affrontato come il problema di ricoprimento delle feature mancanti del grafo mediante *multiset* sovrapposti. Di seguito sarà descritta questa variante del problema del *set cover* e sarà fornita una sua approssimazione *greedy*. SIGMA è stato testato estensivamente eseguendo query costituite da piccole molecole su un database di composti molecolari. Si è provveduto a confrontare il tool proposto con il tool denominato Grafil (15) che costituisce lo stato dell'arte per tale tipologia di problema ed è stato dimostrato che SIGMA è caratterizzato da filtering power migliore e che migliora ulteriormente al crescere della dimensione della query. Per dimostrare l'utilità di SIGMA in un contesto biologico reale, abbiamo eseguito query di S. Cerevisiae su complessi proteici di H. Sapiens. Pur esistendo altri metodi per l'esecuzione di query

su complessi proteici, come ad esempio Torque (64) e QNet (65), questa è la prima applicazione che utilizza tecniche di indicizzazione per risolvere tale tipologia di problemi. In contrapposizione agli altri metodi, SIGMA è in grado di ricercare i match che sono topologicamente simili alla query, senza utilizzare informazioni sull'omomorfismo tra le due specie, come accade in QNet o senza tenere in considerazione l'esatta topologia, come in Torque.

8.2 Descrizione dell'algoritmo

Il contributo di SIGMA nell'ambito della ricerca inesatta all'interno di reti biologiche può essere riassunto in tre principali punti:

- definiamo una nuova pruning rule per il match inesatto basata su *multiset multi-cover*, che rappresenta una variante del più noto problema del set-cover;
- forniamo una approssimazione *tight greedy* per il multiset multi-cover, che svolge un ruolo chiave nel garantire efficienza e efficacia del pruning;
- valutiamo le performance del metodo proposto, effettuando dei confronti con l'approccio che rappresenta lo stato dell'arte, su database di composti molecolari. Inoltre, applichiamo tale metodo ad un confronto sistematico tra complessi di S. Cerevisiae e H. Sapiens.

Definiamo *multiset* una coppia (A, m) in cui A è un insieme m è una funzione da A all'insieme ed $\mathbb{N}$ dei numeri naturali. Diciamo che $m(a)$ rappresenta la molteplicità dell'elemento a. Dato un insieme U, diciamo che $A' = (A, m)$ è un multiset di U se $A \subseteq U$. Per semplicità, estendiamo la funzione $m()$ a tutti gli elementi di U definendo $m(u) = 0$ per ogni $u \in U - A$. Definiamo la cardinalità di un multiset $A' = (A, m)$ come $|A'| = \sum_{a \in A} m(a)$ Siano $A' = (A, m)$ e $B' = (B, n)$ due multiset. Definiamo la differenza $A' - B'$ come l'insieme $C' = (C, p)$ dove $C = \{c \in A | m(c) > n(c)\}$ e $p(c) = m(c) - n(c)$ per ogni elemento $c \in C$. Definiamo l'intersezione $A' \cap B'$ come l'insieme $C' = (C, p)$ dove $C = A \cap B$ e $p(c) = min(m(c), n(c))$ per ogni elemento $c \in C$. Definiamo l'unione $A' \cup B'$ come l'insieme $C' = (C, p)$ dove $C = A \cup B$ and $p(c) = m(c) + n(c)$ per ogni elemento $c \in C$. Diciamo che $A' \subseteq B'$ se per ogni $a \in A$ abbiamo $a \in B$ e $m(a) \leq n(a)$. Dato un multiset C e due multisets $A, B \subseteq C$, è facile verificare che risultano soddisfatte le seguenti relazioni:

- $C - (C - A) = A$
- $C - A \subseteq C - B \Leftrightarrow B \subseteq A$

8.3 Una tecnica di filtering innovativa per il match inesatto

In questa sezione sviluppiamo delle efficienti regole di pruning che risolvono il problema dell'inexact matching. Iniziamo focalizzando il seguente problema: data una query Q ed un grafo G, Q ammette un match inesatto in G con al più r delezioni? Lo schema proposto è basato sull'associazione di un insieme di feature F_e ad ogni arco e della query (i.e., l'insieme di feature che contiene l'arco) e sulla ricerca di una collezione di tali insiemi la cui rimozione consente l'exact matching della query Q con il grafo G. Pertanto, il problema che ne deriva può essere formulato come un problema di set cover: dato un insieme Y (di feature di Q che non sono presenti in G) ed una famiglia di insiemi S (costituiti da feature associati ad ogni arco), ricercare la più piccola sottofamiglia Γ di S che ricopre Y, i.e., $\bigcup_{X\in\Gamma} X \supseteq Y$. Tale sottofamiglia rappresenta un insieme di archi query la cui eliminazione assicura che tutte le feature di Q sono contenute in G. Se una sottofamiglia Γ di dimensione r non esiste, possiamo assumere che se eliminiamo r archi in tutti i modi possibili, possiamo sempre trovare almeno una feature della query che non è contenuta nel grafo, quindi il grafo può essere scartato. Possiamo rafforzare la precedente impostazione considerando la molteplicità delle occorrenze delle feature. Sia $E_\gamma \subseteq E(Q)$ un sottoinsieme degli archi della query. Denotiamo con F_Q il multiset delle feature di Q e con F_{E_γ} il multiset delle feature che contengono almeno uno degli archi in E_γ. Se Q ammette un inexact match in G con r delezioni, deve esistere un insieme di archi E_γ di dimensione-r tale che $F_Q - F_{E_\gamma} \subseteq F_G$. Quindi può essere dedotta la seguente pruning rule:

Regola di pruning 1 *Data una query Q con r delezioni ammesse, un grafo G può essere scartato se per ogni $E_\gamma \subseteq E(Q)$ con $|E_\gamma| = r$ abbiamo:*

$$F_{E_\gamma} \not\supseteq F_Q - F_G$$

Chiaramente, questa pruning rule non può essere applicata efficientemente perchè il numero di possibili r-sottoinsiemi di $E(Q)$ cresce esponenzialmente con r, e la regola deve essere verificata per tutti i grafi presenti nel database. Per ovviare a tale problema, utilizziamo un approccio di tipo *multiset multi-cover* e definiamo una nuova pruning rule basata su un algoritmo di tipo *greedy*. Nel problema multiset multi-cover $Y = (Y', m_Y)$ è un multiset e S una famiglia di multiset. Ogni elemento (feature) f di Y possiede una molteplicità $m_Y(f)$ che specifica il numero di volte in cui f è stato ricoperto, e si trova in ogni insieme X di S con una data molteplicità $m_X(f)$. L'obiettivo diventa dunque quello di trovare l'insieme di dimensione minima Γ tale che $\bigcup_{X\in\Gamma} X \supseteq Y$,

i.e., per ogni $f \in Y'$, $\sum_{X \in \Gamma} m_X(f) \geq m_Y(f)$. Nella sua formulazione generica, un insieme di S può essere scelto molteplici volte (anche Γ è un multiset). Nel seguito consideriamo l'ulteriore vincolo che ogni insieme di S può essere scelto al più una volta. Ne nostro caso, il multiset da ricoprire è $Y = F_Q - F_G$, e la collezione di multisets ricoprenti è $S = \{F_e\}_{e \in E(Q)}$. Se Y non ammette multiset multi-cover di dimensione r allora G può essere scartato (vedi Figura 8.2).Il problema del set Set-cover è di tipo NP-completo (66), ma può essere risolto mediante una semplice euristica greedy con un rapporto di approssimazione $H(max\{|X| : X \in S\})$, dove $H(n) = 1 + 1/2 + ... + 1/n$ (66, 67). E' stato dimostrato che il problema più generale del multiset multi-cover ammette lo stesso rapporto di approssimazione (68).

La Figura 8.3 descrive una euristica greedy per il problema del multiset multi-cover. Ad ogni iterazione, l'algoritmo sceglie il multiset X della famiglia S che massimizza il numero delle più recenti occorrenze di feature di Y che sono state ricoperte. L'insieme scelto è aggiunto al ricoprimento ed i suoi elementi sono rimossi da Y. E' essenziale disporre di un *lower bound* stretto della soluzione ottimale al fine di avere un algoritmo greedy che esegua il filtering in maniera efficiente. Sia $Y = (Y', m_Y)$ il multiset di feature da ricoprire. Sia $cost(f, i)$ una funzione definita da $Y' \times \mathbb{N}$ a $\mathbb{R}$, che assegna un costo ad ogni occorrenza di feature coperta dall'algoritmo greedy. Le occorrenze delle feature sono ordinate per il numero di volte che esse sono coperte dall'algoritmo. Il costo è assegnato ad ogni passo (esecuzione del ciclo while) dell'algoritmo, distribuendo una unità di costo su tutte le occorrenze delle feature che sono state coperte. In altre parole ad ogni occorrenza di feature è assegnato un costo $1/c$, dove c rappresenta in numero di occorrenze di feature recentemente coperte. La funzione *cost* è definita formalmente come segue: Sia $new\_cov(f, s)$ il numero di occorrenze of f recentemente coperte al passo s e $cov(f, s)$ sia il numero totale di occorrenze coperte di f dopo il passo s, i.e. $cov(f, s) = \sum_{t=1...s} new\_cov(f, t)$. La funzione *cost* è definita come:

$$cost(f, i) = \begin{cases} \dfrac{1}{\sum_{f \in Y'} new\_cov(f, s)} & \text{if } cov(f, s-1) < i \leq cov(f, s) \\ 0 & \text{otherwise} \end{cases}$$

Sia Γ la copertura restituita dall'algoritmo greedy, Γ^* la copertura esatta e $r_X(f) = min(m_X(f), m_Y(f))$. Il seguente teorema limita la dimensione della copertura restituita dall'algoritmo greedy.

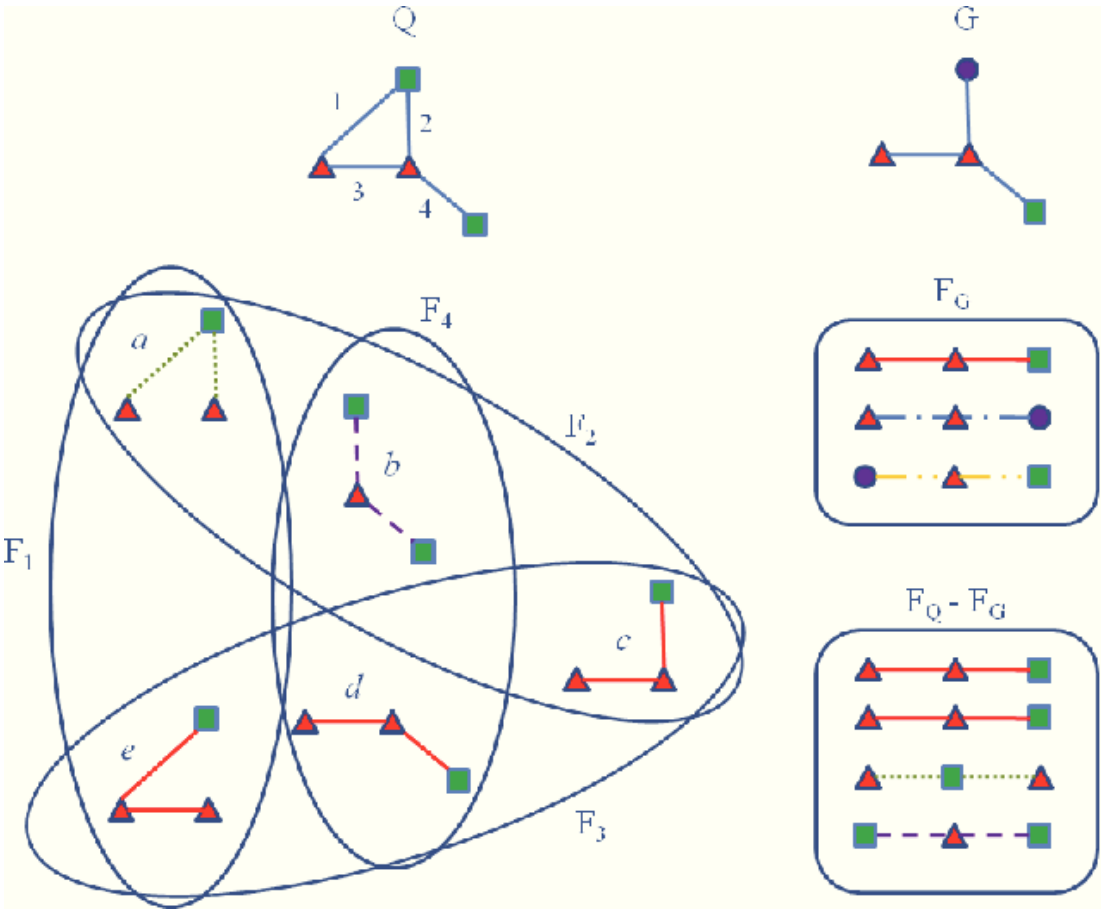


Figura 8.2: Esempio multiset multi-cover - Un esempio di query Q ed un grafo G che contiene una copia di Q con due delezioni. Consideriamo feature tutti i sottografi connessi contenenti esattamente due archi. Sinistra: Q e tutte le occorrenze di feature che essa contiene(F_Q). La tipologia di linea delle occorrenze delle feature è scelta in base alla feature a cui essa corrisponde. Ogni insieme F_i indica tutte le occorrenze della feature che contengono l'arco i. Destra: G, il suo multiset di feature (F_G) ed il multiset di feature mancanti ($F_Q - F_G$). Il ricoprimento minimo $F_Q - F_G$ dalla famiglia $\{F_1, F_2, F_3, F_4\}$ ha cardinalità 2, e ciò implica che almeno due delezioni sono necessarie perché ci sia un match. $\{F_1, F_2\}$ è un possibile covver, e ciò implica che G è candidato ad essere un match per Q con gli archi 1 2 eliminati.

```
Greedy-Multiset-Multicover(Y, S)
    Γ ← φ
    whileY ≠ φ do
        X ← argmax_{X̄∈S} |X̄ ∩ Y|
        Y ← Y − X
        Γ ← Γ ∪ {X}
    return Γ
```

Figura 8.3: Un algoritmo greedy per il problema del multiset multi-cover

Teorema 3 *Sia* $\alpha(f) = cost(f, m_Y(f))$ *e* $\beta = \sum_{f \in Y'} \sum_{i=1}^{m_Y(f)} (\alpha(f) - cost(f, i))$ *allora,*

$$|\Gamma^*| \geq \min_{\Gamma' \subseteq S : \sum_{(X, m_X) \in \Gamma'} \sum_{f \in X} r_X(f)\alpha(f) - \beta \geq |\Gamma|} |\Gamma'|$$

Dimostrazione 1 *Si dimostra che*

$$\sum_{(X, m_X) \in \Gamma^*} \sum_{f \in X} r_X(f)\alpha(f) - \beta \geq |\Gamma|$$

poiché $\Gamma^* \subseteq S$ *ed ogni elemento di un insieme è sempre più grande o uguale al minimo sopra quell'insieme. Il costo totale assegnato a tutte le occorrenze delle feature è pari a* $|\Gamma|$*. Quindi:*

$$\begin{aligned}
|\Gamma| &= \sum_{f \in Y'} \sum_{i=1}^{m_Y(f)} cost(f, i) \\
&= \sum_{f \in Y'} m_Y(f) \cdot cost(f, m_Y(f)) \\
&\quad - \sum_{f \in Y'} \sum_{i=1}^{m_Y(f)} (cost(f, m_Y(f)) - cost(f, i)) \\
&= \sum_{f \in Y'} m_Y(f)\alpha(f) - \beta \\
&\leq \sum_{(X, m_X) \in \Gamma^*} \sum_{f \in X} r_X(f)\alpha(f) - \beta.
\end{aligned}$$

In base al precedente teorema, otteniamo la seguente pruning rule:

Regola di pruning 2 *Sia data una query* Q *con* r *delezioni ammesse ed un grafo* G*. Sia* $|\Gamma|$ *la copertura restituita dall'algoritmo greedy quando è eseguito su* $F_G - F_Q$*.* G

può essere scartato se:

$$r < \min_{\Gamma' \subseteq S : \sum_{(X,m_X)\in\Gamma'} \sum_{f\in X} r_X(f)\alpha(f) - \beta \geq |\Gamma|} |\Gamma'|$$

La parte destra può essere facilmente calcolata classificando l'insieme S mediante il punteggio $\sum_{f\in X} r_X(f)\alpha(f)$ in ordine descrescente e prendendoli uno alla volta fino a quando la somma dei punteggi è più grande o uguale a $|\Gamma| + \beta$.

8.3.1 Un tentativo di incrementare la capacità del filtering

L'uso dei multiset presi singolarmente non riesce a catturare l'interdipendenza presente di tra essi, infatti, due o multiset di feature possono includere la stessa occorrenza di feature ma nella copertura possiamo contarla due volte (vedi la Figura 8.4)A tal fine introduciamo a nuova variante del problema del set-cover, che definiamo *Multi-cover by Overlapping Multisets* (MOM). Sia U un insieme di elementi (occorrenze di feature), $\mathcal{F}$ un insieme di feature ed f una funzione che associa ad ogni elemento di U una feature in $\mathcal{F}$. Con $A \subseteq U$, definiamo la copertura di A, che denotiamo con $Cov_f(A)$, come il multiset $D' = (D, m)$ di $\mathcal{F}$ tale che $D = \{f(a)|a \in A\}$ and $m(d) = |\{a \in A|f(a) = d\}|$. Definiamo il problema MOM come segue: Per un multiset Y of $\mathcal{F}$ ed una data famiglia S di sottoinsiemi di U, cerchiamo la più piccola sottofamiglia Γ di S così che $Cov_f(\bigcup_{X\in\Gamma} X) \supseteq Y$. In Figura 8.3 si può notare come la minima copertura per MOM è $\{F_1, F_6, F_7\}$, così che G non è un candidato a fare match con Q con al più due delezioni. Può essere dimostrato che anche questo problema è NP-hard per riduzione dal problema del set-cover. In Figura 8.5 è descritto un algoritmo greedy per questo problema. In tale algoritmo per MOM in Figura 8.5 un ulteriore insieme Z è utilizzato per tenere traccia degli elementi coperti. Quando si aggiunge un insieme alla copertura, i suoi elementi sono rimossi da Z per evitare di considerarli due volte. Adesso possiamo definire una nuova pruning rule basata su MOM che è equivalmente alla pruning rule 1.

Regola di pruning 3 *Sia data una query Q con r delezioni. Denotiamo come F_e l'insieme delle occorrenze di feature di Q che contengono l'arco $e \in E(Q)$. Possiamo scartare un grafo G se per ogni $E_\gamma \subseteq E(Q)$ di dimensione r:*

$$Cov_f(\bigcup_{e\in E_\gamma} F_e) \not\supseteq F_Q - F_G$$

Poiché $Cov_f(\bigcup_{e\in E_\gamma} F_e) = F_{E_\gamma}$ otteniamo che:

Teorema 4 *La regola di pruning 3 è equivalente alla regola di pruning 1.*

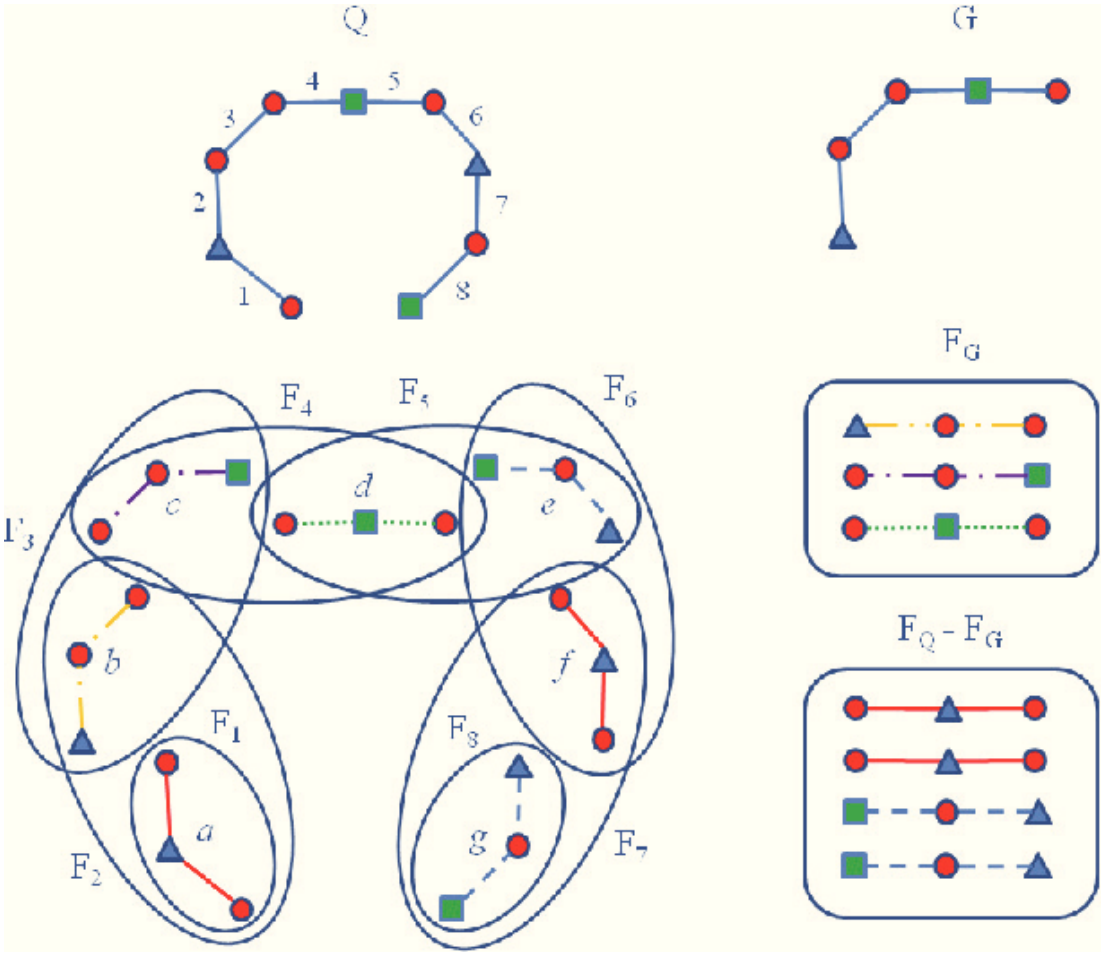


Figura 8.4: Un grafo che non può essere scartato per risolvere il problema del multiset multi-cover - Si ricerca un match inesatto con al più due delezioni. Le features sono sottografi contenenti esattamente due archi connessi. La parte sinistra mostra la query Q a tuttle le sua occorrenze di feature (F_Q). Il tipo di linea di una occorrenza di feature è unicamente associata a quella feature. Ogni insieme F_i indica che tutte le occorrenze di feature contengono l'arco i. La parte destra mostra il grafo target G, il suo multiseet di feature (F_G) e il multiset di feature mancanti ($F_Q - F_G$). Per il problema del multiset multi-cover, $\{F_6, F_7\}$ è una copertura di $F_Q - F_G$ in quanto la feature f è calcolata due volte. Questo significa che Q è candidato a fare match con G con 2 delezioni. Se consideriamo f soltanto una volta (vedi la definizione MOM seguente) la copertura minima dovrebbe essere $\{F_1, F_6, F_7\}$ e G sarebbe scartato

Greedy-MOM(Y, S)
$Z \leftarrow U$
$\Gamma \leftarrow \phi$
while$Y \neq \phi$ do
$X \leftarrow argmax_{\overline{X} \in S} |Cov_f(\overline{X} \cap Z) \cap Y)|$
$Y \leftarrow Y - Cov_f(X \cap Z)$
$Z \leftarrow Z - X$
$\Gamma \leftarrow \Gamma \cup \{X\}$
return Γ

Figura 8.5: Un algoritmo greedy per MOM.

Il teorema 3 e la regola di pruning 2 sia applicano anche all'algoritmo greedy MOM, così che può essere utilizzato lo stesso lower bound utilizzato per scartare i grafi.

8.4 Implementazione

Sono state implementate due distinte versioni del tool SIGMA: la prima è basata sulla impostazione del multiset multi-cover e la seconda è basata sulla impostazione MOM. Entrambi i tool utilizzano Edge come primo passo di pruning e quindi applicano la pruning rule 2. Entrambi i tool sono confrontati con una nostra implementazione di Edge e Grafil (15) (che include Edge come parte di filtering). Al fine di realizzare una analisi quanto più uniforme possibile, sono stati utilizzati path di lunghezza massima 4 come feature per i sistemi confrontati. La verifica dei candidati è stata realizzata calcolando tutti i possibili sottografi della query che possono essere ottenuti eliminando ogni insieme di r archi ed eseguendo un efficiente algoritmo di subgraph isomorphism denominato VF2(69) su tutti i grafi.

8.5 Ricerca inesatta di complessi proteici di S. Cerevisiae in un database di complessi proteici di H. Sapiens

I dati relativi ai complessi proteici di S. Cerevisiae e di H. Sapiens contengono una rappresentazione basata su grafi dell'insieme dei complessi di ogni specie, in cui i vertici corrispondono alle proteine e gli archi alla interazioni proteina-proteina. I complessi di H. Sapiens sono stati ottenuti da CORUM (70), mentre i complessi di S. Cerevisiae da SGD (61). La topologia di ogni complesso è stata ricavata dalla rete di interazione

proteina-proteina ottenuta da BioGRID (62). L'assegnazione delle etichette ai vertici/proteine è avvenuta eseguendo una *all-pair BLAST* sulle proteine di S. Cerevisiae e H. Sapiens, e successivamente clusterizzandole in accordo la punteggio di BLAST. A questo scopo, abbiamo utilizzato un clustering gerarchico di tipo *average-linkage* con uno score di cutoff pari a 40 bits ed una soglia massima di clustering pari a 500. Eseguendo tale procedura abbiamo ottenuto 6703 clusters. Ogni proteina è stata pertanto etichettata con l'id del cluster di appartenenza. Escludendo i complessi privi di archi, abbiamo ottenuto un insieme di 785 complessi di H. Sapiens e 284 di S. Cerevisiae. Abbiamo utilizzato i complessi di H. Sapiens come query sulla collezione di complessi di S. Cerevisiae. L'intera collezione dei complessi di S. Cerevisiae è stata processata in 93 secondi. Ogni complesso di H. Sapiens è stato quindi utilizzato come query sulla collezione di S. Cerevisiae consentendo fino ad un massimo di 4 delezioni. La Figura 8.7a descrive il numero di match e di candidati trovati da SIGMA in base al numero di delezioni ammesse. Il numero di complessi di H. Sapiens utilizzati come query è 785; il numero di complessi di S. Cerevisiae che ne hanno costituito il target è pari a 284. Durante la fase di filtering sono state rimosse tutte le le query con un numero di archi inferiore ad 1. La Figura 8.7b illustra il query time totale. SIGMA è stato capace di individuare match per un totale di 336 complessi proteici di H. Sapiens (1-31 match per query), ottenendo un totale di 2104 matches, dei quali 439 costituivano match esatti e i rimanenti 1635 match inexact.

I match più significativi individuati dal tool sono stati riportati nella Tabella 8.1. Per esempio, il complesso *LSm2-8* di H. Sapiens costituisce un match per il complesso denominato *small nucleolar ribonucleoprotein* di S. Cerevisiae con una delezione. La Figura 8.6 mostra il complesso LSm2-8, il complesso *small nucleolar ribonucleoprotein* ed il loro match.

8.6 Confronto con gli altri tool

Abbiamo applicato i tre metodi precedentemente descritti (SIGMA, Grafil ed Edge) al database AIDS (58) utilizzando query con dimensione variabile tra 16 a 48, con un numero di delezioni possibili comprese tra 1 e 4. Scopo di questo test è stato quello di verificare il *filtering power* dei tre approcci. Il confronto è descritto in Figura 8.8, in cui è riportato il numero medio di candidati rispetto alle 100 query proposte per un prefissato numero di delezioni. Il numero di candidati per ogni query può variare moltissimo: in alcuni casi si registra un solo candidato, in altri addirittura l'intero dataset. Come si può notare dalla figura, SIGMA risulta migliore degli altri due metodi per tutte le

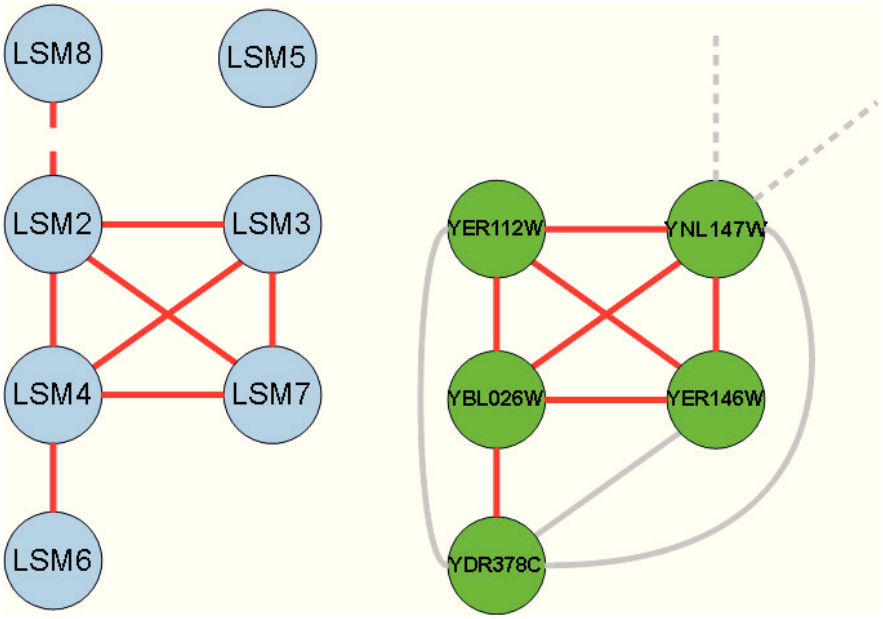
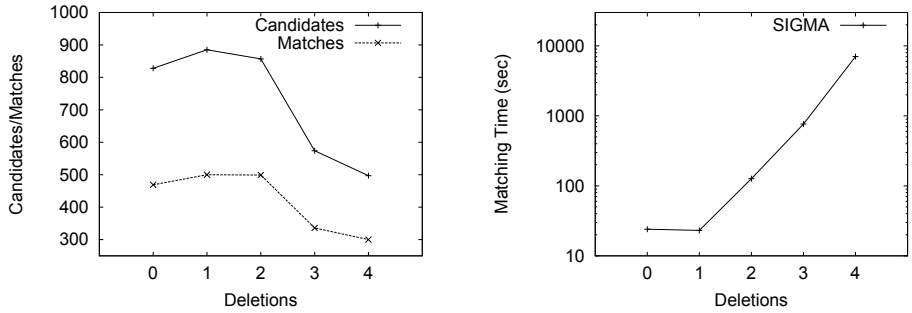


Figura 8.6: Un esempio di match tra un complesso di H. Sapiens ed un complesso di S. Cerevisiae - La parte sinistra rappresenta il complesso di H. Sapiens denominato *LSm2-8*, mentre la parte destra rappresenta la sezione di match con il complesso di S. *small nucleolar ribonucleoprotein* (costituito da 20 nodi e 48 archi). La linea tratteggiata nel complesso a sinistra rappresenta l'arco mancante, mentre la linea rossa sia nella parte destra sia sinistra rappresenta gli archi che fanno match. Infine, le linee grigie nel complesso di destra mostrano gli archi che non hanno un match e le linee grigie tratteggiate nel complesso di destra rappresentano le connessioni alla parte restante del complesso di S. Cerevisiae



(a) (b)

Figura 8.7: Le performance di SIGMA su un dataset di complessi proteici. (a) descrive il numero di candidati prodotto dall'algoritmo ed il numero di match. (b) illustra il query time.

dimensioni di query e questa differenza si evidenzia con l'aumentare della dimensione delle query. Un'analisi più approfondita dei risultati ha mostrato come SIGMA supera Grafil in più del 95% delle query. Per calcolare il pruning power, definito come il rapporto tra il numero di match verificati ed il numero di candidati individuati, è stato applicato un algoritmo di ricerca esaustiva su una parte dei dati. Infatti, è stato considerato un sottoinsieme di 1000 grafi del database target ed è stato fissata a 16 la dimensione delle query. Il risultato, espresso come la media su 10 query è mostrato in Figura 8.9. In questo esempio si può notare come SIGMA sia caratterizzato da un pruning power migliore rispetto agli atri due metodi.

8.7 Conclusioni

E' stata descritta una nuova strategia di indicizzazione di grafi per la ricerca inesatta di grafi integrata all'interno di un tool denominato SIGMA e basato su una innovativa variante del problema di set cover e su un algoritmo greedy che ne approssima la soluzione. I test estensivi che sono stati realizzati su database di composti molecolari hanno mostrato che SIGMA supera i tool esistenti che risolvono lo stesso problema, compreso Grafil (15) che ne costituisce lo stato dell'arte. Esaminando i risultati in dettaglio, crediamo che SIGMA sia più performante di Grafil in quanto quest'ultimo usa solo le informazioni legate al numero di feature della query che mancano nel grafo target. Infatti, in molti casi, questo criterio non risulta sufficientemente selettivo. In contrapposizione, SIGMA tiene in considerazione l'identità delle feature, distinguendo

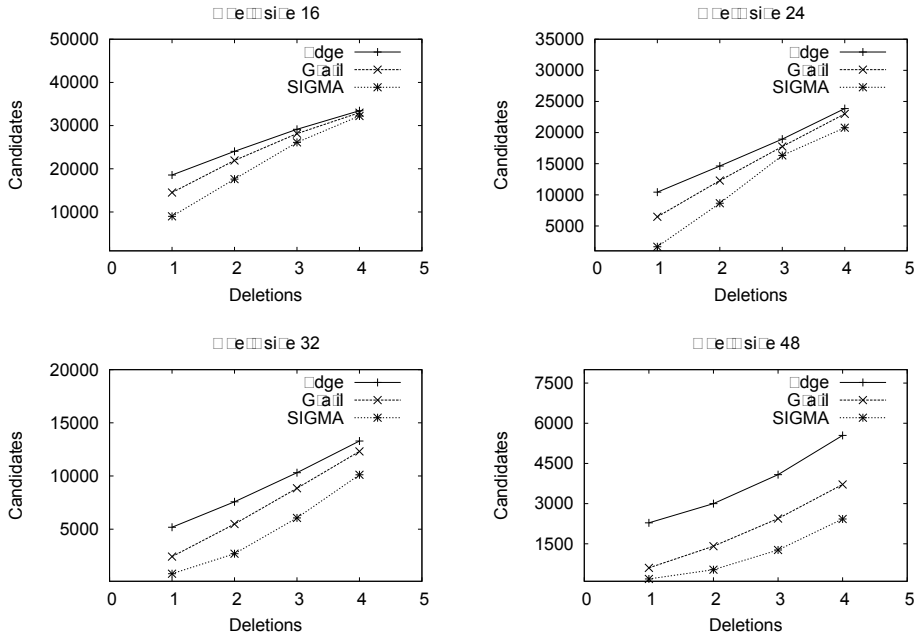


Figura 8.8: Confronto tra il numero di candidati individuati da SIGMA, Grafil ed Edge. Per ciascuna dimensione di query è riportato il numero medio di candidati su 100 query della data dimnesione.

tra feature differenti e ottenendo quindi una capacità di filtering maggiore. Per esempio, consideriamo la query in Figura 8.10. Comparati agli archi periferici, gli archi centrali sono contenuti in un più alto numero di occorrenze di feature, quindi dominano il massimo numero di feature mancanti. Come risultato, il grafo G riportato in figura non può essere scartato da Grafil (15), ma invece viene scartato con successo da SIGMA.

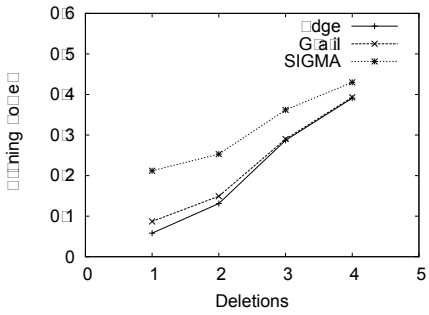


Figura 8.9: Confronto tra il pruning power di SIGMA, Grafil ed Edge.

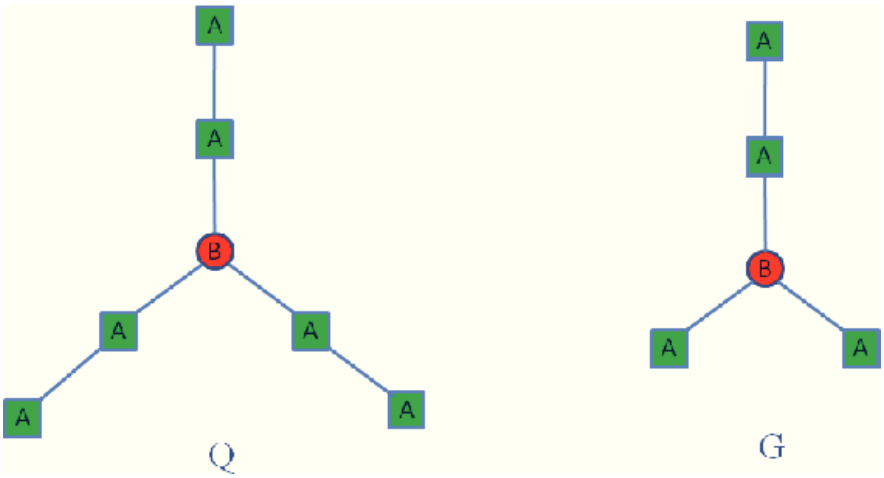


Figura 8.10: Esempio di un grafo scartato da SIGMA ma non da Grafil - Ricerchiamo il grafo query Q con al più 1 delezione, considerando feature costituite da path di lunghezza massima 3. La query contiene 3 occorrenze della feature A-A-B e 3 occorrenze di A-B-A per un totale di 6 occorrenze di feature. Rimuovendo uno degli archi più centrali, perdiamo 3 occorrenze di feature, mentre rimuovendo uno degli archi periferici si perde soltanto una occorrenza di feature.Nel caso in cui sia ammessa una sola delezione, il massimo numero di feature mancanti è 3. G perde 2 occorrenze di feature, quindi non può essere scartato da Grafil. Non esistono archi della query che coprono i due archi mancanti (in G) feature A-A-B , quindi G è scartata da SIGMA.

Complesso Query (H. Sapiens)	**Archi**	**Complessi Match (S. Caeravice)**	**Delezioni**
MCM complex	13	MCM complex	0
		DNA replication preinitiation complex	0
		pre-replicative complex	0
18S U11_U12 snRNP	10	ribonucleoprotein complex	2
		small nuclear ribonucleoprotein complex	2
		spliceosome	2
LSm1-7 complex	9	snRNP U6	0
		ribonucleoprotein complex	0
		small nuclear ribonucleoprotein complex	0
		U4 U6 x U5 tri-snRNP complex	0
		spliceosome	0
		snRNP U5	0
		snRNP U1	0
		small nucleolar ribonucleoprotein complex	2
Lsm2-8 complex	8	snRNP U6	0
		small nuclear ribonucleoprotein complex	0
		ribonucleoprotein complex	0
		snRNP U1	0
		U4 U6 x U5 tri-snRNP complex	0
		spliceosome	0
		snRNP U5	0
		small nucleolar ribonucleoprotein complex	1
SMN1-SIP1-SNRP complex	7	ribonucleoprotein complex	1
12S_U11_snRNP	5	snRNP U5	1
		snRNP U1	1
		ribonucleoprotein complex	1
		small nuclear ribonucleoprotein complex	1
		U4 U6 x U5 tri-snRNP complex	1
		spliceosome	1
		snRNP U5	1
		small nucleolar ribonucleoprotein complex	2
SF3b complex	5	ribonucleoprotein complex	1
		spliceosome	1
		small nuclear ribonucleoprotein complex	2
		snRNP U2	2
p27-cyclinE-Cdk2 Ubiquitin	5	ribonucleoprotein complex	3
E3 ligase(SKP1A-SKP2-		preribosome	3
CUL1-CKS1B-RBX1) complex		90S preribosome	4
		transcription factor complex	4

Tabella 8.1: Una selezione dei match ottenuti utilizzando i complessi di H. Sapiens su un database di complessi di S. Cerevisiae. La colonna Edges si riferisce al numero di archi presenti nella query. L'ultima colonna descrive il numero di delezione che si sono rese necessarie per ottenere il match.

9

Conclusioni e lavori futuri

9.1 Conclusioni

In queste pagine ho cercato di descrivere attraverso i problemi biologici affrontati ed i tool sviluppati, il ricco percorso di ricerca che mi ha condotto dal problema della ricerca esatta di sotto-strutture all'interno delle grandi reti biologiche verso il più dinamico ed adattabile problema della ricerca inesatta. Entrambi i tool sono stati sviluppati tenendo sempre ben presente il problema affrontato, le possibili applicazioni all'ambito biologico e accogliendo i suggerimenti dei ricercatori cui tali tool sono stati presentati direttamente ed indirettamente. Si è dimostrato come la maggior parte dei problemi tipici dei tool che hanno affrontato simili problematiche sono stati generalmente superati. Appare chiaro, tuttavia, che i crescenti sforzi da parte della comunità scientifica nel riuscire a mettere a disposizione nuovi dati, e quindi nuove reti biologiche di dimensioni sempre maggiori, richiedono un impegno maggiore per riuscire a progettare e sviluppare nuovi algoritmi capaci di adattarsi non solo alle nuove dimensioni del problema, ma anche e soprattutto alle reali esigenze della Bioinformatica.

9.2 Lavori futuri

La naturale evoluzione di questa ricerca mi ha condotto verso la progettazione e lo sviluppo di nuovi algoritmi e nuove strutture dati per tentare di risolvere un problema sempre più attuale e cioè il graph querying su grandissime reti biologiche. Mi riferisco a reti formate da milioni di nodi e milioni di archi. Per tale tipologia di problemi gli algoritmi qui proposti non trovano ovviamente applicazione ed è necessario sviluppare

approcci totalmente innovativi per riuscire a risolvere un simile problema con la stessa efficienza dimostrata dai tool SING e SIGMA nel loro contesto di applicazione.

Bibliografia

[1] R. Giugno and D. Shasha. **Graphgrep: A fast and universal method for querying graphs**. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, **2**, pages 112–115, 2002.

[2] X. Yan, P. S. Yu, and J. Han. **Graph Indexing Based on Discriminative Frequent Structure Analysis**. *ACM Transactions on Database Systems*, **30**(4):960–993, 2005. 2, 29, 31, 32, 43, 46, 47, 51, 54

[3] 2, 29, 33, 34, 43, 54

[4] Lei Zou, Lei Chen, Jeffrey Xu Yu, and Yansheng Lu. **A novel spectral coding in a large graph database**. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 181–192, New York, NY, USA, 2008. ACM. 2, 29, 33, 34, 43

[5] L. Cordella, P. Foggia, C. Sansone, and M. Vento. **A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**(10):1367–1372, 2004. 7, 42, 43, 52, 53

[6] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. **Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs**. *Bioinformatics*, **20**(11):1746–1758, 2004. 12, 18, 23, 48, 50, 51

[7] R. Giugno and D. Shasha. **GraphGrep: A Fast and Universal Method for Querying Graphs**. *Proceeding of the International Conference in Pattern recognition (ICPR)*, pages 112–115, 2002. 2, 29

[8] Chen Chen, Xifeng Yan, Philip S. Yu, Jiawei Han, Dong Q. Zhang, and Xiaohui Gu. **Towards graph containment search and indexing**. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 926–937. VLDB Endowment, 2007. 2

[9] Shuo Zhang, Jianzhong Li, Hong Gao, and Zhaonian Zou. **A novel approach for efficient supergraph query processing on graph databases**. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 204–215, New York, NY, USA, 2009. ACM. 2

[10] J. Cheng, Y. Ke, W. Ng, and A. Lu. **Fg-index: towards verification-free query processing on graph databases**. *Proceedings of ACM SIGMOD international conference on Management of data*, pages 857 – 872, 2007. 2, 29, 31, 51

[11] Haoliang Jiang, Haixun Wang, Philip S. Yu, and Shuigeng Zhou. **GString: A Novel Approach for Efficient Search in Graph Databases**. *Data Engineering, International Conference on*, **0**:566–575, 2007. 2

[12] Peixiang Zhao, Jeffrey Xu Yu, and Philip S. Yu. **Graph indexing: tree + delta $\leq$ graph**. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 938–949. VLDB Endowment, 2007. 2, 29, 30, 31, 47, 48, 51

[13] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey X. Yu. **Taming verification hardness: an efficient algorithm for testing subgraph isomorphism**. *Proc. VLDB Endow.*, **1**(1):364–375, 2008. 2

[14] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. **RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs**. *The Computer Journal*, **45**(6):631–644, January 2002. 2

[15] X. Yan, P. S. Yu, and J. Han. **Substructure similarity search in graph databases**. *Proceedings of ACM SIGMOD international conference on Management of data*, pages 766 – 777, 2005. 2, 29, 55, 56, 64, 67, 68

[16] Haichuan Shang, Ke Zhu, Xuemin Lin, Ying Zhang, and Ryutaro Ichise. **Similarity Search on Supergraph Containment**. 2

[17] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. **SAGA: a subgraph matching tool for biological graphs**. *Bioinformatics*, **23**(2):232–239, 2007. 2

[18] William H. Walters. **Google Scholar coverage of a multidisciplinary field**. *Inf. Process. Manage.*, **43**:1121–1132, July 2007.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 6

[20] The Gene Ontology Consortium. **The Gene Ontology in 2010: extensions and refinements**. *Nucleic Acids Research*, **38**(suppl 1):D331–D335, January 2010. 9

[21] R. Giugno M. Mongiovì A. Pulvirenti R. Di Natale, A. Ferro and D. Shasha. **SING: Subgraph search In Non-homogeneous Graphs**. *BMC Bioinformatics 2010, 11:96*, 2010. 10, 12, 23, 26, 29, 35

[22] M. Kanehisa. **The KEGG database.** *Novartis Found Symp*, **247**, 2002. 10, 11

[23] K. Wang, F. Hu, K. Xu, H. Cheng, M. Jiang, R. Feng, J. Li, and T. Wen. **CASCADE SCAN mining signal transduction network from high-throughput data based on steepest descent method.** *BMC Bioinformatics*, **12**(1):164, 2011. 10

[24] W.H. Piel, M.J. Sanderson, and M.J. Donoghue. **The small-world dynamics of tree networks and data mining in phyloinformatics.** *Bioinformatics*, **19**(9):1162–8, 2003. 10

[25] Susie Stephens, Johan Rung, and Xavier Lopez. **Graph Data Representation in Oracle Database 10: Case Studies in Life Sciences.** *IEEE Data Eng. Bull.* 11

BIBLIOGRAFIA

[26] Roded Sharan, Silpa Suthram, Ryan M. Kelley, Tanja Kuhn, Scott McCuine, Peter Uetz, Taylor Sittler, Richard M. Karp, and Trey Ideker. **Conserved patterns of protein interaction in multiple species.** *Proceedings of the National Academy of Sciences of the United States of America*, **102**(6):1974–1979, February 2005. 12

[27] M. Mongiovì, R. Di Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. **A Set-cover-based Approach for Inexact Graph Matching**. Stanford, CA, USA, 2009. Life Science Society. 12, 26

[28] Hiroaki Kitano, Akira Funahashi, Yukiko Matsuoka, and Kanae Oda. **Using process diagrams for the graphical representation of biological networks.** *Nature biotechnology*, **23**(8):961–966, August 2005. 12

[29] Shai S. Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. **Network motifs in the transcriptional regulation network of Escherichia coli**. *Nature Genetics*, **31**(1):64–68, April 2002. 12, 22

[30] Jen-hwa H. Chu, Scott T. Weiss, Vincent J. Carey, and Benjamin A. Raby. **A graphical model approach for inferring large-scale networks integrating gene expression and genetic polymorphism.** *BMC systems biology*, **3**(1):55+, 2009. 12

[31] Christian von Mering, Roland Krause, Berend Snel, Michael Cornell, Stephen G. Oliver, Stanley Fields, and Peer Bork. **Comparative assessment of large-scale data sets of protein–protein interactions**. *Nature*, **417**(6887):399–403, May 2002. 13, 14

[32] E. M. Phizicky and S. Fields. **Protein-protein interactions: methods for detection and analysis**. *Microbiol. Rev.*, **59**(1):94–123, March 1995. 14

[33] A. Vitale. **Physical methods.** *Plant Molecular Biology*. 14

[34] et al. Gavin, A.C. **Functional organization of the yeast genome by systematic analysis of protein complexes**. *Nature*. 14, 16

[35] G. Rigaut, A. Shevchenko, B. Rutz, M. Wilm, M. Mann, and B. Séraphin. **A generic protein purification method for protein complex characterization and proteome exploration.** *Nat Biotechnol*, **17**(10):1030–1032, October 1999. 14

[36] Y. Ho, A. Gruhler, A. Heilbut, G.D. Bader, L. Moore, S. Adams, A. Millar, P. Taylor, K. Bennett, K. Boutilier, L. Yang, C. Wolting, I. Donaldson, S. Schandorff, J. Shewnarane, M. Vo, J. Taggart, M. Goudreault, B. Muskat, C. Alfarano, D. Dewar, Z. Lin, K. Michalickova, A.R. Willems, H. Sassi, P.A. Nielsen, K.J. Rasmussen, J.R. Andersen, L.E. Johansen, L.H. Hansen, H. Jespersen, A. Podtelejnikov, E. Nielsen, J. Crawford, V. Poulsen, B.D. Sørensen, J. Matthiesen, R.C. Hendrickson, F. Gleeson, T. Pawson, M.F. Moran, D. Durocher, M. Mann, C.W.V. Hogue, D. Figeys, and M. Tyers. **Systematic identification of protein complexes in Saccharomyces cerevisiae by mass spectrometry.** *Nature*, **415**(6868):180–3, 2002. 14

[37] M. Mann, R. C. Hendrickson, and A. Pandey. **Analysis of proteins and proteomes by mass spectrometry.** *Annual review of biochemistry*, **70**:437–473, 2001. 14

[38] Lukasz Salwinski, Christopher S. Miller, Adam J. Smith, Frank K. Pettit, James U. Bowie, and David Eisenberg. **The Database of Interacting Proteins: 2004 update.** *Nucleic acids research*, **32**(Database issue):D449–D451, January 2004. 20

[39] H. Schoof, M. Spannagl, L. Yang, R. Ernst, H. Gundlach, D. Haase, G. Haberer, and K.F.X. Mayer. **Munich information center for protein sequences plant genome resources: a framework for integrative and comparative analyses 1(W).** *Plant Physiol*, **138**(3):1301–9, 2005. 20

[40] T. S. Keshava Prasad, Renu Goel, Kumaran Kandasamy, Shivakumar Keerthikumar, Sameer Kumar, Suresh Mathivanan, Deepthi Telikicherla, Rajesh Raju, Beema Shafreen, Abhilash Venugopal, Lavanya Balakrishnan, Arivusudar Marimuthu, Sutopa Banerjee, Devi S. Somanathan, Aimy Sebastian, Sandhya Rani, Somak Ray, C. J. Harrys Kishore, Sashi Kanth, Mukhtar Ahmed, Manoj K. Kashyap, Riaz Mohmood, Y. L. Ramachandra, V. Krishna, B. Abdul Rahiman, Sujatha Mohan, Prathibha Ranganathan, Subhashri Ramabadran, Raghothama Chaerkady, and Akhilesh Pandey. **Human Protein Reference Database—2009 update**. *Nucleic Acids Research*, **37**(suppl 1):D767–D772, January 2009. 20

[41] Jingkai Yu, Svetlana Pacifico, Guozhen Liu, and Russell Finley. **DroID: the Drosophila Interactions Database, a comprehensive resource for annotated gene and protein interactions**. *BMC Genomics*, **9**(1):461+, October 2008. 20

[42] Henning Hermjakob, Luisa Montecchi Palazzi, Gary Bader, Jerome Wojcik, Lukasz Salwinski, Arnaud Ceol, Susan Moore, Sandra Orchard, Ugis Sarkans, Christian von Mering, Bernd Roechert, Sylvain Poux, Eva Jung, Henning Mersch, Paul Kersey, Michael Lappe, Yixue Li, Rong Zeng, Debashis Rana, Macha Nikolski, Holger Husi, Christine Brun, K. Shanker, Seth G. Grant, Chris Sander, Peer Bork, Weimin Zhu, Akhilesh Pandey, Alvis Brazma, Bernard Jacq, Marc Vidal, David Sherman, Pierre Legrain, Gianni Cesareni, Ioannis Xenarios, David Eisenberg, Boris Steipe, Chris Hogue, and Rolf Apweiler. **The HUPO PSI's molecular interaction format–a community standard for the representation of protein interaction data.** *Nature biotechnology*, **22**(2):177–183, February 2004. 15

[43] Peer Bork, Lars J. Jensen, Christian von Mering, Arun K. Ramani, Insuk Lee, and Edward M. Marcotte. **Protein interaction networks from yeast to human.** *Current opinion in structural biology*, **14**(3):292–299, June 2004. 16

[44] R. Overbeek, N. Larsen, G. D. Pusch, M. D'Souza, E. Selkov, N. Kyrpides, M. Fonstein, N. Maltsev, and E. Selkov. **WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction.** *Nucleic acids research*, **28**(1):123–125, January 2000. 17, 18

[45] H. Jeong, S. P. Mason, A. L. Barabasi, and Z. N. Oltvai. **Lethality and centrality in protein networks**. *Nature*, **411**(6833):41–42, May 2001. 18

[46] Emmanuel D. Levy, Jose B. Pereira-Leal, Cyrus Chothia, and Sarah A. Teichmann. **3D complex: a structural classification of protein complexes.** *PLoS computational biology*, **2**(11), November 2006. 19

[47] H. W. Mewes, D. Frishman, U. Güldener, G. Mannhaupt, K. Mayer, M. Mokrejs, B. Morgenstern, M. Münsterkötter, S. Rudd, and B. Weil. **MIPS: a database for genomes and protein sequences.** *Nucleic acids research*, **30**(1):31–34, January 2002. 19

[48] N. Przulj, D. A. Wigle, and I. Jurisica. **Functional topology in a network of protein interactions.** *Bioinformatics*, **20**(3):340–348, February 2004. 19

[49] A.P. Potapov, N. Voss, N. Sasse, and E. Wingender. **Topology of Mammalian transcription networks.** *Genome Inform*, **16**(2):270–8, 2005. 21

[50] R. Milo, Et Al, and Cell Biology. **Network motifs: simple building blocks of complex networks**. *Science*, pages 824–827, 2002. 22

[51] S. Mangan and U. Alon. **Structure and function of the feed-forward loop network motif**. *Proceedings of the National Academy of Sciences*, **100**(21):11980–11985, October 2003. 22

[52] Euripides G. M. Petrakis and Christos Faloutsos. **Similarity Searching in Medical Image Databases**. *IEEE Transactions on Knowledge and Data Engineering*, **9**(3):435–447, May 1997. 25

[53] T. Madej, J. F. Gibrat, and S. H. Bryant. **Threading a database of protein cores.** *Proteins*, **23**(3):356–369, November 1995. 25

[54] Roy Goldman. **DataGuides: Enabling query formulation and optimization in semistructured databases**, 1997. 25

[55] Stephen A. Cook. **The complexity of theorem-proving procedures**. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. 25, 26

[56] D. W. Williams, Jun Huan, and Wei Wang. **Graph Database Indexing Using Structured Graph Decomposition**. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 976–985, 2007. 29, 31

[57] S. Zhang, M. Hu, and J. Yang. **TreePi: A Novel Graph Indexing Method**. *Proceedings of IEEE 23rd International Conference on Data Engineering*, pages 181–192, 2007. 29, 31, 37

[58] **NCI DTP Antiviral Screen data.** `http://dtp.nci.nih.gov/docs/aids/aids_data.html`. 42, 65

[59] J.R. Ullmann. **An algorithm for Subgraph Isomorphism**. *Journal of the Association for Computing Machinery*, **23**:31–42, 1976. 43

[60] **National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov/**. 50

[61] **Saccharomyces genome database. http://www.yeastgenome.org/, Sept. 2008**. 51, 64

[62] C. Stark, B. J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. **BioGRID: a general repository for interaction datasets.** *Nucleic Acids Res*, **34**(Database issue), January 2006. 51, 52, 65

[63] D. Bijl. **The serotonin syndrome**. *The Netherlands journal of medicine*, **62**(9):309–313, 2004. 56

[64] Sharon Bruckner, Falk Huffner, Richard M. Karp, Ron Shamir, and Roded Sharan. **Torque: topology-free querying of protein interaction networks**. *Nucl. Acids Res.*, **37**(suppl_2):W106–108, July 2009. 57

[65] Banu Dost, Tomer Shlomi, Nitin Gupta, Eytan Ruppin, Vineet Bafna, and Roded Sharan. **QNet: A Tool for Querying Protein Interaction Networks**. *Journal of Computational Biology*, **15**(7):913–925, 2008. 57

[66] R. M. Karp. **Reducibility among combinatorial problems**. *Complexity of Computer Computations*, pages 85–103, 1972. 59

[67] D. S. Johnson. **Approximation algorithms for combinatorial problems**. *J. Comput. System Sci.*, pages 256–278, 1974. 59

[68] S. Rajagopalan and V. V. Vazirani. **Primal-dual RNC approximation algorithms for (multi)-set (multi)-cover and covering integer programs**. In *SFCS '93: Proceedings of the Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 322–331, Washington, DC, USA, 1993. IEEE Computer Society. 59

[69] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. **An Improved Algorithm for Matching Large Graphs**. *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001. 64

[70] A. Ruepp, B. Brauner, I. Dunger-Kaltenbach, G. Frishman, C. Montrone, M. Stransky, B. Waegele, T. Schmidt, O. N. Doudieu, V. Stumpflen, and H. W. Mewes. **Corum: the comprehensive resource of mammalian protein complexes**. *Nucleic Acids Research*, **36**, 2008. 64