POLITECNICO DI MILANO
Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Informatica

UNO STUDIO SISTEMATICO DELLE INCONSISTENZE NEI NOMI DEI MALWARE

Relatore:
Prof. Stefano ZANERO
Correlatori:
Ing. Federico MAGGI
Ing. Guido SALVANESCHI

Tesi di Laurea di:
Andrea BELLINI
Matr. n. 725576

Anno Accademico 2009–2010

Dedicato a mamma e a papà che avrebbe voluto tanto vedere questo giorno e che sicuramente lo sta vedendo.

Andrea Bellini

Ringraziamenti

Ringrazio il Prof. Stefano Zanero e l'Ing. Guido Salvaneschi per l'interesse riposto in questo lavoro, la disponibilità ed i consigli. Ringrazio Marco Ribolini per l'inestimabile aiuto con le immagini. Infine ringrazio, in modo particolare, l'Ing. Federico Maggi che ha seguito con passione la totale evoluzione di questa tesi e che si è reso disponibile per consigli e revisioni anche a discapito del suo tempo libero (e.g., revisione in Skype il 23 Dicembre alle 6 PM).

Indice

Indice **iv**

Elenco delle figure **vi**

Elenco delle tabelle **vii**

1 Introduzione **1**

2 Analisi del problema **5**
2.1 Tipi di malware 6
2.2 Tecniche di rilevazione e offuscamento 7
2.3 Origine del problema del naming 14
2.4 Soluzioni proposte al problema del naming 15

3 Una tecnica di rilevazione delle inconsistenze **19**
3.1 Fase 1: generazione del modello del sistema di naming 21
3.1.1 Estrazione dei pattern 21
3.1.2 Clustering 23
3.1.3 Albero dei cluster 24
3.1.4 Dettagli implementativi 25
3.2 Fase 2: confronto dei modelli ed analisi 28
3.2.1 Distanza tra due sistemi di naming 29
3.2.2 Scatter measure 31
3.2.3 Estrazione inconsistenze 33
3.2.4 Dettagli implementativi 36

4 Risultati sperimentali **45**
4.1 Generazione del dataset 45
4.2 Visualizzazione strutture di naming 46
4.3 Distanza tra modelli 51
4.4 Scatter measure 53
4.5 Analisi inconsistenze 53

5 Conclusioni e sviluppi futuri **67**

A Distanza $d_H(c, c')$ **69**

Bibliografia **71**

Elenco delle figure

1.1 Andamento del numero di file sottomessi al portale VirusTotal nei giorni che vanno dal 4 al 10 Febbraio 2011 [VirusTotal, 2011]. . . 3

2.1 Dieci packer più frequenti presenti nelle scansioni di VirusTotal . 15

3.1 Processo di analisi 20
3.2 Esempio di output della **Fase 1** 24
3.3 Esempio di estrazione pattern 27
3.4 Esempio di $\Gamma(c, \mathcal{C}')$ 31
3.5 Scelta tra i possibili insiemi $\Gamma(c, \mathcal{C}')$ 32
3.6 Esempio di consistenza, inconsistenza debole e inconsistenza forte 34
3.7 Esempio di relazione poco significativa 35
3.8 Confronto della sensibilità ai disturbi tra normalizzazione per cluster e per elementi. 37
3.9 Confronto della sensibilità ai disturbi tra normalizzazione a singolo fattore e a due fattori. 39
3.10 Confronto tra $d_{H_1}(c, c')$, $d_{H_2}(c, c')$, $d_J(c, c')$. 41
3.11 Confronto tra $d_{H_1}(c, c')$, $d_{H_2}(c, c')$, $d_{H_3}(c, c')$ e $d_J(c, c')$ 43

4.1 Visualizzazione dei modelli dei sistemi di naming 48
4.2 Zoom sui modelli dei sistemi di naming 49
4.3 Esempio di come i differenti vendor clusterizzano gli stessi malware 50
4.4 Distanza tra sistemi di naming $D(\mathcal{C}, \mathcal{C}')$. 52
4.5 Scatter measure valutata per differenti valori di T_σ 60
4.6 Esempio reale di inconsistenza debole e forte 61
4.7 Esempio reale di inconsistenza forte di grandi dimensioni 62
4.8 Inconsistenze a $\mathcal{C}_{type_n}$ 63
4.9 Inconsistenze a $\mathcal{C}_{family}$ 64
4.10 Inconsistenze a $\mathcal{C}_{version_n}$ 65

Elenco delle tabelle

3.1 Esempi di distanze 40
3.2 Percentuale di confronti per cui $|d_{H_2} - d_J| \leq T_h$ e $|d_{H_1} - d_J| \leq T_h$ . . 42

4.1 Distanza tra i modelli 54
4.2 Risultati scatter measure per il taglio $\mathcal{C}_{type_n}$ al variare di T_σ 54
4.3 Risultati scatter measure per il taglio $\mathcal{C}_{family}$ al variare di T_σ 55
4.4 Risultati scatter measure per il taglio $\mathcal{C}_{version_n}$ al variare di T_σ . . . 56
4.5 Risultati relativi all'estrazione delle inconsistenze a livello $\mathcal{C}_{type}$. . 57
4.6 Risultati relativi all'estrazione delle inconsistenze a livello $\mathcal{C}_{family}$. 58
4.7 Risultati relativi all'estrazione delle inconsistenze a livello $\mathcal{C}_{version_n}$ 59

Capitolo 1

Introduzione

Un codice malevolo, o più comunemente un *malware*, è un software installato senza il consenso dell'utente, il cui principale scopo è quello di sottrarre informazioni, o più in generale quello di causare danni al sistema sul quale è eseguito.

Quando un antivirus rileva un file come malevolo (e.g., virus o più in generale malware) vi associa una stringa, in gergo denominata *label* (e.g., "*Win32.Net-Worm.Allaple.b.4*"). Tale stringa contiene informazioni che permettono all'utente di identificare il malware e di ricercare ulteriori informazioni utili alla sua rimozione o alla conoscenza dei potenziali danni subiti.

Le informazioni contenute nei label e la loro struttura sono fortemente dipendenti dalle convenzioni di assegnazione dei label (o "sistemi di naming") adottate dall'antivirus che le ha generate. Per uno stesso malware, antivirus differenti, possono generare label differenti, con strutture ed informazioni diverse. Ad esempio "*Win32/FakePAV.E*" rispetto a "*Win32.Net-Worm.Allaple.b.4*" non fornisce informazioni sul tipo (i.e., Worm) e contiene altri caratteri separatori (i.e., "/" al posto di "-"), mentre "*RogueAntiSpyware.SecurityEssentialFraud!rem*" non include informazioni sulla piattaforma necessaria all'esecuzione del malware.

Storicamente, per l'assegnazione dei nomi ai malware non è mai stata seguita o stabilita un'unica convenzione. Ricercatori e produttori di antivirus erano soliti assegnare i nomi in base a caratteristiche ritenute interessanti. Tra i ricercatori, prese piede addirittura una specie di gioco denominato "*name that virues*", che consisteva nel riconoscere un virus in base alle sue caratteristiche [Pierre-Marc Bureau, 2008].

D'altra parte, per l'utente medio, conoscere il nome che il proprio antivirus assegna al campione in analisi risulta di poca rilevanza (nel caso fossero necessarie informazioni più dettagliate spesso queste sono disponibili in una base di conoscenza gestita dai produttori di antivirus). Per alcuni tipi di ricerche, una nomenclatura pragmatica come quella proposta nella Wild

List[Andreas Marx, 2007] può risultare sufficiente. D'altro canto però, l'assegnazione dei nomi risulta un vero problema quando si cerca di correlare, o effettuare data mining su dati forniti da antivirus differenti. Ad esempio, confrontare le "top-ten malware list" pubblicate periodicamente dai più svariati produttori di antivirus, risulta uno sforzo inutile a causa delle differenti capacità di rilevazione e delle differenze nell'assegnazione dei nomi [infosecurity.com, 2010]. Un simile problema è presente negli scenari enterprise, dove differenti tipi di antivirus possono essere eseguiti parallelamente. Anche in questo caso correlare i vari report diventa un problema di non facile soluzione.

In uno panorama di questo tipo rispondere alla semplice domanda *"L'antivirus X è in grado di rilevare il virus Y?"* risulta estremamente complesso ed apre una serie di congetture su "cos'è Y?", "chi l'ha definito?", etc. Più propriamente oggi si è in grado di rispondere alla precedente domanda così riformulata: "L'antivirus X è in grado di rilevare minacce analizzando il campione Y?. La sostanziale differenza tra le due formulazioni risiede nel fatto, che nella seconda Y è un oggetto fisico che può essere analizzato e non un nome assegnato da qualcuno per identificare tale oggetto.

Il problema dell'assegnazione dei nomi, che fino a qualche anno fa era considerato semplicemente una seccatura, è diventato qualcosa di potenzialmente pericoloso a causa del cambiamento del target dei malware. I codici malevoli scritti da autori in cerca di notorietà hanno lasciato il posto a malware il cui scopo è quello di arricchirne l'autore. In questo scenario, il download di trojan, in modalità "drive by", da pagine web "infette" è diventata la principale minaccia. Inoltre motori polimorfici permettono con facilità la creazione di diversi strati di pacchettizzazione dei virus, offuscamenti, ricompilazioni e tecniche di self-updating per le bot-net. Questo cambiamento ha portato ad un aumento continuo del numero di famiglie di virus generate ogni giorno, che spesso risultano mutazioni di malware già conosciuti. In Figura 1.1 è riportato l'andamento del numero di file inviati a VirusTotal tra il 4 e il 10 Febbraio 2011. Si può notare con facilità che il numero di nuovi malware scovati ogni giorno è dell'ordine di alcune migliaia. A questi devono essere aggiunti tutti non ancora riconosciuti; quindi si può affermare con sufficiente certezza che il numero di malware prodotti giornalmente sia addirittura molto maggiore. Conseguentemente all'evoluzione dei virus, sono evolute anche le tecniche di rilevazione che, non potendosi più basare sul riconoscimento esatto del virus a causa dell'aumento del loro numero, si basano oggi su signature generiche ed euristiche [Kelchner, 2010]. Conseguentemente l'inconsistenza tra i sistemi di assegnazione dei nomi diventa un problema di ancor più difficile soluzione.

Questo lavoro si propone di andare oltre le congetture presenti in letteratura sul problema del naming, per focalizzarsi su un approccio sistematico in grado di identificare e quantificare le discrepanze presenti nella classificazione dei malware da parte di differenti antivirus. Intuitivamente ci si aspetta

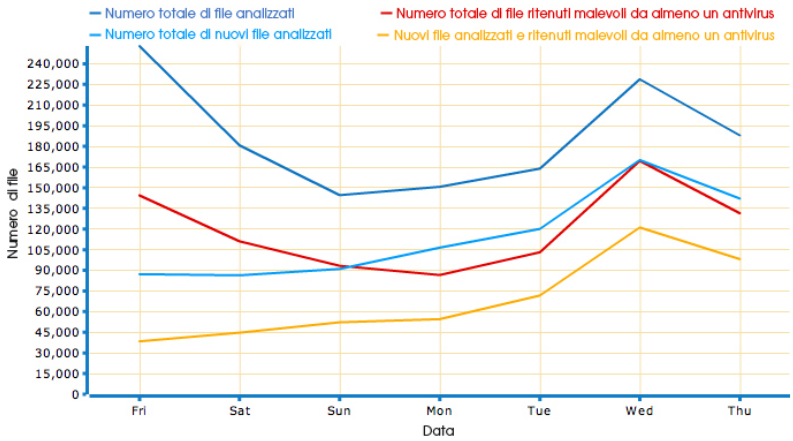


Figura 1.1: Andamento del numero di file sottomessi al portale VirusTotal nei giorni che vanno dal 4 al 10 Febbraio 2011 [VirusTotal, 2011].

che due antivirus A e B, assegnino nomi differenti alle stesse famiglie di malware, ma che, in qualche modo, sia possibile trovare una corrispondenza tra la classificazione fornita da A e quella fornita da B su uno stesso insieme di campioni. Lo scopo di questa lavoroè la realizzazione di uno strumento di analisi dei nomi, utile a costruire automaticamente evidenza sperimentale per confermare o smentire suddette congetture.

Il presente documento, inquadra il problema e descrive il software implementato. In particolare:

Capitolo 2: in Sezione 2.1 è fornita una breve panoramica sui tipi di malware esistenti, in Sezione 2.2 vengono esposti i differenti approcci alla scansione e i principali metodi di offuscamento, in Sezione 2.3 è introdotto come il cambiamento nello sviluppo dei malware, e la conseguente evoluzione degli antivirus, hanno portato ad avere sistemi di naming sempre più inconsistenti. Infine in Sezione 2.4 sono presentate le tecniche e le soluzioni proposte dalla comunità per la risoluzione del problema, motivando così la necessità di un approccio sistematico in grado di rilevare e quantificare il livello di inconsistenza tra i differenti sistemi di naming.

Capitolo 3 contiene il contributo fornito con questo lavoro, più precisamente: in Sezione 3.1 è definito un approccio sistematico in grado di modellare i sistemi di naming tramite l'utilizzo di grafi. Modelli sui quali saranno poi definiti i concetti di *inconsistenza debole* e *forte* ed in Sezione 3.2 sono proposte due misure quantitative in grado di definire il livello globale di inconsistenza presente tra i sistemi di naming differenti. É inoltre descritto un semplice algoritmo in grado di estrarre

porzioni di modelli, dei sistemi di naming, che sono considerate tra loro *consistenti*, *debolmente inconsistenti* o *fortemente inconsistenti*.

Capitolo 4: sono esposti i risultati ricavati dall'applicazione delle tecniche presentate nel Capitolo 3 su un dataset reale di circa 100.000 differenti campioni, etichettati da quattro antivirus. Più precisamente sono presentate: le strutture dei modelli, le inconsistenze (deboli e forti), le consistenze ed i due indici riassuntivi del livello globale di inconsistenza.

Capitolo 5: sono presentate le conclusioni generali tratte dai dati ricavati nel Capitolo 4. Sono poi discusse le limitazioni dell'approccio congiuntamente alle eventuali modifiche necessarie per superarle.

I risultati sperimentali ivi descritti, ricavati dall'implementazione e dall'applicazione delle tecniche presentate nel Capitolo 3, mettono in luce alcune proprietà del problema del naming interessanti e non intuitive. In primo luogo dimostrano sperimentalmente, grazie all'estrazione di porzioni di modelli tra loro inconsistenti, che il problema è di tipo semantico-strutturale, e non è generato da banali differenze sintattiche facilmente risolvibili. In secondo luogo, l'attenzione si pone sul livello di inconsistenza tra due sistemi di assegnazione nomi, facendo emergere come vi siano coppie di antivirus più consistenti di altre.

Questi risultati sono stati sottoposti a peer review del comitato tecnico di un'importante conferenza internazionale del settore.

Capitolo 2

Analisi del problema

Il problema del naming consiste nell'aver associato allo stesso malware label tra loro inconsistenti. Quest'inconsistenza, che all'inizio degli anni 90 era sostanzialmente contenuta, può generare difficoltà quando si tenta di correlare i risultati forniti da vari antivirus, oppure quando si cerca di effettuare operazioni di data mining su di essi. La comunità ha proposto svariate soluzione al problema, che spaziano da software in grado di tenere traccia delle corrispondenze tra label generati da antivirus differenti (i.e., Vgrep [VirusBullettin, 2011]), a standard riguardanti la generazione dei label (i.e., CARO), a elenchi o liste di campioni rappresentati singole famiglie o minacce (i.e., Wild List, CME). Nessuna delle soluzioni proposte è stata però in grado di risolvere in maniera soddisfacente il problema, che è diventato sempre di più grave, soprattutto causa della sostituzione delle classiche tecniche di analisi statica, che effettuano l'identificazione esatta di un preciso malware analizzandone il codice, con le più generali tecniche dinamiche, che identificano i malware basandosi sul monitoraggio del loro comportamento. L'evoluzione è stata necessaria per poter fronteggiare il continuo incremento del numero di nuovi malware generati ogni giorno, incremento causato perlopiù dall'utilizzo intensivo di tecniche di offuscamento, che permettono di camuffare i malware in modo che non sia possibile riconoscere all'interno del loro codice una sequenza di istruzioni che li identifichi. Tipicamente, queste tecniche possono essere suddivise in due grosse categorie: *polimorfiche*, che cifrano o comprimono parte del codice del malware per decifrarlo o decomprimerlo a run time esattamente prima che venga eseguito, e *metamorfiche* che modificano il codice del malware ogni volta che esso si replica.

Nel seguito, prima di fornire una panoramica sulle tecniche di scansione e metodi di offuscamento (Sezione 2.2), sulle origini del problema del naming (Sezione 2.3) e sulle soluzioni proposte dalla comunità (Sezione 2.4), si descrivono le tipologie più comuni di malware, per permettere al lettore una maggiore comprensione degli esempi e dei dati presentati nel corso del documento.

2.1 Tipi di malware

Il codice malevolo è solitamente classificato in base ai suoi obbiettivi e al metodo di propagazione, tale classificazione compare spesso all'interno dei label generati dagli antivirus, con lo scopo di permettere l'identificazione del tipo di malware che ci si ritrova a fronteggiare. Di seguito è fornito un elenco delle classificazioni più comuni estratto da [Microsoft, 2011].

Virus malware che comunemente replica se stesso infettando altri file. L'apertura di tali file consente l'esecuzione del codice malevolo e la sua propagazione.

Trojan applicazione malevola che non è in grado di diffondersi di propria iniziativa, solitamente appare come un applicazione legittima ed utile ma, in realtà esegue codice malevolo attaccando il sistema o sottraendo dati. Sono presenti numerose tipologie di Trojan come *Dropper/Downloader* che hanno lo scopo di installare altro codice malevolo, oppure come *Proxy* che permettono di configurare il computer infettato come un internet-proxy.

Backdoor codice malevolo che permette l'accesso al sistema da parte di entità esterne. Solitamente modificano le politiche di sicurezza locale in modo da permette il completo accesso remoto tramite la rete.

Spyware programma che colleziona informazioni sull'utente, ad esempio i siti web che visita, senza un suo consenso. L'installazione di tali programmi avviene tramite l'installazione di software utile, solitamente senza che l'utente ne venga informato o ne fornisca il consenso.

Spammer programma che invia una grande quantità di e-mail non richieste.

Password Stealer o PWS è un malware il cui scopo è quello di trasmettere informazioni personali, come username e password. Solitamente sono eseguiti assieme a dei keylogger che inviano screenshot e digitazioni.

Dialer programma che esegue chiamate telefoniche non autorizzate. Spesso tali chiamate riguardano numeri telefonici molto costosi.

Rootkit programma il cui scopo principale è l'esecuzione di alcune funzioni che non possono essere rilevate o rimosse con facilità dall'amministratore di sistema. Generalmente tali funzioni sono nascondere se stesso oppure altri malware.

Worm programma che si propaga automaticamente da un computer ad un altro utilizzando svariati vettori di infezione tra cui mail, dispositivi USB removibili, file shearing etc.

Adware programma che visualizza pubblicità. Mentre alcuni di questi programmi possono essere utili in quanto permettono di sponsorizzare un servizio, altri forniscono uno visualizzazione continua di messaggi pubblicitari senza il consenso dell'utente.

Browser Modifier programma che modifica i settaggi del browser, come l'homepage, senza il consenso dell'utente.

Exploit codice malevolo che sfrutta le vulnerabilità del sistema che attacca.

Rogue Security Software software che appare utile per migliorare la sicurezza del sistema, generando una serie di allarmi erronei o ingannevoli, e che tramite social engineering fa partecipare l'utente a transazioni fraudolente.

Le categorie elencate, oltre a presentare svariate sotto-categorie, sono tra loro combinabili, ad esempio un Worm può contenere un payload che installa un Backdoor, permettendo così la velocissima creazione di una rete di computer accessibile in maniera remota, da cui è possibile lanciare attacchi di distribuited denial-of-service (DDoS) o altro tipo.

2.2 Tecniche di rilevazione e offuscamento

A causa della continua interconnessione dei computer la presenza di codice malevolo è diventata una minaccia pericolosa e onnipresente. I malware possono infettare un host utilizzando vari metodi: nascondendosi all'interno di funzionalità appartenenti a programmi regolari, sfruttando difetti e vulnerabilità presenti in alcuni sistemi o software oppure tramite il social engineering. Dato il potenziale effetto devastante che i malware possono avere all'interno di una struttura informatica, la loro rilevazione e rimozione diventa un'operazione di cruciale importanza [Christodorescu and Jha, 2003].

La modalità di analisi, utilizzata dagli antivirus per rilevare i malware, può essere fondamentalmente di due tipi: *analisi statica* che si basana sull'analisi del codice e sul riconoscimento di alcune sue parti come malevole, *analisi dinamicha* che si basa sull'osservazione del comportamento del malware in esecuzione.

Analisi Statica

L'analisi statica è la tecnica più datata e classica, l'idea che sta alla base dell'analisi statica è quella di ricercare, all'interno di un programma, la presenza di una specifica sequenza di istruzioni attribuite ad un virus (i.e., virus signature). Se all'interno di un programma è presente una sequenza di istruzioni riconducibile ad un virus signatur allora con alta probabilità tale programma è infetto o malevolo. Il virus "*Chernobyl/CIH*" è, ad esempio, rilevato tramite

la presenza del seguente codice esadecimale:

```
E800 0000 005B 8D4B 4251 5050
0F01 4C24 FE5B 83C3 1CFA 8B2B
```

che costituisce la parte del corpo del virus, corrispondente alle seguente sequenza di istruzioni IA-32.

```
E8 00000000   call 0h
5B            pop ebx
8D 4B 42      lea ecx, [ebx + 42h]
51            push ecx
50            push eax
50            push eax
0F01 4C 24 FE sidt [esp - 02h]
5B            pop ebx
83            C3 1C add ebx, 1Ch
FA            cli
8B 2B         mov ebp, [ebx]
```

Il principale problema di questo approccio è innanzitutto la sua incapacità di rilevare gli zero-day malware, in quanto ogni nuovo malware deve essere prima analizzato e da esso devono essere estratti i virus signature che lo caratterizzano e lo identificano come tale. Inoltre l'approccio è efficace finché il codice originale dei malware non subisce modifiche, oppure subisce solamente piccoli aggiornamenti, dato che, per poterne permettere l'identificazione, la porzione di codice che lo caratterizza (i.e., virus signature) deve continuare ad essere condivisa tra le varianti dello stesso virus. Ad esempio le versioni 1.2, 1.3, 1.4 di "*Chernobyl/CIH*" differiscono tra loro e dalla versione vanilla, principalmente nella parte relativa alla specifica della data di attivazione del virus, e di conseguenza lo stesso virus signature può essere utilizzato per rilevare tutte le versioni.

Gli scrittori di malware e gli sviluppatori di software si ritrovano quindi a giocare ad un gioco di offuscamento-deoffuscamento, dove gli scrittori di malware cercano di offuscare le versioni vanilla dei malware, in modo che i virus signature, utilizzati dagli antivirus non siano in grado di rilevare queste versioni modificate. Mentre gli sviluppatori di antivirus, ricercano tecniche in grado di deoffuscare i malware, permettendo così il loro riconoscimento tramite i virus signature [Christodorescu and Jha, 2003].

Le tecniche di offuscamento si dividono in due grosse categorie: *polimorfiche* e *metamorfiche*. Un malware polimorfo presenta il proprio payload crip-

tato e lo decripta, tramite un apposita routine, esattamente prima dell'esecuzione. Quando il malware si replica, infettando un nuovo sistema, cripta il suo payload con una chiave differente e modifica il codice della routine di decriptazione con l'inserimento di *nop* [1], trasposizione di codice o salti incondizionati. Un malware di tipo metamorfo invece, quando si replica modifica il suo codice in svariate maniere: tramite l'inserimento di nop , con la trasposizione di codice, riassegnazione dei registri etc. Inoltre tale tipologia di malware puoi sparpagliare il suo codice all'interno del programma infettato, rendendo così impossibile il rilevamento di virus signature in quanto il codice è mischiato con il codice del programma [Christodorescu and Jha, 2003, Brand, 2007].

Metamorfismo Il problema del metamorfismo può essere definito formalmente come la ricerca di una sequenza di istruzioni β', appartenente alla versione offuscata $\mathcal{O}(M)$ del malware M, tale che β' sia semanticamente equivalente a β sequenza di istruzioni appartenente a M, che identificano il codice come malevolo.

Di seguito sono proposte le tecniche di offuscamento più comuni utilizzate dai malware metamorfici.

Dead-Code Insertion conosciuta anche come trash insertion, aggiunge codice al programma senza modificarne il comportamento. L'esempio più semplice di dead-code insertion è l'inserimento di istruzioni nop. Offuscamenti più complessi sono costituiti da sequenze di codice e costrutti che modificano lo stato del programma per ripristinarlo subito dopo. Di seguito è proposto un esempio relativo all'inserimento di istruzioni nop nel codice trattato precedentemente.

```
E8 00000000   call 0h
5B            pop ebx
8D 4B 42      lea ecx, [ebx + 45h]
90            nop
51            push ecx
50            push eax
50            push eax
90            nop
0F01 4C 24 FE sidt [esp - 02h]
5B            pop ebx
83 C3 1C      add ebx, 1Ch
90            nop
```

[1] Istruzione che non ha nessun effetto, si limita ad incrementare il program counter di uno

```
FA          cli
8B 2B       mov ebp, [ebx]
```

La virus signatur modificata per riconoscere tale sequenza di istruzioni è la seguente:

```
E800 0000 005B 8D4B 4290 5150
5090 0F01 4C24 FE5B 83C3 1C90
FA8B 2B
```

Anche se la tecnica sembra banale, è da notare che non tutte le sequenze di dead-code possono essere rilevate ed eliminate, dato che il problema è equivalente a quello di: *decidere se una sequenza di istruzioni è equivalente ad un programma vuoto*, che è un problema indecidibile.

Code Trasposition la tecnica consiste nel riorganizzare la sequenza delle istruzioni all'interno del codice, in modo che l'ordine di esecuzione delle istruzioni sia differente dall'ordine con cui le istruzioni compaiono nell'eseguibile o nei virus signature utilizzati dagli antivirus. Un primo offuscamento può essere fatto ordinando in maniera casuale le istruzioni e poi ricostruendo la corretta sequenza tramite dei salti incondizionati, un secondo tipo di offuscamento, può invece consistere nell'effettuare lo swap di due istruzioni che non sono interdipendenti.

La prima tecnica è relativamente semplice da implementare, mentre la seconda è più complicata in quanto le interdipendenze delle istruzioni devono essere accertate. Dal punto di vista dell'analisi, il code trasposition può creare problemi ad un analizzatore umano, ma molti dei tool, utilizzati per l'analisi automatica, inglobano rappresentazioni del codice come il controlo flow graph, oppure il program dependece graph, che sono insensibili ai cambiamenti superflui del control flow del programma.

Di seguito è proposto un esempio di code trasposition relativo al codice analizzato precedentemente.

```
S1:  call 0h
     pop ebx
     jmp S2
S3:  push eax
     push eax
     sidt [esp - 02h]
     jmp S4
S5:  add ebx, 1Ch
```

```
     jmp S6
S2:  lea ecx, [ebx+42h ]
     push ecx
     jmp S3
S4:  pop ebx
     jmp S5
S6:  cli
     mov ebp, [ebx]
```

Register Reassignment Questa tecnica scambia i nomi dei registri e non ha nessun altro effetto sul comportamento del programma. Ad esempio se il regsitro *ebx* è inutilizzato in uno specifico live range del registro *aex*, allora *ebx* può reimpiazzare *aex* per quel live range.

Istruction Substitution questa tecnica utilizza un dizionario di istruzioni equivalenti, per sostituire una sequenza di istruzioni, con una sequenza di istruzioni equivalente. Per gestire questo tipo di offuscamento, il tool di analisi deve mantenere un dizionario di istruzioni equivalenti, simile al dizionario utilizzato per offuscare il codice [Christodorescu and Jha, 2003].

Di seguito è proposto un esempio di code trasposition relativo al codice analizzato precedentemente.

```
call 0h
pop ebx
lea ecx, [ebx+42h]
sub esp, 03h
sidt [esp - 02h]
add [esp], 1Ch
mov ebx, [esp]
inc esp
cli
mov ebp, [ebx]
```

[Christodorescu and Jha, 2003] hanno implementato queste semplici tecniche di offuscamento e hanno dimostrato che antivirus del calibro di Norton non sono in grado di riconoscere la versione offuscata, tramite il solo inserimento di nop, di alcuni famosi malware.

[Gheorghescu, 2009] propongono una tecnica per gestire questo tipo di offuscamenti, che si basa sull'utilizzo di control flow graphs, sulla suddivisione del codice in più parti e sul successivo confronto, tramite opportune metriche, di tali parti di codice con dei virus signatur.

Polimorfismo I malware di tipo polimorfico sono caratterizzati dall'avere il payload compresso o criptato, e dall'avere una routine che si preoccupa di estrarlo/decriptarlo ed eseguirlo. Per riferirsi a malware il cui payload è compresso oppure cifrato si utilizza il termine *packed* in tutte le sue variazioni.

Per decomprimere il payload del malware, la routine di unpacking può essere invocata una sola vola, in tal caso il payload è estratto in memoria in un singolo passo, oppure può essere invocata più volte, quando differenti parti del payload sono estratte in memoria in momenti differenti. Dal punto di vista di un analista, questo significa che il malware deve essere eseguito in un ambiente isolato affinché l'analisi del payload possa essere effettuata. Dal punto di vista di un tool di analisi significa che l'analisi del codice deve essere posposta fino a dopo l'inizio della sua esecuzione, i.e., quando la routine di unpacking ha decriptato il payload.

L'utilizzo di packer permette di aggirare le classiche tecniche di riconoscimento basate sulle signature l'asciando i malware a lungo non rilevati. L'approccio tradizionale per gestire i malware di tipo polimorfico è quello di utilizzare delle specifiche routine di unpacking, una per ogni algoritmo di packing conosciuto, prima di effettuare l'analisi del codice. Va da se che questo tipo di approccio non fornisce ottimi risultati, in quanto i malware possono possedere più strati di packing e non è possibile conoscere tutti gli algoritmi.

[Martignoni et al., 2007] propongono un tool che segue una semplice strategia per gestire i malware polimorfici. Il tool tiene traccia di tutte le scritture in memoria e del program counter. Se il program counter raggiunge un'indirizzo di memoria già scritto allora nel programma è presente una qualche forma di unpacking, *self-modification*, o *code-generation*. La strategia prevede quindi che tutte le locazioni di memoria scritte e poi eseguite, (oppure scritte ed in procinto di essere eseguite) devono essere analizzate da un malware detector.

Analisi dinamica

L'analisi dinamica costituisce la soluzione per superare molte delle limitazioni imposte dall'analisi statica. La tecnica si basa sull'esecuzione del campione in esame in un ambite isolato (dov'è possibile monitorare le chiamate al sistema operativo, il traffico di rete, l'accesso ai registri, al file system etc.), con lo scopo di capire come il malware si comporta e quando lancerà il suo attacco.

[Bayer et al., 2009] hanno monitorato il comportamento di un insieme di malware, il cui target è la piattaforma windows, ed hanno definito un insieme di comportamenti che spesso caratterizzano azioni malevole.

File System activity molti malware interagiscono con il file system, creando o modificando file esistenti.

I malware che creano file, spesso creano file eseguibili o mix di differenti tipi di file temporanei come librerie DLL, sia in cartelle di sistema sia nella cartella utente, indicando così la presenza malware che possono essere eseguiti con i permessi di un utente normale.

I malware che modificano file già presenti nel sistema spesso si limitano a modificare file relativi alla registrazione di eventi.

Registry activity un numero molto significativo di malware aggiunge voci al registro di sistema. La maggior parte delle volte le voci create sono collegate al controllo del settaggio dei network adapter, ma sono presenti anche casistiche in cui i malware creano voci sotto la chiave "*SystemCertificates/TrustedPublisher/Certificates*", installando così il proprio certificato come certificato affidabile. Altre tipologie di malware invece, interagendo con i registri, tentano di disabilitare il firewall di windows, di rendere impossibile l'invocazione del task manager da parte dell'utente, di interagire con i settaggi di sicurezza di windows oppure di inserirsi nella lista di programmi avviati allo start-up, in modo da sopravvivere ai reboot.

Network activity alcuni malware si connettono a siti remoti per scaricare altri file, altri utilizzano SLL associato ad HTTPS per criptare le comunicazioni, anche se spesso la connessione è droppata prima della fine dell'handshake. Inoltre quasi la metà dei malware effettuano query a server DNS e nella maggior parte dei casi il server risponde con successo.

GUI activity sorprendentemente molti malware interagiscono aprendo finestre GUI, la maggior parte delle quali sono semplici message box che si preoccupano di trasmettere un errore di qualche tipo, riducendo il sospetto dell'utente, in quanto un messaggio di errore attira meno attenzione di un eseguibile il cui comportamento non è quello atteso.

Sandbox detection un altro aspetto interessante dei malware è quello relativo alla capacità di individuare la presenza di ambienti di analisi come Anubis. Solitamente quando un malware individua la presenza di una sandbox modifica il suo comportamento o semplicemente termina la propria esecuzione.

Le tecniche di rilevazione delle sandbox ricadono in due categorie *istruction level detection* e *ASI level detection*. Le prime sono tecniche che si basano sulle differenza presenti tra una CPU reale ad una CPU emulata, mentre le seconde effettuano delle query sull'ambiente tramite l'invocazione di API.

2.3 Origine del problema del naming

All'inizio del 1990 il problema dell'inconsistenza dei nomi era relativamente contenuto, i vendor cercavano di mantenere una certa correlazione all'interno dei loro database, dizionari e soluzioni di terze parti (e.g., Vgrep) erano sufficienti a permettere un mapping tra i differenti nomi riducendo il problema ad una semplice operazione di associazione. In questo periodo il problema dei virus era decisamente contenuto, gli scrittori di codice malevolo erano guidati più dalla ricerca di fama e di notorietà rispetto che da un ROI, i malware si diffondevano con relativa lentezza e non subivano modifiche o variazioni significative nel loro codice, le scansioni si basavano su tecniche di analisi statica ed un aggiornamento settimanale degli antivirus era sufficiente.

L'avvento di internet e il suo sempre più comune utilizzo, hanno evoluto lo scenario precedente in un nuovo scenario, dove la propagazione dei malware avviene molto più rapidamente (il Worm "*Sapphire/SQL*" ha raggiunto il suo picco di infezione circa dopo 10 minuti dal suo lancio, duplicandosi sulla rete circa ogni 8.5 secondi [Christodorescu and Jha, 2003]) e gli scrittori di codice malevolo sono più orientati ad azioni che possano generare utile rispetto che alla pura fama. In questo nuovo scenario prende il via il gioco dell'offuscamento-deoffuscamento, dove i malware modificano il proprio codice ad ogni propagazione o lo criptano/comprimono con runtime packers. La percentuale dei malware packed aumenta sempre di più, passando dal 29% nel 2003 al 35% nel 2005 all'80% nel 2007, la situazione è inoltre peggiorata dalla facilità con cui internet permette di ottenere e modificare i sorgenti dei vari algoritmi di packing, oggigiorno più del 50% dei malware sono semplicemente versioni repacked di malware già esistenti (in Figura 2.1 è fornito un esempio dei packer più comuni) [Martignoni et al., 2007].

L'avvento di queste nuove tipologie di malware, in grado di evadere le classiche tecniche di scansione signature based, ha costretto i produttori di antivirus ad evolvere i propri software. Inoltre il sempre maggior numero di nuovi malware creati ogni giorno, ha reso inutilizzabili le tecniche d'identificazione esatta, in quanto il tempo necessario per effettuare la scansione di ogni singola nuova versione ha reso il processo infattibile, senza contare inoltre che sarebbe necessario fornire un infinità di aggiornamenti giornalieri ai software di scan. In questo nuovo scenario ha preso sempre maggior piede l'utilizzo di signature generiche in grado di riconoscere famiglie intere di malware e di euristiche avanzate che si basano sul comportamento dei malware per effettuarne l'identificazione.

Tutto questo ha aggiunto un nuovo strato di complessità nell'assegnazione dei nomi, l'utilizzo estensivo di euristiche ha reso impossibile il matching tra i rilevamenti di compagnie differenti, dato che, come è ovvio aspettarsi, compagnie differenti utilizzano algoritmi ed euristiche differenti.

In generale anche quando un label sembra estremamente specifico (ad

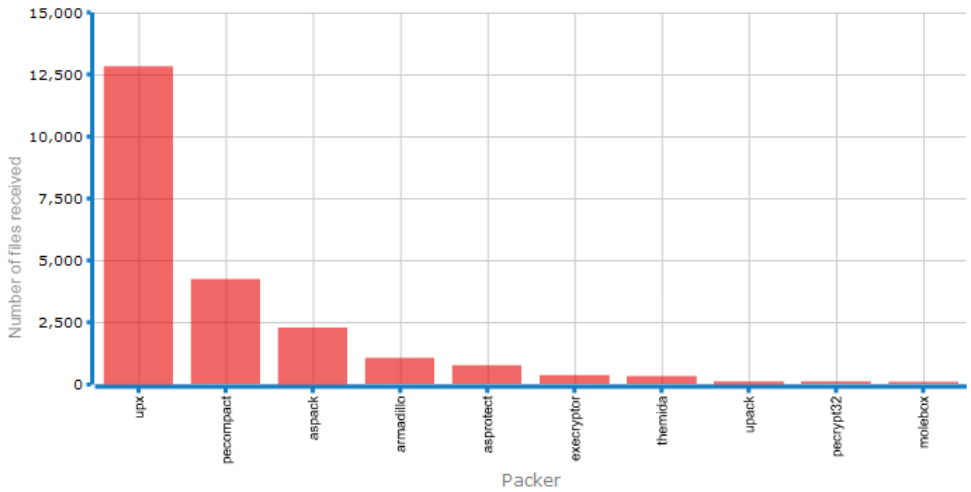


Figura 2.1: Dieci packer più frequenti presenti nelle scansioni di VirusTotal relative al 19 Febbraio 2011 [VirusTotal, 2011].

esempio "*WIN32/Allaple.C*") può racchiudere centinaia o addirittura migliaia di varianti o sotto-varianti di uno stesso campione. Tali varianti posso condividere o meno parti di codice, e possono comparire molto differenti a scansioni di tipo signature based (resistenti ad offuscamenti etc.), mentre possono comparire uguali ad euristiche che si basano sul comportamento del malware. Di contro è anche possibile che a due malware molto simili, scansionati tramite due euristiche che definiscono regole differenti, siano associati label differenti. Ad esempio varianti di "*Conficker*" possono essere classificate in base all'utilizzo (o non utilizzo) del file Autorun. Alcuni vendor potrebbero addirittura decidere di raggruppare sotto un unico label tutti i malware che utilizzano il file autorun (tra cui anche alcune versioni di "*Conficker*"), mentre altri potrebbero assegnare la stessa label a tutte le versioni di "*Conficker*" generando così inconsistenza [Harley, 2009].

2.4 Soluzioni proposte al problema del naming

Negli anni passati, per superare o tentare di arginare il problema dell'inconsistenza dei nomi, sono stati sviluppati dizionari, standard e soluzioni di terze parti, che però non hanno avuto molto successo. Di seguito è fornita una panoramica sulle più importanti.

Vgrep Software messo a disposizione da virus bullettin, che permette di effettuare un cross-reference tra i nomi dei vari vendor. Vgrep funzionava relativamente bene prima dell'esplosione del numero di malware e dell'avvento del problema del naming [Gordon, 2003]. Nel 2005 il suo

database era costituito da circa 250.000 classi di nomi (righe) generate dai 20 antivirus più comuni (colonne) [Gheorghescu, 2009].

CME (Common Malware Enumeration) CME si basa sull'idea di rappresentare una singola minaccia, a cui è assegnato un unico ID, attraverso un set di campioni rilevanti (sia malevoli, sia dati infettati da software malevoli).

In questo modo un utente può confrontare un proprio campione con i vari insiemi e ricavare l'ID unico della minaccia al quale il campione appartiene. Se il campione non appartiene a nessun insieme è profondamente analizzato ed inserito nell'insieme corretto, oppure viene creato un nuovo insieme ed un nuovo ID per classificare la nuova minaccia.

Questo approccio è stato compromesso dalla rapidità con cui giornalmente compaiono nuove famiglie, versioni e varianti di malware.[Pierre-Marc Bureau, 2008] Inoltre tale approccio non risolve il problema ma lo aggira, passando da un insieme di nomi associati ad un unico malware, ad un insieme di campioni associati a tale malware.[Harley, 2009]

CARO è uno standard per la creazione di nomi dei malware, tale standard è stato cambiato più volte venendo incontro alle necessita dei vari vendor. La forma generale di un nome sotto CARO è:

```
<malware type>://<platform>/<family name>.<group
name>.<infective leght>.<sub variant><devolution><modifiers>
```

Avere però uno standard, che definisce la grammatica con cui deve essere generato un nome, è la soluzione a solo una piccola parte del problema.

Il reale problema è come riconciliare i nomi che non fanno parte dello standard e come assegnare dei nomi così specifici e consistenti in un ambiente in cui il tempo è una dimensione critica ed in cui le moderne tecniche di scansione tendono sempre più a fornire informazioni generiche [Gordon, 2003].

Un altro importate problema riguarda l'effettiva utilizzabilità di un nome come quello generato da CARO, che cerca di codificare tutte le informazioni relative ad un malware in un identificativo, rendendolo troppo lungo ed ingombrante per un suo reale utilizzo [Ivan Kirillov].

Wild List è un elenco di virus ricavati "sul campo" fondato nel 1993 da un ricercatore statunitense, il cui scopo era quello di tener traccia e capire quali e quanti fossero i virus in circolazione. L'elenco anche se utile nella prima metà degli anni 90 presenta una serie di problematiche relative al design, che con il passare degli anni si sono trasformate in veri e propri problemi.

- La Wild List contiene solamente alcuni particolari tipi di software malevoli: quelli capaci di replicarsi da se infettando altri file e Pc in un ambiente di rete (Virus, Worm, alcuni tipi di Bot), mentre non copre tutte le altre tipologie e.g.: Trojan, Adware, Spyware, Dialer etc.
- La Wild List non è disponibile alle compagnie di AV e tester subito dopo la fine di un periodo di report, ma indicativamente dopo 40 giorni, questo comporta che la Wild List non riflette la situazione attuale.
- I reporter sono poco numerosi e appartengono a compagnie di AV, quindi non sono interessati ad effettuare un report di un malware che il proprio AV non riconosce, in quanto la Wild List è poi utilizzata come banckmark per fornire certificazioni sugli AV. [Andreas Marx, 2007]

Le soluzioni sinora proposte, non hanno avuto molto successo perchè necessitano di label specifici, derivati da identificazioni esatte, mentre i vendor si spostano sempre più sul fornire label generiche, derivate da identificazioni generiche che si baso su euristiche e comportamenti generici. Nel seguito del documento si dimostra che il problema non è mitigabile attraverso la generazione di una corrispondenza tra i vari label, ma è necessaria una soluzione che risolva le differenze semantico-strutturali e che non necessiti di modifiche radicali delle convenzioni di generazione dei nomi. Inoltre tale soluzione non dovrebbe essere influenzata da eventuali interessi nel confronto della qualità dei vari antivirus.

Capitolo 3

Una tecnica di rilevazione delle inconsistenze

In questo capitolo è descritto un approccio che permette, in maniera pratica ed efficace, di fornire una visione d'insieme sul livello di (in)consistenza tra i sistemi di naming.

Il metodo proposto prende analizza un insieme di malware (identificati dai loro codici hash, e.g., MD5 o SHA-1) ed i label a loro associati, genera una comparazione grafica e quantitativa delle strutture dei sistemi di naming, calcola due indicatori in grado di definire il livello globale di (in)consistenza ed estrae i sottoinsiemi di malware etichettati in maniera inconsistente.

L'approccio proposto, schematizzato in Figura 3.1 , è suddiviso in due fasi:

- **Fase 1**: per ogni vendor in esame i malware sono raggruppati (o clusterizzati) in base ai label che ne codificano nome e caratteristiche. Il processo di clustering è definito in dettaglio nella Sezione 3.1.2.
- **Fase 2**: i raggruppamenti generati nella fase precedente sono utilizzati come modelli di confronto per il calcolo di due indici sintetici e l'estrazione dei gruppi di malware etichettati in maniera inconsistente, come dettagliato in Sezione 3.2.

L'idea alla base di questo approccio deriva dalla necessità di ricercare inconsistenze di tipo semantico-strutturali, andando quindi oltre la visualizzazione delle note differenze sintattiche che compaiono all'interno dei label associati allo stesso malware. Ad esempio, prendendo in esame un insieme di malware, se all'interno dei loro label sono presenti pattern tipo: "*Backdoor*", "*Backdoor.Dialer*" e "*Packed.Backdoor*" si può assumere, che per il vendor in esame, i malware in analisi siano tutti caratterizzati dall'essere dei Backdoor, e che all'interno della **Fase 1** vengano raggruppati tutti in uno stesso insieme (o cluster). In altre parole, un sistema di naming è modellato tramite i raggruppamenti indotti dai label su un insieme di malware. Tale modello è generato

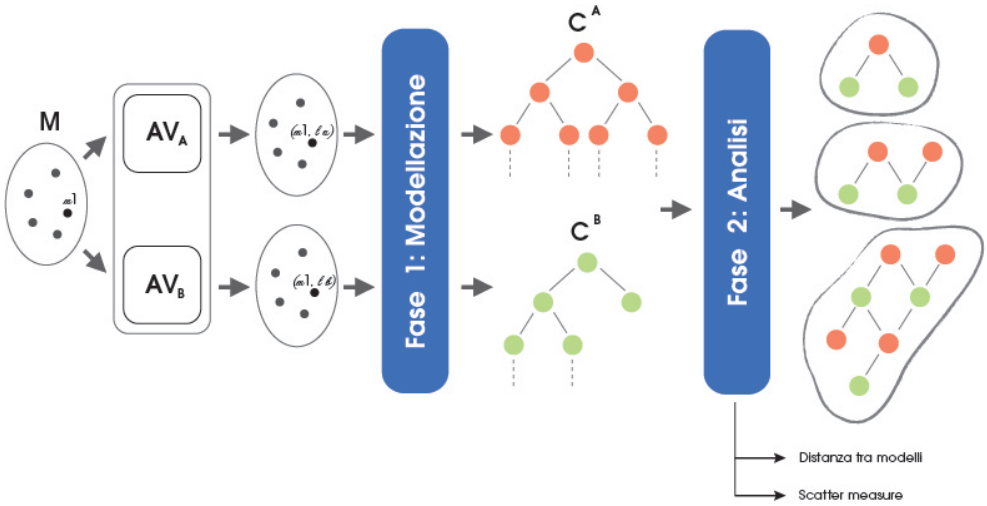


Figura 3.1: Processo di analisi dell'inconsistenza tra due sistemi di namig. Il processo prende in input un insieme di malware ed attraverso gli antivirus gli associa i label. La fase 1 del processo modella i sistemi di naming basandosi sull'insieme di malware fornito in input. La fase due confronta ed analizza i modelli generati nella fase precedente ed estrae le (in)consistenze e due indici analitici.

per ogni vendor e in **Fase 2**, due sistemi di naming si considerano consistenti se raggruppano nella stessa maniera lo stesso insieme di malware, indipendentemente dalle differenze sintattiche presenti nei label. Ad esempio, un cluster costituito da due malware $\{m_1,m_2\}$ aventi associati rispettivamente i label "*foo-backdoor*" e "*bla-backdoor*", è consistente con un cluster contenente esattamente gli stessi malware $\{m_1,m_2\}$, aventi associati due label differenti e.g., "*bar-trojan*" e "*tar-trojan*".

Per illustrare meglio il processo di analisi proposto in seguito, si anticipa il concetto di inconsistenza, poi formalizzato in Sezione 3.2.3. Intuitivamente dati due vendor ed un set di malware si possono definire e ricercare due tipi di inconsistenze:

- **inconsistenze deboli**: si presentano quando uno dei due vendor divide i malware in esame in più insiemi, mentre l'altro li raggruppa in un unico insieme, realizzando così una relazione di tipo uno-a-molti, in opposizione ad una relazione di tipo uno-a-uno che indicherebbe consistenza. Quest'inconsistenza non è da considerarsi grave, in quanto è generata dal fatto che uno dei due vendor inserisce all'interno dei suoi label un'informazione più fine e dettagliata.

- **inconsistenze forti**: si presentano quando i vendor suddividono l'insieme di malware in più gruppi, in modo tale che non possano essere

ricondotti ad un inconsistenza debole.

Va inoltre precisato che la **Fase 2** del processo è stata disegnata in modo da essere completamente indipendente dalla **Fase 1**, di conseguenza non è necessario effettuare il raggruppamento dei malware in base ai label, ma è possibile effettuarlo a partire da qualunque altra caratteristica di cui si vuole valutare l'inconsistenza. In alternativa è possibile raggruppare i malware tramite l'utilizzo di tool automatici (e.g., Anubis [Bayer et al., 2009]) oppure effettuare un raggruppamento a mano.

3.1 Fase 1: generazione del modello del sistema di naming

L'obbiettivo di questa fase è quello di generare un modello per i sistemi di naming, l'idea principale che sta alla base del processo è quella di raggruppare i malware in base al valore assunto dai loro label (i.e. la stringa che in accordo con il sistema di naming di uno specifico vendor ne codifica il nome).

Per effettuare il raggruppamento è stato adottato un sistema clustering gerarchico di tipo top-dow (descritto in Sezione 3.1.2) che, partendo da un insieme (o cluster) di malware, lo suddivide in maniera ricorsiva in sotto-cluster nidificati. L'approccio fornito garantisce che all'interno di un insieme vi siano solamente malware i cui label condividono pattern simili. I pattern sono estratti offline (come descritto in Sezione 3.1.1) e da qui in in avanti saranno identificati utilizzando il `teletype font`.

3.1.1 Estrazione dei pattern

La tecnica proposta si basa sull'identificazione di quatto classi di pattern, che da qui in avanti verranno indicate tra parentesi angolate in teletype font (e.g., `<classi>`).

- threat `<type>` racchiude i pattern che identificano il tipo di attività eseguita dal codice malevolo. Questa classe contiene pattern come `Backdoor`, `Trojan`, `Packed`, `Tool` etc.

- `<family>` racchiude i pattern che identificano la famiglia alla quale il malware appartiene. In questa classe sono contenuti pattern come `Conficker`, `Allaple`, `Mudrop`, `Fokin` etc.

- `<platform>` racchiude i pattern che identificano il target del codice malevolo ad esempio `WNT`, `W32`, `iPhoneOS`, oppure i pattern che identificano il tipo di linguaggio necessario all'esecuzione del malware o quello di cui sfruttano le vulnerabilità, ad esempio `PHP` o `JS`.

- `<version>` racchiude i pattern che forniscono informazioni sulla versione del malware (e.g., le stringhe `A` e `D` all'interno dei label "*PHP:IRCBot-B*" e "*PHP:IRCBot-D*"), oppure i pattern che forniscono informazioni aggiuntive, come il fatto che la signature utilizzata per il riconoscimento sia generica, o che il file sia danneggiato, ad esempio `Gen`, `Gen44`, `damaged`.

Le classi appena definite, permettono di analizzare virtualmente, tutta l'informazione contenuta nei label generati dalla maggior parte dei vendor. L'approccio risulta comunque totalmente flessibile ed estendibile: nuove classi possono essere aggiunte e classi presenti possono essere rimosse o ignorate. Un esempio della flessibilità dell'approccio sarà fornito durante la generazione del modello di V_4 che, a differenza degli altri vendor presi in esame, nei suoi label non fornisce alcuna informazione sul tipo. Di conseguenza, per V_4, risulta impossibile estrarre pattern appartenenti alla classe `<type>` che sarà quindi ignorata.

In base alle analisi effettuate, i pattern contenuti nei label possono essere di due tipi:

- **pattern semplici**: un pattern è definito semplice quando è l'unica sottostringa del label ad appartenere ad una determinata classe. Ad esempio all'interno del label "*PHP:IRCBot-B*", `IRCBot` è un *pattern semplice* appartenente alla classe `<family>`. In generale è stato notato che le classi che contengono solamente pattern semplici sono la classe `<family>` e la classe `<platform>`.

- **gerarchia di pattern semplici**: all'interno di un label è presente una *gerarchia di pattern semplici* se vi sono presenti più sotto-stringhe (o pattern) che appartengono alla stessa classe, questo può accadere quando i vendor specificano un tipo ed un sotto-tipo per i malware. Ad esempio all'interno del label "*Trojan.Dropper.Rustock.B*" i due *pattern semplici* `Trojan` e `Dropper` appartengono entrambi alla classe `<type>` e costituiscono una gerarchia di pattern semplici. Una gerarchia di pattern semplici è identificata dalla sequenza dei pattern semplici che la compongono, separati dal carattere "/" (e.g., `pattern` = `pattern1 / pattern2 / pattern3`). É importante notare che, la sequenza dei pattern semplici, all'interno della gerarchia di pattern semplici, non deve necessariamente rispecchiare la sequenza delle sotto-stringhe all'interno del label.

  L'introduzione di questo tipo di pattern comporta la suddivisione delle classi, in cui queste pattern compaiono, in classi di tipo gerarchico (e.g., `<type>` = `<type1>/<type2>/<type3>`). Continuando l'esempio precedente, per la gerarchia di pattern semplici identificata dalla stringa `Trojan/Dropper`, i pattern semplici `Trojan` e `Dropper` sono contenuti

rispettivamente in `<type1>` e `<type2>`. Oltre alla classe `<type>`, anche la classe `<version>` dimostra una buona predisposizione al contenimento di gerarchie di pattern semplici.

I pattern semplici possono essere ricavati a mano a partire da un insieme ristretto di label, oppure possono essere ricavati tramite l'utilizzo di dispositivi di inferenza che derivano la sintassi più probabile di un set di stringhe. Per ottenere risultati accurati, in entrambi i casi, risulta necessario un intervento o una revisione manuale, di conseguenza, si è optato per la scelta di un'euristica (definita in Sezione 3.1.1) che permette l'estrazione dei pattern in maniera semi-automatica.

Per l'ordinamento dei patter all'interno della gerarchia di pattern semplici è stato invece adottato un semplice criterio: i *pattern semplici* sono ordinati da sinistra verso destra secondo la loro ricorrenza all'interno del dataset, se la frequenza è la stessa la gerarchia di pattern semplici è gestita come un pattern semplice. Continuando l'esempio precedente il pattern `Trojan` avendo una frequenza maggiore del pattern `Dropper` genera la gerarchia di pattern semplici `Trojan/Dropper`.

3.1.2 Clustering

Il processo di clustering è eseguito per ogni vendor: riceve in ingresso un set di pattern, opportunamente associati alla corretta classe, ed un insieme di malware $\mathcal{M}$ con associati i rispettivi label. L'algoritmo suddivide in maniera ricorsiva l'insieme di malware di partenza $\mathcal{M}$ in più sottoinsiemi, utilizzando come criterio di suddivisione la presenza nei label dei pattern precedentemente estratti. Per effettuare la suddivisione è presa in considerazione una classe di pattern alla volta, quando l'insieme non è più suddivisibile secondo i pattern presenti nella classe in analisi, l'algoritmo passa alla classe successiva. La sequenza con cui sono prese in considerazione le classi è la seguente: `<type>`, `<family>`, `<platform>` e `<version>`.

Il modo migliore per spiegare l'esecuzione dell'algoritmo è tramite un esempio. Iniziando la clusterizzazione dall'insieme di partenza $\mathcal{M}$ la prima suddivisione è indotta dai pattern contenuti nella classe `<type>`. Nell'esempio mostrato in Figura 3.2, nella classe `<type>` sono contenuti il pattern semplice `Backdoor` e le gerarchie di pattern semplici `Trojan/Dropper` e `Trojan/Downloader`, di conseguenza, la prima suddivisione raggruppa i malware denominati "*Backdoor.Perl.Shellbot.cd*" e "*Backdoor.PHP.Shellbot.t*" nello stesso cluster, mentre i malware "*Trojan-Downloader.Win32.Fokin.da*", " *Trojan-Dropper.Win32.Mudrop.fkt*" e "*Trojan-Dropper.Win32.Mudrop.jts*" vengono raggruppati in un cluster differente. A questo punto se il vendor in esame adotta pattern di tipo gerarchico la procedura è ripetuta per ogni sottoclasse. Continuando l'esempio precedente il cluster *Trojan* viene suddiviso in due sotto-cluster: *Downloader* che contiene il malware etichettato come "*Trojan-*

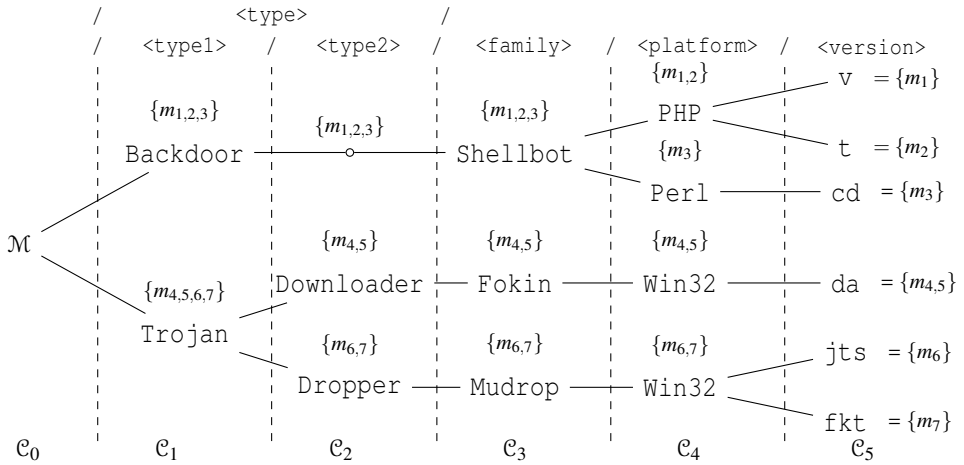


Figura 3.2: Esempio di output della **Fase 1** su un insieme $\mathcal{M}$ costituito da sette campioni. $\mathcal{C}_1 = \{\{m_1, m_2, m_3\}, \{m_4, m_5, m_6, m_7, \}\}$ e $\mathcal{C}_{2-4} = \{\{m_3\}, \{m_6, m_7\}, \{m_4, m_5\}, \{m_1, m_2, m_3\}, \{m_1, m_2\}\}$. Notare che, `<type>` comprende `/<type1>/<type2>/`.

Downloader.Win32.Fokin.da" e *Dropper* che contiene i malware etichettati come "*Trojan-Dropper.Win32.Mudrop.jts*" e " *Trojan-Dropper.Win32.Mudrop.fkt*".

Quando un cluster, non può più essere suddiviso in base ai pattern presenti nella classe in esame, l'algoritmo passa ad analizzare la classe successiva. Continuando l'esempio, la classe successiva da prendere in analisi è `<family>`, la quale contiene i pattern `Fokin`, `Mudrop`, e `Shellbot` che non inducono ulteriori suddivisioni. Successivamente il primo cluster generato (i.e., *Backdoor*) è suddiviso in due cluster uno contenente il malware denominato "*Backdoor.Perl.Shellbot.cd*", e l'altro contenente i malware denominati "*Backdoor.PHP.Shellbot.t*" e "*Backdoor.PHP.Shellbot.v*", in accordo con i pattern presenti nella classe `<platform>`. Dopo aver analizzato la classe `<platform>`, l'algoritmo passa ad analizzare la classe `<version>`, che induce gli ultimi split visualizzati in Figura 3.2.

La procedura di clustering si arresta quando tutte le possibili suddivisioni sono state effettuate, ovvero quando l'ultima classe è stata considerata. Ad ogni suddivisione, è inoltre tenuta traccia della relazione tra insieme padre ed insieme figlio, in modo da poter costruire un albero di cluster la cui radice è l'insieme di partenza $\mathcal{M}$.

3.1.3 Albero dei cluster

Dato un set di malware $\mathcal{M}$, l'output della **Fase 1** è un albero di cluster $\mathcal{C} = \mathcal{F}(\mathcal{M})$, di tale albero è possibile selezionare un sottoinsieme $\mathcal{C}_d \subset \mathcal{C}$ degli elementi che lo compongono, dove d può essere: un numero che indica la profondità dell'albero (e.g., $\mathcal{C}_1$), un intervallo tra due livelli dell'albero

(e.g. $\mathcal{C}_{2-4}$), oppure un'espressione mnemonica che identifica un insieme di elementi, o un singolo elemento.

Considerando sempre l'esempio in Figura 3.2, $\mathcal{C}_1 = \{\{m_1, m_2, m_3\}; \{m_4, m_5, m_6, m_7\}\}$, $\mathcal{C}_{2-4} = \{\{m_3\}; \{m_6, m_7\}; \{m_4, m_5\}; \{m_1, m_2, m_3\}; \{m_1, m_2\}\}$, mentre l'albero completo è definito come $\mathcal{C}_0$ o $\mathcal{C}^V$ dove V è il nome del vendor in esame.
Continuando ad esaminare l'esempio fornito in Figura 3.2, i cluster al terzo livello di profondità possono essere indicati tramite l'utilizzo dell'espressione mnemonica */*/family*, che indica la clusterizzazione indotta su $\mathcal{M}$ dalla classe `<family>`, e dalle classi che la precedono i.e., $\mathcal{C}_{/*/family} = \mathcal{C}_3 = \{\{m_1; m_2; m_3\}; \{m_4; m_5\}; \{m_6, m_7\}\}$. Come espressioni mnemoniche, possono essere utilizzate anche stringhe in grado di identificare opportuni cluster e.g., *Backdoor* identifica l'insieme $\mathcal{C}_{Backdoor} = \{\{m_1, m_2, m_3\}\}$. Un ultima considerazione va fatta sulle classi di tipo gerarchico, l'utilizzo dell'espressione */*/type_x* indica la clusterizzazione indotta su $\mathcal{M}$ dalla classe `<type_x>`, dove x è il numero che indica la posizione della sottoclasse all'interno gerarchia, mentre */*/type_n* identifica il livello più basso della gerarchia. Tenendo sempre come riferimento la Figura 3.2 $\mathcal{C}_2 = \mathcal{C}_{<type_2>} = \mathcal{C}_{<type_n>} = \{\{m_1, m_2, m_3\}; \{m_4, m_5\}; \{m_6, m_7\}\}$.

3.1.4 Dettagli implementativi

Questa sezione ha lo scopo di fornire una serie di dettagli implementativi, che permettono di descrivere la **Fase 1** del processo di analisi nella sua completezza. La sezione si suddivide in due paragrafi: il primo descrive in maniera dettagliata l'euristica utilizzata per l'estrazione dei pattern, mentre il secondo si occupa di analizzare casistiche particolari che possono venirsi a creare durante il processo di clustering.

Estrazione dei pattern L'euristica utilizzata per l'estrazione delle pattern è un algoritmo semi-supervisionato, eseguito per ogni vendor, che prende in input: (1) un insieme di label $\mathcal{L}$, (2) un ridotto insieme di caratteri separatori $\mathcal{Z}$ (e.g., [. ! : _ / -]), e procede a suddividere, in base ai caratteri contenuti in $\mathcal{Z}$, i label in pattern e successivamente ad assegnarli alla corretta classe.

Inizialmente le classi sono vuote (e.g., `<type>`=''), alla i-esima iterazione è selezionato un piccolo sottoinsisme di label (e.g., 10) $\mathcal{L}_i \subset \mathcal{L}$. I label in esso presenti sono suddivisi in pattern, in accordo con i caratteri separatori contenuti in $\mathcal{Z}$, successivamente l'operatore associa ogni nuovo pattern estratto la corretta classe.

Ad esempio se all'i-esima iterazioni sono estratti i pattern `win32`, `PHP`, `Allaple`, `Trojan` le rispettive classi sono aggiornate come segue: `<platform>` = `<platform>`$_{i-1} \bigcup \{$`win32,PHP`$\}$, `<family>` = `<family>`$_{i-1} \bigcup \{$`Allaple`$\}$, `<type>` = `<type>`$_{i-1} \bigcup \{$`Trojan`$\}$.

Ogni pattern estratto dai label $l \in \mathcal{L}_i$ deve essere assegnato esattamente ad una sola classe. Se esistono label che possiedono almeno un pattern che non è assegnato ad alcuna classe, tali label sono posposti, eliminati da $\mathcal{L}_i$ e reinseriti in $\mathcal{L}$. In alternativa l'operatore può decidere di espandere $\mathcal{Z}$, inserendo nuovi caratteri di separazione, in modo da poter estrarre in maniera corretta i pattern dal label ed associarli alle corrette classi.

Quando tutti i label presenti in $\mathcal{L}_i$ sono stati analizzati, l'insieme $\mathcal{L}$ è ridotto, eliminando tutti i label che possono essere interamente processati con i pattern già presenti nelle classi. Successivamente dall'insieme $\mathcal{L}$ è prelevato un altro insieme casuale $\mathcal{L}_{i+1}$. La procedura si ripete fino al raggiungimento della condizione $\mathcal{L} = \varnothing$.

La velocità e l'accuratezza della procedura, oltre che dalla dimensione di $\mathcal{L}_i$, dipendono fortemente dalla capacità e dall'esperienza dell'operatore, che deve assegnare i pattern estratti all'opportuna classe. Va notato che la procedura, anche se può sembrare lenta e laboriosa, deve essere effettuata solo una volta per ogni vendor, ed inoltre, i pattern estratti per un vendor, possono essere riutilizzati per l'estrazione dei pattern di un altro vendor, in quanto, spesso per le classi `<platform>`, `<type>` ed in maniera ridotta anche per `<family>` vendor differenti tendono ad usare le stesse sottostringhe simili.

L'algoritmo può inoltre essere reso molto più efficiente, ed il grado di automazione può essere notevolmente aumentato, a patto che i label in analisi siano ben strutturati (almeno per alcune delle loro parti) e mantengano tale struttura sulla maggior parte dei campioni presenti nel dataset. Ad esempio, se è noto che la sotto-stringa contenuta tra i caratteri separatori ":" e " / " indica la piattaforma, non è necessaria la revisione manuale per l'estrazione dei pattern appartenenti alla classe `<platform>`, ma basta una semplice scansione dei label. In una situazione reale però, va prestata attenzione a quei label che non possiedono un struttura tale da essere suddivisi in pattern (label scritti a mano da eventuali analisti), come ad esempio "*Ashampoo Firewall Stealthing Component*". Per questi campioni è necessaria un'ispezione dell'albero di cluster ed un inserimento manuale del campione in analisi all'interno del corretto cluster, al fine di non generare errori grossolani. Nel dataset utilizzato per la generazione dei risultati sperimentali questa casistica è stata poco ricorrente, ma non del tutto assente.

In Figura 3.3 è riportato un esempio di estrazione dei label, che esemplifica il problema dei label mal strutturati e l'efficienza della della fase di riduzione.

Clustering La fase di clustering, così com'è stata presentata nella Sezione 3.1.2, può portare alla generazione di cluster contenenti solamente un elemento. Tali cluster non possono essere considerati significativi per modellare il sistema di naming di un vendor, quindi è stata introdotta la seguente regola: se la suddivisione di un cluster, secondo i pattern presenti nella classe

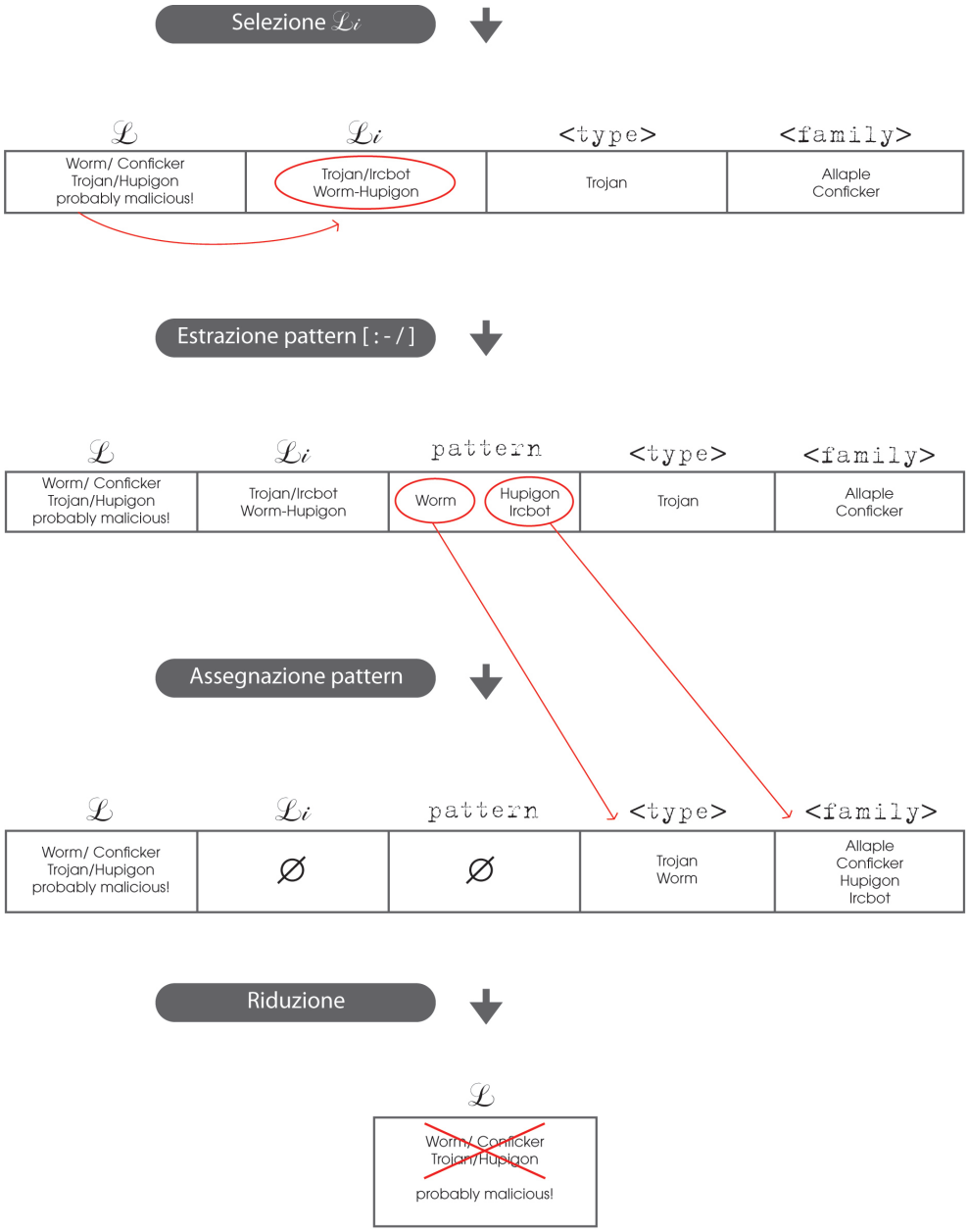


Figura 3.3: Esempio di estrazione pattern a partire da un insieme di label. Dopo l'i-esimo passo di estrazione, i pattern presenti nelle classi permettono di ridurre $\mathcal{L}$. Notare la presenza di un label scritto particolarmente male che dovrà essere processato a mano

in analisi, crea cluster di dimensioni inferiori a $\mathcal{P}$, tali cluster sono uniti in un unico cluster denominato $\mathcal{C}_{misc}$ che codifica un certo grado di incertezza, inoltre, tale cluster non verrà suddiviso e analizzato ulteriormente.

Riprendendo l'esempio di Figura 3.2 la suddivisione del cluster $\mathcal{C}_{shellbot}$ porterà alla generazione del cluster $\mathcal{C}_{PHP} = \{m_1, m_2\}$ e del cluster $\mathcal{C}_{misc} = \{m_3\}$, inoltre il cluster $\mathcal{C}_{Mudrop/win32}$ non potrà essere ulteriormente suddiviso, in quanto, non permette la generazione di cluster di dimensioni maggiori di $\mathcal{P}$ (in questo progetto è stato scelto $\mathcal{P} = 2$).

L'idea alla base della regola sta nel fatto che, se i vendor sono consistenti allora genereranno alberi di cluster tra loro simili/coincidenti, e di conseguenza tenderanno a raggruppare gli elementi "spuri" all'interno di cluster $\mathcal{C}_{misc}$ tra loro simili/coincidenti.

3.2 Fase 2: confronto dei modelli ed analisi

In questa fase del processo sono utilizzati gli alberi di cluster, generati in **Fase 1** (Sezione 3.1), come modelli dei sistemi di naming da confrontare. Dati due vendor A e B, la loro comparazione avviene tramite la comparazione dei due alberi di cluster $\mathcal{C}^A$, $\mathcal{C}^B$ (definiti in Sezione 3.1.3).

L'output di questa fase consiste in due indicatori numerici: *distanza tra sistemi di naming* e *scatter measure* (definiti rispettivamente in Sezione 3.2.2 ed in Sezione 3.2.1), che forniscono un idea sul livello globale di (in)consistenza presente tra due sistemi di naming. Oltre al calcolo dei due indicatori, in **Fase 2**, sono anche estratte le inconsistenze tra i sistemi di naming, introdotte nelle sezioni precedenti, la cui definizione formale è fornita in Sezione 3.2.3.

Va ricordato che l'albero di cluster costituisce la rappresentazione della gerarchia dei cluster (i.e., tiene traccia delle relazioni presenti tra insieme padre ed insieme figlio), le seguenti analisi sono effettuate su insiemi di cluster derivati dal taglio dell'albero a differenti livelli di profondità (come esemplificato in Sezione 3.1.3). Per alleggerire la notazione, nelle successive sezioni di questo capitolo, il livello di profondità d, che definisce il taglio dell'albero dal quale derivano gli insiemi di cluster utilizzati nell'analisi, è omesso a favore della seguente notazione $\mathcal{C}^A_d = \mathcal{C}^A$ e $\mathcal{C}^B_d = \mathcal{C}^B$.

Due delle analisi definite in questo capitolo: la scatter measure e l'*estrazione delle inconsistenze* (Sezione 3.2.2 e Sezione 3.2.3), richiedono che i sottoinsiemi $\mathcal{C}^A_d$ e $\mathcal{C}^B_d$ siano partizioni di $\mathcal{M}$. Ovvero richiedono che: presi due cluster $c, c' \in \mathcal{C}_d$ la loro intersezione sia nulla $c \bigcap c' = \varnothing$ e che: l'unione di tutti i cluster appartenenti al taglio contenga tutti gli elementi del dataset $\bigcup_{c_i \in \mathcal{C}_d} c_i = \mathcal{M}$.

Il calcolo della *distanza tra due sistemi di naming* (Sezione 3.2.1) non richiede invece nessun tipo di condizione sui cluster che costituiscono gli insiemi $\mathcal{C}^A_d$ e $\mathcal{C}^B_d$.

3.2.1 Distanza tra due sistemi di naming

In questa sezione è definita la distanza tra due sistemi di naming $D(\mathcal{C}^{\mathcal{A}}, \mathcal{C}^{\mathcal{B}})$, tale misura è definita come la distanza media tra i cluster che costituiscono gli insiemi $\mathcal{C}^A$ e $\mathcal{C}^B$. Più precisamente:

$$D(\mathcal{C}^A, \mathcal{C}^B) := \frac{1}{2}\left(\frac{\sum_{c\in\mathcal{C}^A}\delta(c,\mathcal{C}^B)}{|\mathcal{C}^A|} + \frac{\sum_{c\in\mathcal{C}^B}\delta(c,\mathcal{C}^A)}{|\mathcal{C}^B|}\right) \tag{3.1}$$

dove $\delta(c, \mathcal{C}') = \min_{c'\in\mathcal{C}'} d(c, c'), \forall c \in \mathcal{C}$ è la minima distanza tra l'insieme $c \in \mathcal{C}$ ed ogni altro possibile insieme presente in $\mathcal{C}'$ secondo la metrica $d(\cdot,\cdot)$. Il denominatore è un fattore di normalizzazione che assicura che $D(\cdot,\cdot)$ ricada nell'insieme $[0, 1]$. Il processo di calcolo di $D(\mathcal{C}^A, \mathcal{C}^B)$ è esemplificato dall'Algoritmo 1.

Una definizione di distanza simile è stata utilizzata da [Mark K.Goldberg, 2010] per calcolare la similitudine tra set di cluster overlappanti, in questa tesi è tuttavia proposto un fattore di normalizzazione differente. Altri tipi di normalizzazione sono analizzati in Sezione 3.2.4.

Per la definizione di $d(c, c')$, distanza tra due cluster, sono state scelte le seguenti metriche:

- $d_J(c, c') = 1 - J(c, c')$ dove $J(c, c')$ è l'indice Jaccard tra i cluster c e c' ovvero $J(c, c') = \frac{|c\bigcap c'|}{|c\bigcup c'|}$.

- $d_H(c, c') = \frac{H(X_c|X_{c'})}{H(X_c)}$, dove $H(X_c|X_{c'})$ è l'entropia condizionata, e $H(X_c)$ è l'entropia della variabile casuale [MacKay, 2003, p.138]. X_c e $X_{c'}$ sono variabili casuali che modellano gli insiemi c e c', come processi binomiali, i.e., $X_c \sim \text{Bin}(\frac{|c|}{|\mathcal{M}|})$, e $X_{c'} \sim \text{Bin}(\frac{|c'|}{|\mathcal{M}|})$.

Una distanza simile a $d_H(c, c')$ è utilizzata in [Lancichinetti et al., 2009] per calcolare il grado di overlapping in grafi che rappresentano comunità gerarchiche, tuttavia la definizione utilizzata in questo progetto è una versione modificata dell'originale, i cui dettagli implementativi sono forniti in Sezione 3.2.4, mentre per una definizione completa e dettagliata di $d_H(c, c')$, si rimanda all'Appendice A.

Per il calcolo della distanza tra due cluster $d(c, c')$, tra tutte le possibili definizioni sono state scelte d_J e d_H in quanto, per il loro calcolo richiedono solamente la conoscenza di $|c|$, $|c'|$, $|c\bigcap c'|$ e $|\mathcal{M}|$, che sono anche le uniche informazioni disponibili. Le distanze convenzionali, utilizzate per calcolare la distanza tra due cluster, non sono invece applicabili in questo contesto. Il loro calcolo si basa sulla distanza tra i singoli elementi contenuti nei cluster (e.g., per entrambi i cluster è estratto il centroide e la distanza tra i due cluster

Algorithm 1 Calcolo di $D(\mathcal{C}^A, \mathcal{C}^B)$

```
1:  Δ_{a,b} ← 0
2:  Δ_{b,a} ← 0
3:  for all c_a ∈ C^A do
4:      δ_a ← 1
5:      for all c_b ∈ C^B do
6:          if d(c_a, c_b) < δ_a then
7:              δ_a ← d(c_a, c_b)
8:          end if
9:      end for
10:     Δ_{a,b} ← Δ_{a,b} + δ_a
11: end for
12: for all c_b ∈ C^B do
13:     δ_b ← 1
14:     for all c_a ∈ C^A do
15:         if d(c_b, c_a) < δ_b then
16:             δ_b ← d(c_b, c_a)
17:         end if
18:     end for
19:     Δ_{b,a} ← Δ_{b,a} + δ_b
20: end for
21: Δ_{a,b} ← Δ_{a,b} / |C^A|
22: Δ_{b,a} ← Δ_{b,a} / |C^B|
23: D(C^A, C^B) ← (Δ_{a,b} + Δ_{b,a}) / 2
```

è considerata uguale alla distanza tra i due centroidi). Gli elementi presenti all'interno dei cluster in analisi su cui si potrebbero applicare concetti di distanza sono due:

- **label**: non ha senso calcolare la distanza tra due cluster come la distanza (in qualunque sua definizione) tra le stringhe presenti al loro interno, in quanto, si andrebbe a ricercare le ben note inconsistenze di tipo sintattico e non quelle di tipo semantico-strutturale di cui ci si occupa in questa tesi.

- **codice hash**: calcolare una distanza tra due hash non ha nessun significato. I codici hash sono utilizzati come identificatori univoci ma, il loro valore non è in alcun modo dipendente dal label a cui sono associati.

Risulta quindi impossibile applicare le definizioni comuni di distanza per la natura dei cluster in esame, altre definizioni di distanza possono essere trovate in [Wolda, 1981].

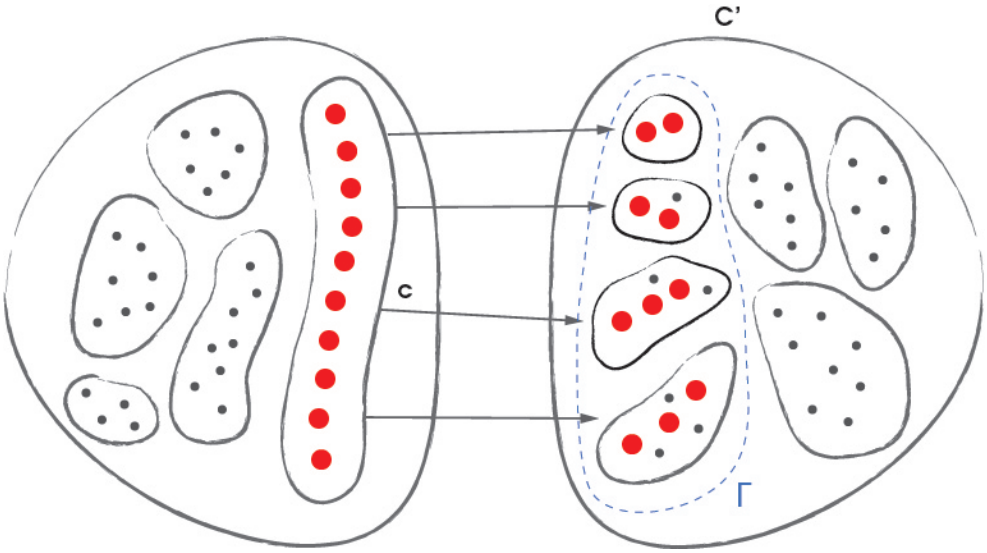


Figura 3.4: Esempio di ricerca di $\Gamma(c, \mathcal{C}')$, l'insieme di scatter è costituito da tutti i cluster di $\mathcal{C}'$ che hanno almeno un elemento in comune con c.

3.2.2 Scatter measure

In questa sezione è definita $S(\mathcal{C}^A, \mathcal{C}^B)$, *scatter measure* tra due sistemi di naming. Questo indicatore ha lo scopo di dare una misura media dello "sparpagliamento" dei cluster, ovvero visualizza il numero medio di cluster di $\mathcal{C}^B$ necessari a contenere tutti i malware contenuti in un cluster di $\mathcal{C}^A$ e vice-versa.

Ad alto livello, valore assunto da questo indicatore, può essere considerato il numero medio di label che il vendor B associa ad una label del vendor A e vice-versa.

Più formalmente la scattering measure è definita come:

$$S(\mathcal{C}^A, \mathcal{C}^B) := \frac{1}{2}\left(\frac{\sum_{c \in \mathcal{C}^A} |\Gamma(c, \mathcal{C}^B)|}{|\mathcal{C}^A|} + \frac{\sum_{c \in \mathcal{C}^B} |\Gamma(c, \mathcal{C}^A)|}{|\mathcal{C}^B}|\right) \tag{3.2}$$

dove $\Gamma(c, \mathcal{C}')$ è *l'insieme di scatter* di c rispetto a $\mathcal{C}'$ definito come:

$$\Gamma(c, \mathcal{C}') := \{c' \in \mathcal{C}' \mid c \cap c' \neq \varnothing\}. \tag{3.3}$$

In Figura 3.4 è riportato un esempio di $\Gamma(c, \mathcal{C}')$.

Generalizzazione della misura Nella definizione precedente $\Gamma(c, \mathcal{C}')$ è l'insieme di cluster appartenenti a $\mathcal{C}'$, tali che la loro unione contiene tutti i malware contenuti dal cluster $c \in \mathcal{C}$, ovvero $c \subseteq \bigcup_{c'_i \in \Gamma} c'_i$. La definizione può es-

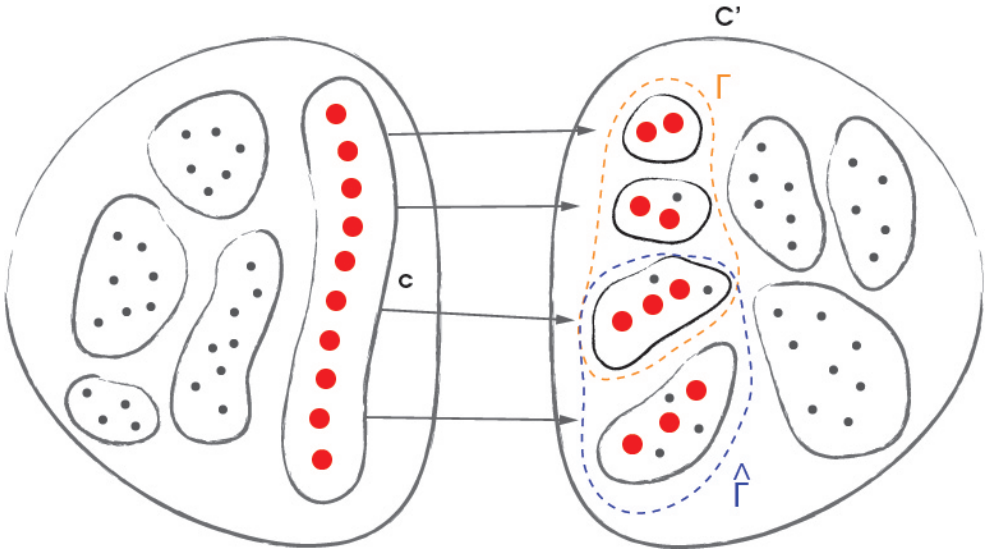


Figura 3.5: Esempio di $\widehat{\Gamma}(c, \mathcal{C}')$. Esistono numerosi insiemi che coprono c per una percentuale $T_\sigma = 60\%$, ma tra tutti i possibili raggruppamenti $\widehat{\Gamma}$ è quello di cardinalità minima.

sere generalizzata, ricercando un insieme di cluster la cui unione contenga almeno una percentuale T_σ dei malware contenuti in c. Più formalmente:

$$\sigma(\Gamma) := \frac{\left| c \cap \left(\bigcup_{c' \in \Gamma(c, \mathcal{C}')} c' \right) \right|}{|c|} \%, \tag{3.4}$$

definisce la percentuale di malware che c (e.g., cluster appartenente a $\mathcal{C}^A$), condivide con l'unione dei cluster appartenei all'insieme di scatter derivato da $\mathcal{C}'$ (e.g., insieme dei cluster del vendor B). Fissato un valore di trashold $T_\sigma \in [0, 100]$ è possibile che esistano più insiemi di scatter che soddisfano la condizione $\sigma(\Gamma) \geq T_\sigma$, tra tutti questi possibili candidati è scelto l'insieme Γ di cardinalità minima (gli elementi di Γ sono cluster, non i malware in essi contenuti). Più formalmente si ricerca $\widehat{\Gamma}$ tale che:

$$\widehat{\Gamma}_{T_\sigma} : \nexists \Gamma(c, \mathcal{C}') \qquad \sigma(\Gamma) \geq T_\sigma \ \wedge \ |\Gamma| < |\widehat{\Gamma}| \tag{3.5}$$

che è il minimo *insieme di scatter* che copre c per almeno una percentuale T_σ, come visualizzato in Figura 3.5 ed esemplificato dall'Algoritmo 2.

In una situazione ideale di completa consistenza si avrà $S(\cdot, \cdot, T_\sigma) = 1$ per ogni T_σ, in quanto ogni label generato dal vendor A sarà associato esattamente ad un solo label generato dal vendor B. In una situazione reale invece, si avrà $S(\cdot, \cdot, T_\sigma) \geq 1$ e un andamento a tratti costante e a tratti crescente.

Algorithm 2 Calcolo di $\widehat{\Gamma}_{T_\sigma(c,\mathcal{C}')}$

```
1: Γ̂ ← ∅
2: σ ← 0
3: while σ < T_σ do
4:     γ ← ∅
5:     for all c' ∈ C' do
6:         if |γ ⋂ c| < |c' ⋂ c| then
7:             γ ← c'
8:         end if
9:     end for
10:    C' ← C' − γ
11:    Γ̂ ← Γ̂ ⋃ γ
12:    σ ← σ + |γ ⋂ c| * 100 / |c|
13: end while
```

3.2.3 Estrazione inconsistenze

In questa sezione è formalizzato un metodo che consente l'estrazione delle inconsistenze presenti tra due sistemi di naming, il cui concetto è stato anticipato in Sezione 3.

Per effettuare l'estrazione delle suddette inconsistenze è necessario modellare le relazioni presenti tra i cluster ottenuti in **Fase 1**. Va inoltre precisato che gli insiemi di cluster in analisi (i.e., $\mathcal{C}^A$, $\mathcal{C}^B$) devono essere partizioni di $\mathcal{M}$. La modellazione delle relazioni avviene attraverso la generazione di un grafo non orientato, i cui nodi rappresentano i cluster in analisi ed i cui archi codificano la presenza di una relazione tra i due cluster.

Grafo dei cluster Dati $\mathcal{C}^A$ a $\mathcal{C}^B$ il grafo $\mathcal{G}^{AB} = \langle \mathcal{V}^{AB}, \mathcal{E}^{AB} \rangle$ è definito come $\mathcal{V} = \mathcal{C}^A \cup \mathcal{C}^B$ e

$$\mathcal{E}^{AB} = \{(c, c') \mid c \in \mathcal{C}^A, c' \in \mathcal{C}^B \wedge c \cap c' \neq \varnothing\}. \tag{3.6}$$

In altre parole, dato un cluster c contenente malware etichettati da A, ed un cluster c' contenente malware etichettati da B, l'arco di connessione tra c e c' è creato solamente se c ha almeno un elemento in comune con c' (nel paragrafo successivo è fornita una generalizzazione che tiene conto del numero di campioni in comune).

Ad un livello più alto di astrazione $\mathcal{G}^{AB}$ rappresenta le relazioni tra label generati dai vendor A e B per l'insieme di malware $\mathcal{M}$.

A questo punto, il problema della ricerca delle (in)consistenze, può essere ricondotto all'estrazione delle componenti connesse del grafo $\mathcal{G}^{AB}$, problema per cui è sfruttabile uno degli efficienti algoritmi (e.g., [Tarjan, 1972]) implementati nelle librerie dei più comuni linguaggi di programmazione.

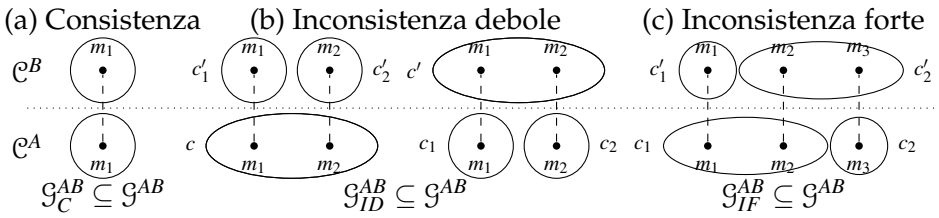


Figura 3.6: Esempi di consistenza $\mathcal{G}^{AB}_C$, inconsistenza debole $\mathcal{G}^{AB}_{ID}$ ed inconsistenza forte $\mathcal{G}^{AB}_{IF}$, i.e., componenti connesse del grafo $\mathcal{G}^{AB}$ generato dal confronto tra il vendor A ed il vendor B. Ogni linea verticale rappresenta la condivisione di un malware. Ogni insieme può contenere un numero maggiore di malware: le connessioni presenti identificano solo un caso base.

Una volta estratte le componenti connesse è necessario analizzarle, e stabilirne la natura:

Consistenza (C) La componetene connessa è costituita da due cluster che contengono esattamente gli stessi campioni. É importante notare che la consistenza persiste anche se i campioni presenti nel cluster di A possiedono label differenti dai campioni contenuti nel cluster di B. Un esempio è visualizzato in Figura 3.6a.

Inconsistenza debole (ID) La componetene connessa contiene solamente un cluster $c \in \mathcal{V}^A = \mathcal{C}^A$ e tutti i gli altri cluster appartenerti alla componetene connessa $c' \in \mathcal{V}^B = \mathcal{C}^B$ ne costituiscono un sottoinsieme $c' \subset c$ (o vice-versa). In questo caso il vendor B adotta un sistema di naming che fornisce una grana più fine di informazione rispetto a quello adottato dal vendor A, però $\mathcal{C}^A$ e $\mathcal{C}^B$ non posso essere considerati consistenti, in quanto differiscono sulla quantità di informazione codificata nei loro label. Un esempio è visualizzato in Figura 3.6b.

Inconsistenza forte (IF) La componetene connessa contiene più di un cluster per ogni vendor. Dato che i cluster sono partizioni dell'insieme di malware $\mathcal{M}$, è necessaria la presenza di almeno quattro cluster nella componenete connessa: $c_1, c_2 \in \mathcal{V}^A = \mathcal{C}^A$, $c'_1, c'_2 \in \mathcal{V}^B = \mathcal{C}^B$ tali che la seguente condizione sia verificata:

$$c_1 \bigcap c'_1 \neq \varnothing \;\; \wedge \;\; c_2 \bigcap c'_2 \neq \varnothing \;\; \wedge \;\; c'_2 \cap c_1 \neq \varnothing$$

L'inconsistenza include tutti i cluster che fanno parte della componetene connessa. In altre parole i cluster condividono alcuni campioni senza essere sottoinsiemi gli uni degli altri. Questo avviene per sistemi di naming tra loro differenti ed inconsistenti, un esempio è visualizzato in Figura 3.6c.

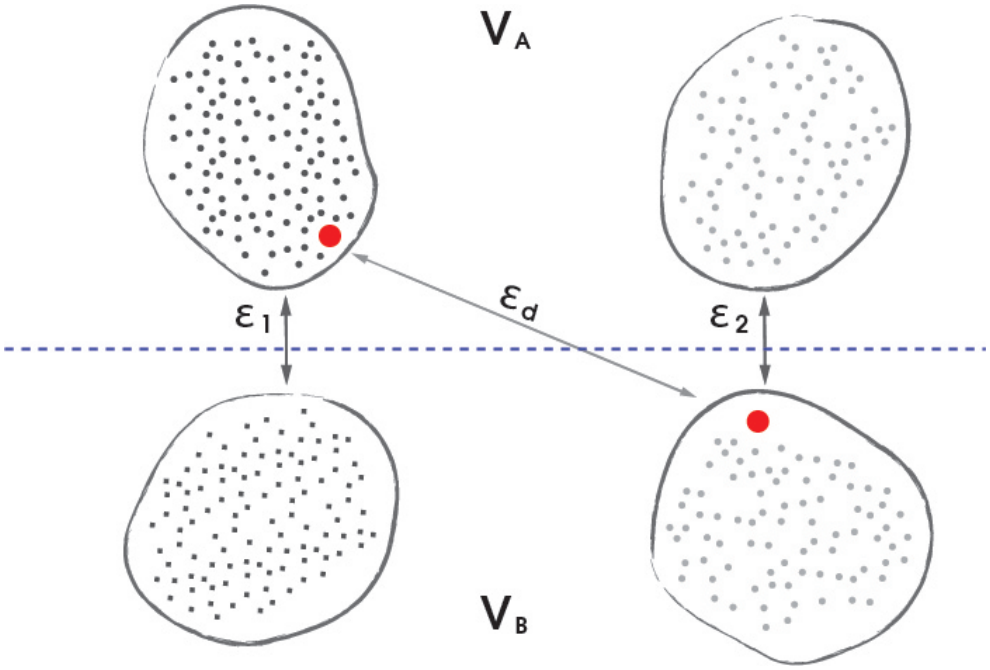


Figura 3.7: Esempio di relazione poco significativa. La relazione modellata dall'arco ε_d è poco significativa rispetto a quelle modellate dagli archi ε_1 ed ε_2, in quanto è causata dalla condivisione di un solo elemento. ε_d può essere considerato un effetto di bordo e la sua eliminazione trasformerebbe l'inconsistenza di tipo forte in due più sensate consistenze.

Estensione del modello Le relazioni tra cluster, modellate dagli archi del grafo $\mathcal{G}^{AB}$, possono essere caratterizzate, in base al numero di elementi che i due cluster condividono, da differenti livelli di significatività. Per tenere traccia di tale paramento il modello presentato nel paragrafo precedente è stato esteso, introducendo una funzione di peso degli archi definita come segue:

$$W(c,c') := \max\left\{\frac{|c \cap c'|}{|c|}\%, \frac{|c \cap c'|}{|c'|}\%\right\}, \forall (c,c') \in \mathcal{E}^{AB} \tag{3.7}$$

In questo nuovo modello ogni arco rappresenta il grado massimo di sovrapposizione tra i due cluster, la funzione proposta fornisce esattamente l'attuale frazione di c, cluster di A, condivisa con c', cluster di B, indipendentemente dalla dimensione di c' che potrebbe essere sproporzionatamente grande rispetto a quella di c (o vice-versa). Fissando un valore di trashold $T_w \in [0,100]$, il grafo può essere preprocessato rimuovendo gli archi aventi peso $W(c,c') \leq T_w$, eliminando così dal modello le relazioni non significative generate da campioni spuri, come esemplificato in Figura 3.7.

3.2.4 Dettagli implementativi

Questa sezione si suddivide in due sottosezioni, la prima riguarda l'analisi dei differenti tipi di normalizzazione per la distanza $D(\mathcal{C}^A, \mathcal{C}^A)$. La seconda definisce in maniera dettagliata la formula della distanza $d_H(c, c')$ utilizzata nell'implementazione del metodo, e motiva sperimentalmente la modifica applicata alla definizione originale.

Fattori di normalizzazione alternativi

Lo scopo di questa sottosezione è quello di confrontare, attraverso dati sperimentali, i possibili fattori di normalizzazione per la distanza $D(\mathcal{C}^A, \mathcal{C}^B)$.

$$D(\mathcal{C}^A, \mathcal{C}^B) = \sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B) + \sum_{c \in \mathcal{C}^B} \delta(c, \mathcal{C}^A) \tag{3.8}$$

La funzione (3.8) può essere normalizzata tramite un unico fattore di normalizzazione (i.e., (3.9) e (3.11)) o come media delle due distanze parziali normalizzate (i.e., (3.10) e (3.12)). Il fattore normalizzante può essere inoltre definito o in base al numero di cluster in analisi (i.e., (3.9) e (3.10)) o in base al numero di elementi contenuti nei cluster (3.11) e (3.12)).

$$D_1(\mathcal{C}^A, \mathcal{C}^B) = \frac{\sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B) + \sum_{c \in \mathcal{C}^B} \delta(c, \mathcal{C}^A)}{|\mathcal{C}^A| + |\mathcal{C}^B|} \tag{3.9}$$

$$D_2(\mathcal{C}^A, \mathcal{C}^B) = \frac{1}{2} \left(\frac{\sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B)}{|\mathcal{C}^A|} + \frac{\sum_{c \in \mathcal{C}^B} \delta(c, \mathcal{C}^A)}{|\mathcal{C}^B|} \right) \tag{3.10}$$

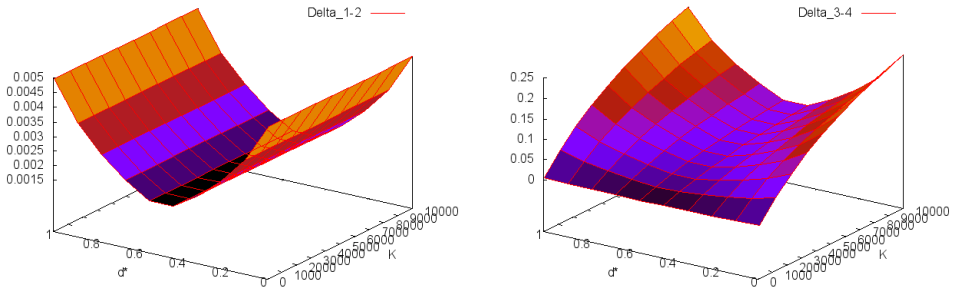
$$D_3(\mathcal{C}^A, \mathcal{C}^B) = \frac{\sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B)|c| + \sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B)|c|}{\sum_{c \in \mathcal{C}^B} |c| + \sum_{c \in \mathcal{C}^A} |c|} \tag{3.11}$$

$$D_4(\mathcal{C}^A, \mathcal{C}^B) = \frac{1}{2} \left(\frac{\sum_{c \in \mathcal{C}^A} \delta(c, \mathcal{C}^B)|c|}{\sum_{c \in \mathcal{C}^A} |c|} + \frac{\sum_{c \in \mathcal{C}^B} \delta(c, \mathcal{C}^A)|c|}{\sum_{c \in \mathcal{C}^B} |c|} \right) \tag{3.12}$$

La sottosezione si suddivide in due casi di studio: il **Caso 1** analizza la stabilità della normalizzazione per elementi rispetto alla normalizzazione per cluster, mentre il **Caso 2** mette a confronto le distanze D_1 e D_2.

Caso 1 In questo paragrafo si dimostra la maggiore stabilità della normalizzazione per cluster rispetto alla normalizzazione per elementi.

Come caso di studio si considera un modello costituito da due insiemi di cluster $\mathcal{C}^A$ e $\mathcal{C}^B$, costituiti entrambi da 100 cluster, che a loro volta contengono



(a) Δ_{1-2} (b) Δ_{3-4}

Figura 3.8: Sensibilità di D_{1-2} e D_{3-4} ai disturbi. D_{1-2} non risente della dimensione del cluster di disturbo quindi risulta molto meno sensibile ai disturbi di D_{3-4}.

100 elementi. Inoltre, per ogni cluster appartenete al modello esiste un cluster della controparte tale che $\delta(c, \mathcal{C}^A) = \delta(c, \mathcal{C}^B) = 0.5$. Per la simmetria dovuta alla costruzione si ha che $D_1(\mathcal{C}^A, \mathcal{C}^B) = D_2(\mathcal{C}^A, \mathcal{C}^B)$ e che $D_3(\mathcal{C}^A, \mathcal{C}^B) = D_4(\mathcal{C}^A, \mathcal{C}^B)$. D'ora in avanti ci si riferirà quindi con D_{1-2} al valore fornito dalle distanze normalizzate per cluster (i.e., (3.9) (3.10)), e con D_{3-4} al valore fornito dalle distanze normalizzate per elementi (i.e., (3.11) (3.12)).

Il modello originale è perturbato tramite l'inserimento di due cluster a_p e b_p di dimensioni $|a_p| = |b_p| = K$, con $K \in [1, 100]$, che contribuiscono al calcolo della distanza tra $\mathcal{C}_p^A$ e $\mathcal{C}_p^B$ (modelli perturbati) tramite la somma di $\delta(a_p, \mathcal{C}^B) = \delta(b_p, \mathcal{C}^A) = d^*$, con $d^* \in [0, 1]$.

In Figura 3.8, al variare di K e d^*, sono riportate le differenze tra le distanze calcolate sui modelli originali e le distanze calcolate sui modelli perturbati[1]. Più precisamente in Figura 3.8a è riportata $\Delta_{1-2} = |D_{1-2}(\mathcal{C}^A, \mathcal{C}^B) - D_{1-2}(\mathcal{C}_p^A, \mathcal{C}_p^B)|$ che visualizza la risposta alle perturbazioni della normalizzazione per cluster, mentre in Figura 3.8b è riportata $\Delta_{3-4} = |D_{3-4}(\mathcal{C}^A, \mathcal{C}^B) - D_{3-4}(\mathcal{C}_p^A, \mathcal{C}_p^B)|$, che visualizza la risposta alle perturbazioni della normalizzazione per elementi.

Si nota che Δ_{1-2} assume valori molto piccoli rispetto a Δ_{3-4} (circa due ordini di grandezza inferiori) e che, come ovvio aspettarsi, Δ_{1-2} dipende solamente dal valore assunto da d^*[2], mentre Δ_{3-4} dipende anche dal valore assunto da K. Si nota inoltre che Δ_{3-4} tende a rimanere stabile, o comunque ad assumere valori molto contenuti, solamente se i cluster di perturbazione sono molto piccoli rispetto alla dimensione del dataset, mentre se la dimensione del cluster diventa significativa Δ_{3-4} assume valori notevoli. In

[1] La simmetria garantita dalla costruzione fa si che anche $D_1(\mathcal{C}_p^A, \mathcal{C}_p^B) = D_2(\mathcal{C}_p^A, \mathcal{C}_p^B)$ e che $D_3(\mathcal{C}_p^A, \mathcal{C}_p^B) = D_4(\mathcal{C}_p^A, \mathcal{C}_p^B)$

[2] Per $d^* = 0.5$ $\Delta_{1-2} = 0$ e $\Delta_{3-4} = 0$ perché per costruzione $D_{1-2}(\mathcal{C}^A, \mathcal{C}^B) = D_{3-4}(\mathcal{C}^A, \mathcal{C}^B) = 0.5$

altre parole D_{3-4} è estremamente sensibile a perturbazioni dovute a cluster di grosse dimensioni. Questa proprietà non è desiderabile, perchè, possibili errori nel modello potrebbero portare alla generazione di cluster erronei di grandi dimensioni, che influenzerebbero pesantemente il valore di distanza calcolato. La normalizzazione per numero di cluster invece non è affetta da questa caratteristica negativa, e possiede invece la desiderabile proprietà di ammortizzare l'errore introdotto da eventuali cluster erronei in maniera proporzionale alle dimensioni di $|\mathcal{C}^A|$ e $|\mathcal{C}^B|$, che per il caso in analisi raggiungono valori dell'ordine delle migliaia. Alla luce delle precedenti considerazione è abbandonata la normalizzazione per elementi a favore di quella per cluster.

Caso 2 In questo paragrafo si mettono a confronto D_1 e D_2, cercando di stabilire quale delle due è meno sensibile alle eventuali perturbazioni introdotte da un insieme di cluster erronei.

Come caso di studio si considera un modello costituito da due insiemi di cluster $\mathcal{C}^A$ e $\mathcal{C}^B$, costituiti entrambi da N cluster, con $N \in [1, 1000]$. Inoltre per ogni cluster appartenete al modello esiste un cluster della controparte tale che $\delta(c, \mathcal{C}^A) = \delta(c, \mathcal{C}^B) = 0.5$.

Il modello originale è perturbato con l'inserimento di N_a cluster, con $N_a \in [1, N]$, che contribuiscono al calcolo della distanza con $\delta(c_p, \mathcal{C}^B_p) = \delta(c_p, \mathcal{C}^A_p) = 1$. Gli N_a cluster che perturbano il modello originale sono inseriti per una percentuale $K\%$ in $\mathcal{C}^A$, trasformandolo in $\mathcal{C}^A_p$, e per una percentuale $100 - K\%$ in $\mathcal{C}^B$, trasformandolo in $\mathcal{C}^B_p$.

In Figura 3.9 sono riportate, al variare di N, N_a e K le differenze tra le distanze dei modelli originali e quelle dei modelli perturbati. Più formalmente sono riportate $\Delta_1 = |D_1(\mathcal{C}^A, \mathcal{C}^B) - D_1(\mathcal{C}^A_p, \mathcal{C}^B_p)|$ e $\Delta_2 = |D_2(\mathcal{C}^A, \mathcal{C}^B) - D_2(\mathcal{C}^A_p, \mathcal{C}^B_p)|$.

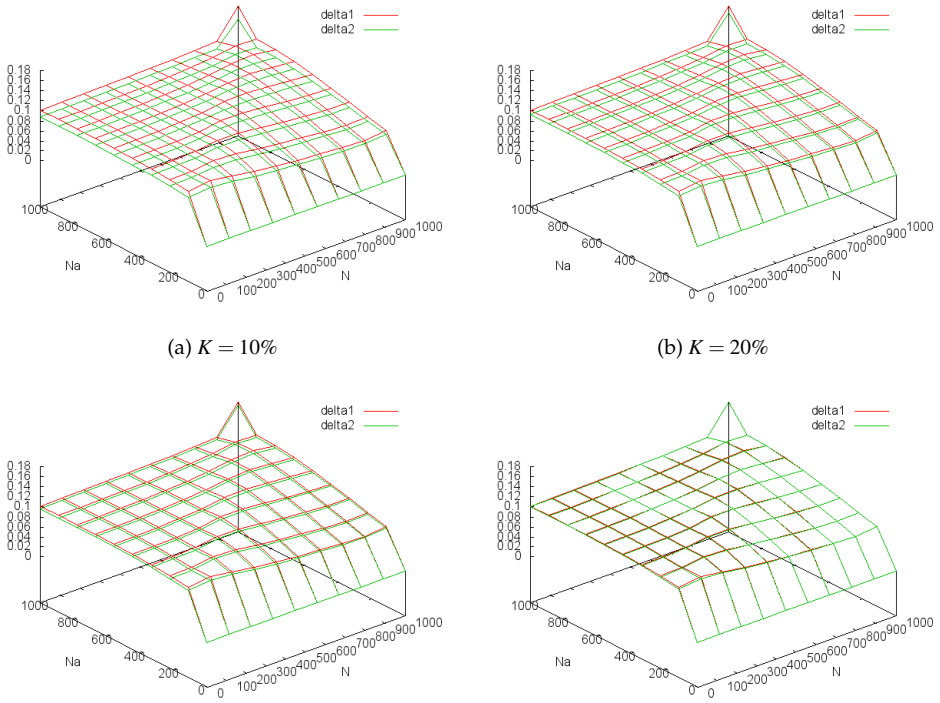
Si nota che D_1 e D_2 reagiscono in maniera simile alla presenza di cluster di perturbazione. D_2, nei casi in cui le perturbazioni non sono equamente distribuite tra i due insiemi $\mathcal{C}^A$ e $\mathcal{C}^B$ (Figura 3.9a, Figura 3.9b), tende a fornire risultati leggermente migliori, mentre nel caso in cui le perturbazioni sono completamente distribuite non vi è differenza[3].

Il fatto che D_2 fornisca una risposta leggermente migliore rispetto ad D_1 e che permetta di dare un interpretazione più comprensibile alla formula della distanza (la distanza tra $\mathcal{C}^A$ e $\mathcal{C}^B$ è definita come la media della distanza da $\mathcal{C}^A$ a $\mathcal{C}^B$ e vice-versa), ha portato a scegliere D_2 come formula definitiva per il calcolo della distanza.

Distanza tramite entropia

In questa sottosezione si vuole illustrare il procedimento che ha portato alla modifica della distanza $d_H(c, c')$ dalla sua formulazione originale (3.13), alla

[3]Non è sempre stato possibile separare gli N_a cluster perfettamente in due ed inserirli per metà in $\mathcal{C}^A$ e $\mathcal{C}^B$, nei casi in cui N_a non è multiplo di 2 risulta impossibile, quindi è stata fatta un approssimazione, senza la quale si avrebbe $\Delta_2 = \Delta_1$ per $K = 50\%$



(a) $K = 10\%$ (b) $K = 20\%$

(c) $K = 30\%$ (d) $K = 50\%$

Figura 3.9: Sensibilità di D_1 e D_2 ai disturbi. La reazione all'inserimento di cluster di perturbazione tende ad essere simile, anche se D_2 tende a fornire risultati leggermente migliori.

formulazione utilizzata nella fase sperimentale (3.15).

$$d_H(c,c') = \begin{cases} \frac{H(X_c|X_{c'})}{H(X_c)} & \text{se } h(P(1,1)) + h(P(0,0)) > h(P(1,0)) + h(P(0,1)) \\ 1 & \text{altrimenti} \end{cases} \tag{3.13}$$

Dove con $P(\alpha, \beta)$ si indica la probabilità che le variabili aleatorie X_c e $X_{c'}$ assumano rispettivamente i valori α e β, e con $h(p)$ si indica l'entropia $-pLog(p)$. Per maggiori informazioni e dettagli su (3.13) si rimanda all'Appendice A.

D'ora in avanti la definizione fornita in (3.13) verrà denominata $d_{H_1}(c,c')$, mentre verrà denominata $d_{H_2}(c,c')$ la seguente definizione, che non tiene conto della condizione applicata in (3.13).

$$d_{H_2}(c,c') = \frac{H(X_c \mid X_{c'})}{H(X_c)} \tag{3.14}$$

La condizione posta nella formulazione originale della distanza (3.13),

$\|\mathbf{c}\|$	$\|\mathbf{c}'\|$	$\|\mathbf{c}\bigcap\mathbf{c}'\|$	$\mathbf{d_{H_1}}$	$\mathbf{d_{H_2}}$
4	4	2	1	0.5935
100	100	55	1	0.5826
50000	48000	100	1	0.0764

Tabella 3.1: Esempi di distanze

penalizza le coppie di cluster che hanno un intersezione non negabile ma, le cui dimensioni sono piccole rispetto alla dimensione del dataset $\mathcal{M}$.

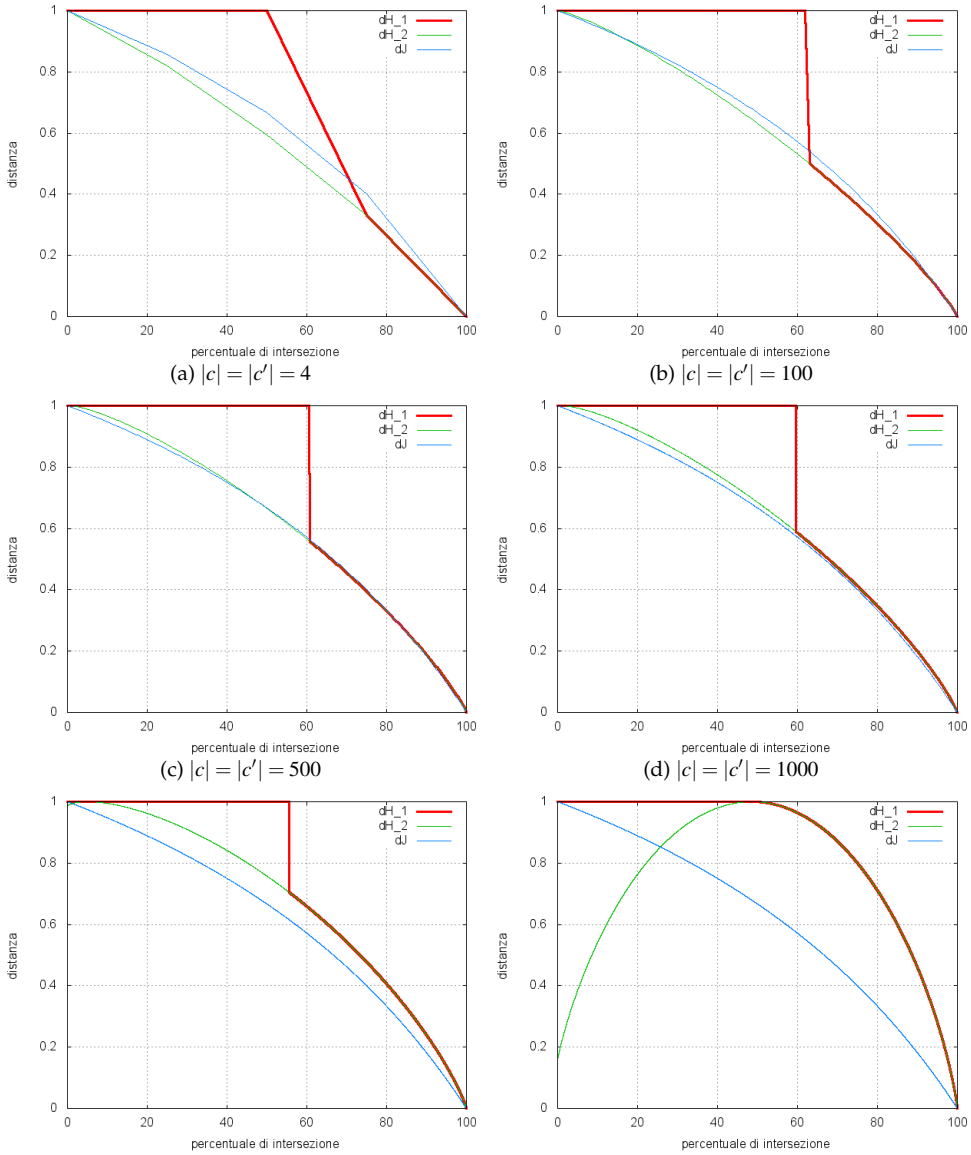
In Tabella 3.1 sono riportati degli esempi di calcolo di $d_{H_1}(c,c')$ e $d_{H_2}(c,c')$, dove c, c' sono cluster di dimensioni variabili, generati a partire da un dataset di $|\mathcal{M}| = 98798$ elementi[4]. Osservando i valori riportati in tabella, per i primi due casi il valore fornito da $d_{H_2}(c,c')$ è preferibile in quanto l'intersezione non è nulla, mentre per il terzo caso è preferibile utilizzare il valore ricavato tramite $d_{H_1}(c,c')$.

Risulta quindi necessario trovare un parametro che permetta di decidere in maniera automatica se è preferibile utilizzare $d_{H_1}(c,c')$ o $d_{H_2}(c,c')$. In Figura 3.10, per differenti dimensioni di c e c', sono plottati gli andamenti di $d_{H_1}(c,c')$, $d_{H_2}(c,c')$ e $d_J(c,c')$ al variare della percentuale di intersezione $|c\bigcap c'|$. Dall'immagine si nota che per cluster di piccole dimensioni d_{H_2} è molto vicina a d_J, mentre d_{H_1}, per percentuali di intersezione inferiori al 60%, tende a fornire valori molto pessimistici (i.e., tipicamente $d_{H_1}(c,c') = 1$ per $|c\bigcap c'| < 60\%$), per poi coincidere con d_{H_2} per percentuali superiori. Per cluster di grosse dimensioni invece, (Figura 3.10f) d_{H_2} tende a dare valori alquanto bizzarri, molto lontani da quelli forniti da d_J, che risultano più simili a quelli assunti da d_{H_1}.

A partire da queste osservazioni si può pensare di utilizzare il valore assunto da d_J per decidere se utilizzare il valore fornito da d_{H_1}, o quello fornito da d_{H_2} secondo la seguente euristica: *se la condizione posta nella definizione originale della distanza (3.13) è soddisfatta allora si ritiene d_{H_1} attendibile, se non è soddisfatta si utilizza il valore di d_J come riferimento: se $|d_J - d_{H_2}| \leq T_H$ allora si considera d_{H_1} troppo penalizzate ed è preferibile utilizzare il valore fornito da d_{H_2}, se invece, $|d_J - d_{H_2}| > T_h$ allora si considera d_{H_2} affetta dai problemi relativi alla formulazione della distanza tramite entropia, e si opta per d_{H_1}.* Più formalmente si definisce d_{H_3} come segue:

$$d_H(c,c') = \begin{cases} \frac{H(X_c|X_{c'})}{H(X_c)} & h(P(1,1)) + h(P(0,0)) > h(P(1,0)) + h(P(0,1)) \quad \text{(a)} \\ & \vee\ |\frac{H(X_c|X_{c'})}{H(X_c)} - J(c,c')| \leq T_H \quad \text{(b)} \\ 1 & \text{altrimenti} \end{cases} \tag{3.15}$$

[4] 98798 è la dimensione del dataset utilizzato per l'analisi sperimentale



(a) $|c| = |c'| = 4$ (b) $|c| = |c'| = 100$

(c) $|c| = |c'| = 500$ (d) $|c| = |c'| = 1000$

(e) $|c| = |c'| = 5000$ (f) $|c| = |c'| = 48000$

Figura 3.10: Confronto tra $d_{H_1}(c,c')$, $d_{H_2}(c,c')$, $d_J(c,c')$ al variare dell'intersezione $|c \bigcap c'|$ per differenti dimensioni di $|c|$, $|c'|$ su un dataset di $|\mathcal{M}| = 98798$ elementi. Si nota che per cluster di piccole dimensioni d_{H_2} è molto vicina a d_J, mentre per cluster di grandi dimensioni, con basse percentuali di intersezione, d_{H_2} tende ad allontanarsi da d_J e a fornire valori bizzarri.

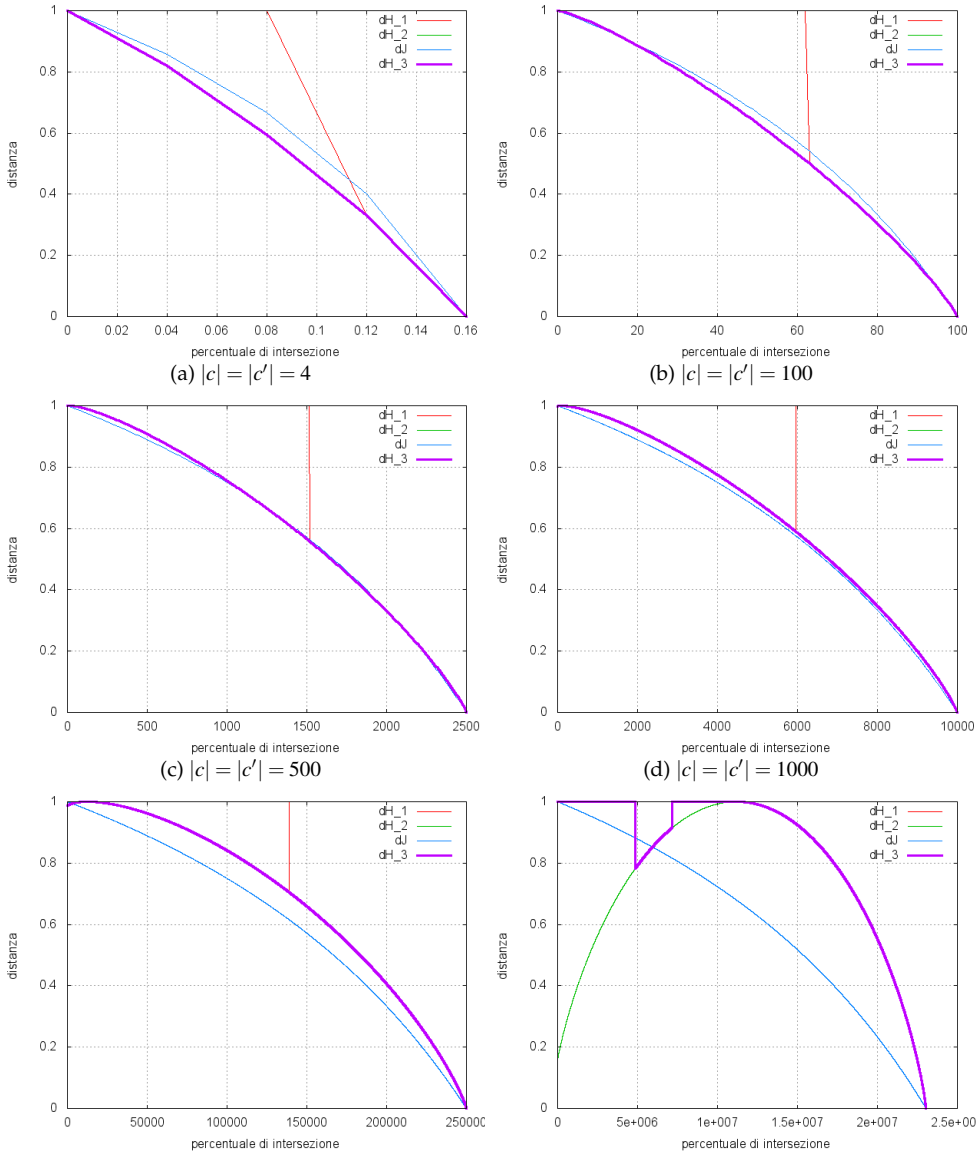
	$\mathbf{T_h = 0.10}$	$\mathbf{T_h = 0.20}$	$\mathbf{T_h = 0.30}$
$\mathbf{d_{H_1}}$	43.06%	66.76%	83.18%
$\mathbf{d_{H_2}}$	81.33%	92.07%	96.59%

Tabella 3.2: Percentuale di confronti per cui $|d_{H_2} - d_J| \leq T_h$ e $|d_{H_1} - d_J| \leq T_h$

Il valore del parametro T_H, è stato ricavato da simulazioni sperimentali. Dati due cluster c, c', appartenenti ad un dataset contenente $|\mathcal{M}| = 98798$ elementi, aventi una dimensione variabile tra $(0, 100]$ elementi ed un'intersezione variabile tra lo $[0, 100]\%$ sono state calcolate, per tutte le possibili combinazioni, le differenze $|d_{H_1}(c, c') - d_J(c, c')|$ e $|d_{H_2}(c, c') - d_J(c, c')|$. I risultati sintetici di tale simulazione sono riportati in Tabella 3.2.

Si nota per l'81% dei casi vede $|d_{H_2}(c, c') - d_J(c, c')| \leq 0.1$, si può quindi ipotizzare che, per cluster di piccole dimensioni, d_{H_2} tende ad essere molto vicina ad d_J (e.g., $|d_{H_2}(c, c') - d_J(c, c')| \leq 0.1$). Di conseguenza se d_{H_2} e d_J si discostano più dello 0.1 allora d_{H_2} non è più da considerarsi attendibile per il calcolo della distanza ed è preferibile l'utilizzo di d_{H_1}.

In Figura 3.11 viene aggiunto l'andamento di d_{H_3} ai precedenti andamenti visualizzati in Figura 3.10, come ci si aspetta dalle considerazioni precedenti, d_{H_3} tende a sovrapporsi a d_{H_2} per cluster di piccole dimensioni, mentre per cluster di grandi dimensioni (e.g., Figura 3.11f) tende a sovrapporsi meglio a d_{H_1}.



(a) $|c| = |c'| = 4$ (b) $|c| = |c'| = 100$

(c) $|c| = |c'| = 500$ (d) $|c| = |c'| = 1000$

(e) $|c| = |c'| = 5000$ (f) $|c| = |c'| = 48000$

Figura 3.11: Confronto tra $d_{H_1}(c,c')$, $d_{H_2}(c,c')$, $d_{H_3}(c,c')$ e $d_J(c,c')$ al variare dell'intersezione $|c \bigcap c'|$ per differenti dimensioni di $|c|$, $|c'|$ su un dataset di $|\mathcal{M}| = 100.000$ elementi. Si nota che d_{H_3} tende a sovrapporsi a d_{H_2} per cluster di piccole dimensioni, mentre per cluster di grandi dimensioni tende a sovrapporsi meglio a d_{H_1}.

Capitolo 4

Risultati sperimentali

In questo capitolo sono presentati e discussi i risultati sperimentali ricavati dall'applicazione del processo di analisi, descritto nel Capitolo 3, su un dataset costituito da circa 100.000 campioni.A tali campioni sono associati i label di quattro differenti antivirus, di cui non sarà fornito il nome, e che saranno denominati V_1, V_2, V_3 e V_4. La scelta di non fornire il nome deriva dal fatto che lo scopo della sperimentazione è di dimostrare la presenza di inconsistenze strutturali e non quello di fare una classifica tra gli antivirus.

La prima parte del capitolo si sofferma brevemente sulla costruzione del dataset e sulle sue caratteristiche (Sezione 4.1), per poi passare alla presentazione grafica della struttura dei modelli dei sistemi di naming (Sezione 4.2). Successivamente si discutono le distanze tra i modelli (Sezione 4.3), la scatter measure (Sezione 4.5) ed infine sono presentate le inconsistenze strutturali (Sezione 4.5). Le analisi sono ripetute per differenti tagli degli alberi che costituiscono il modello, con lo scopo di verificare se l'utilizzo parziale dell'informazione contenuta nei label tende a rendere i sistemi più consistenti.

4.1 Generazione del dataset

Per la generazione del dataset ci si è appoggiati al portale VirusTotal. Tale portale mette a disposizione degli utenti, tramite interfaccia Web o API, una serie di servizi relativi alla scansione di file e URL. I servizi utilizzati all'interno di questo progetto sono fondamentalmente due:

Analisi di un file il portale permette, tramite diverse interfacce, di effettuare l'upload di un file, dopo di che ne effettua la scansione con una batteria di antivirus e fornisce, sempre tramite diverse interfacce, i report generati da tali antivirus.

Analisi di un hash il portale permette l'analisi di codici hash (i.e., MD5, SHA256). Se un file avente lo stesso codice hash è già stato analizzato in passato, il portale restituisce i report relativi alla scansione di tale file.

Oltre a questi due semplici servizi VirusTotal ne mette a disposizione svariati altri, che per questa tesi risultano però di poco interesse, mentre risulta fondamentale la possibilità di effettuare delle query sullo storico delle scansioni richieste al portale. In particolare è possibile richiedere i codici hash dei file che sono stati sottoposti a VirusTotal, specificandone importanti proprietà e caratteristiche, come il fatto che siano stati rilevati come malevoli da almeno un antivirus. Inoltre è possibile ordinare tele risultato in base a numerosi attributi, tra cui il numero di sottomissioni effettuate per singolo ogni file [VirusTotal, 2011].

Il dataset è stato generato selezionando i primi 100.000 codici hash forniti da VirusTotal come risultato della query "*file ritenuti malevoli da almeno un antivirus ordinati per numero di sottomissioni*". Il numero di sottomissioni di un file al portale, può essere considerato un indice d'interesse della comunità e della popolarità del file. I 100.000 hash ricavati, sono stati poi nuovamente sottoposti a VirusTotal, in modo da poter associare ad ogni singolo hash i report forniti dagli antivirus.

Analizzando manualmente i report, tra tutti gli antivirus, sono stati scelti V_1, V_2, V_3 e V_4 come campione rappresentativo per l'analisi. La scelta è stata pilotata della buona struttura dei loro label, cosa che ha permesso di ridurre notevolmente l'intervento manuale in **Fase 1**. Dato che VirusTotal non garantisce che ad un codice hash venga associata la totalità dei report forniti dagli antivirus che costituiscono il servizio, è stato necessario scartare alcuni campioni, ovvero quelli a cui non erano associati tutti e quattro i label degli antivirus presi in esame, riducendo così il numero di elementi da 100.000 a 98.798.

4.2 Visualizzazione strutture di naming

In questa sezione sono prima presentate le grammatiche utilizzate dagli antivirus selezionati per la generazione dei label, successivamente sono visualizzati graficamente i modelli dei sistemi di naming a differenti livelli di zoom, in modo da poter associare le differenze grafiche alle differenze grammaticali.

Di seguito sono proposte le grammatiche ricavate manualmente dall'osservazione di molti label.

V_1 `<type>:<platform>/<family>[.gen[!<version1>]|<version2>]`

V_2 `<type>.<platform>/<family>[.<version1>[.<version2>]]`

V_3 `<type>/<platform>.<family>[.gen]`

V_4 `<platform>/<family>[-gen|-<version>]`

- La presenza di classi di pattern (e.g., `<type>`) indica la presenza di un pattern appartenente a quella classe in tale posizione.

- Le parentesi quadre racchiudono un lista di possibili alternative separate dal carattere " | ". Tra tutte le alternative presenti vi è anche l'alternativa " stringa vuota", che è stata omessa per alleggerire la notazione della grammatica.

- Le sotto-stringhe ed i caratteri non racchiusi tra parentesi angolate (i.e. gen) si presentano direttamente all'interno dei label così come sono.

Le grammatiche sono molto ricche e tra loro molto differenti: V_1 e V_2 forniscono molta più informazione sulla versione dei malware rispetto a V_3 e V_4. V_4 inoltre, a differenza degli altri tre vendor, non fornisce alcuna informazione sul tipo di threat. Analizzando manualmente questa casistica, è stato scoperto che la classe `<family>` molto raramente è utilizzata anche per contenere informazioni relative al tipo (e.g., "Malware", "Dropper", "Trojan"). Dato che il fenomeno è veramente circoscritto, è ragionevole considerarlo parte della semantica della convenzione di naming. Per questa regione, e solamente per V_4, si può considerare il tipo di threat e la famiglia del malware allo stesso livello di importanza. É da notare che gli altri antivirus circumnavigano il problema assegnando `<family>` = "`generic`" quando non è possibile assegnare una famiglia specifica al malware in esame.

In Figura 4.1 sono visualizzate le strutture complete dei modelli dei sistemi di naming, generati applicando la **Fase 1** del processo ai 98798 differenti campioni che costituiscono il dataset. I differenti livelli di profondità, che caratterizzano la struttura dei modelli, sono causati dalla differente ricchezza di informazione presente nei label. Si nota che V_1, V_2 possiedono un livello di clusterizzazione molto più profondo rispetto a V_3 e V_4. Questo è da imputare all'informazione maggiormente dettagliata che V_1 e V_2 codificano nei loro label al livello `<version>`. V_4 invece, che è il vendor che all'interno dei propri label fornisce meno informazione di tutti, è quello che presenta una clusterizzazione maggiormente vicina alla radice.

É importante notare che, a questo livello di osservazione, la grande differenza visiva tra i modelli rispecchia la differenza tra le grammatiche, più precisamente rispecchia solamente la differenza sulla quantità di informazione fornita nei label ma, non permette ancora di fare assunzioni sull'inconsistenza tra i sistemi di naming. Ad esempio V_3 e V_4 potrebbero codificare con differenti patter contenuti nelle classi `<family>` e `<platform>`, le differenze che V_1 e V_2 codificano attraverso i pattern presenti nelle classi `<version>` generando così alberi meno profondi. Tali alberi potrebbero però presentare lo stesso numero di foglie degli alberi generati da V_1 e V_2, foglie che potrebbero modellare cluster tra loro completamente overlappanti, e consistenti.

Alternativamente V_1 e V_2 potrebbero fornire solamente dell'informazione aggiuntiva rispetto a V_3 e V_4, cosa che porterebbe portare alla sola generazione di inconsistenze deboli. In tal caso sarebbe sufficiente eliminare l'informazio-

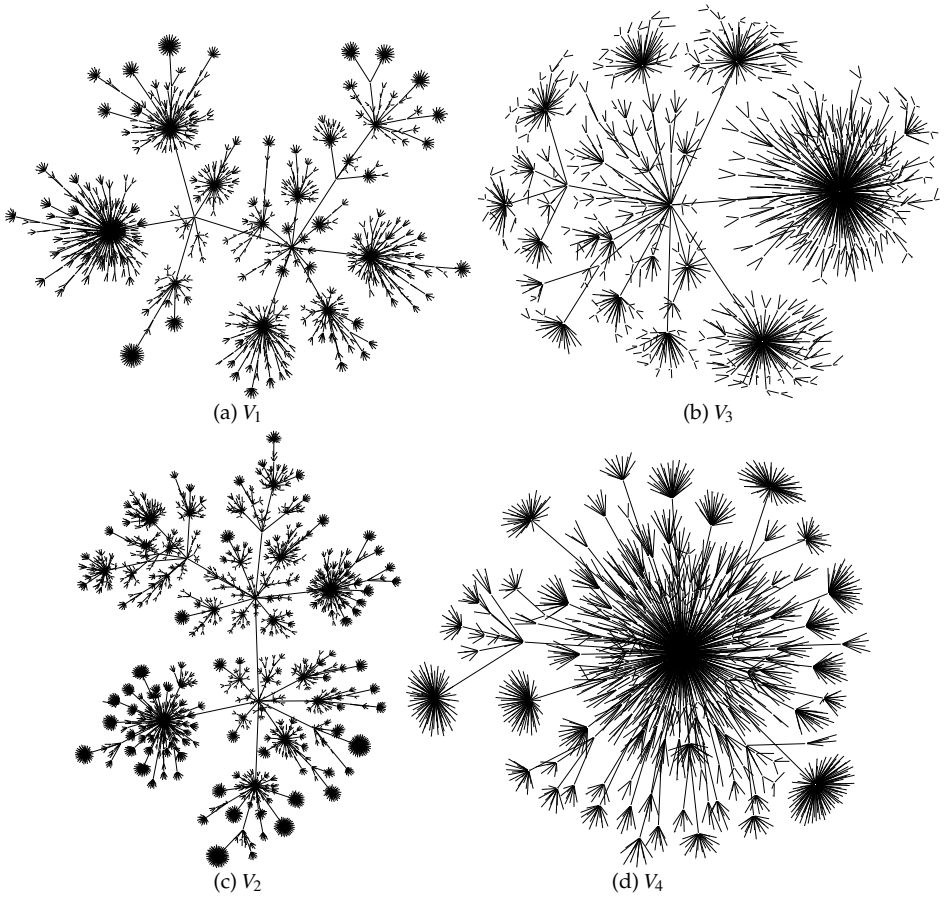


Figura 4.1: Visualizzazione dei modelli dei sistemi di naming di V_1, V_2, V_3 e V_4. I differenti livelli di profondità degli alberi rispecchiano la differente ricchezza di informazione nei label.

ne aggiuntiva proposta da V_1 e V_2 per tornare ad una situazione di completa consistenza.

In Figura 4.2 si vuole proporre uno zoom dei modelli, visualizzando solamente i cluster appartenerti al taglio $\mathcal{C}_{<type>}$. Ovviamente V_4 non è presente nell'immagine in quanto non possiede la classe `<type>`.

A questo livello di zoom, è possibile vedere come i differenti sistemi di naming utilizzino una quantità di informazione differente nella codifica del tipo di threat. Osservando la Figura 4.2 si nota che V_2 è l'antivirus che fornisce un informazione di grana più fine, definendo più spesso, rispetto a V_1 e V_3, eventuali sottotipi. Dall'osservazione dell'immagine ci si può aspettare che, utilizzando solamente l'informazione relativa al tipo, vi sai un certo livello di inconsistenza tra i tre vendor. L'assunzione deriva dal fatto che i tre antivirus tendono a raggruppare lo stesso insieme di malware in un numero di cluster sensibilmente differente l'uno dall'altro, di conseguenza nel

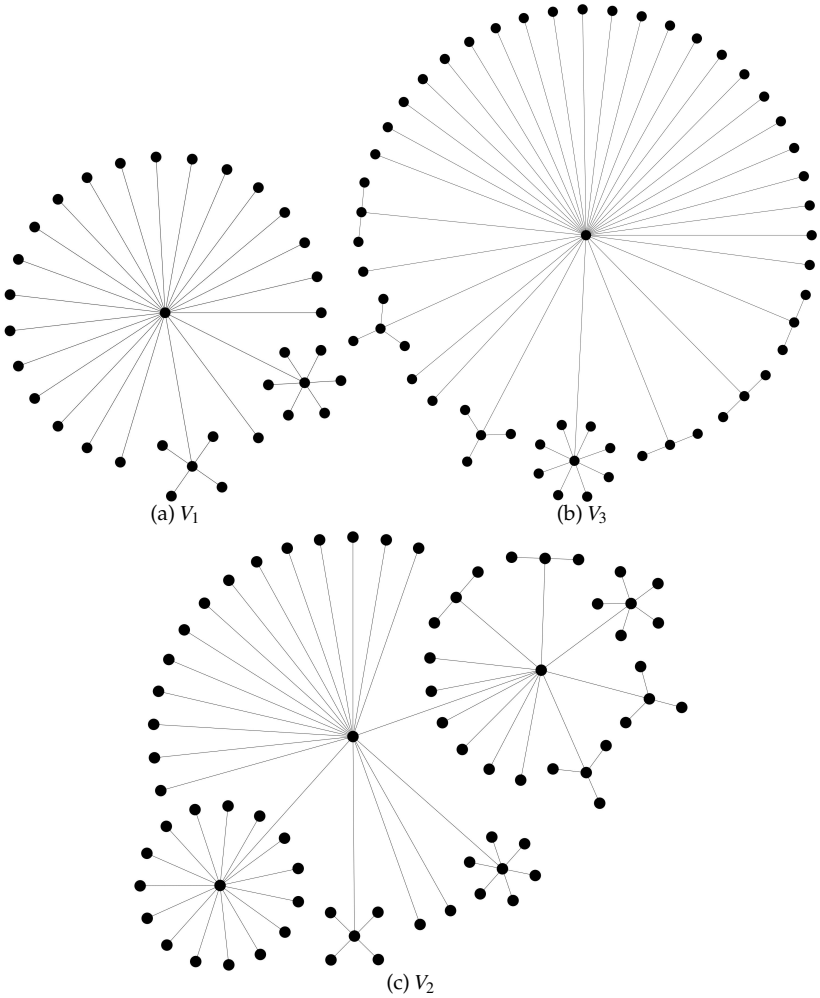


Figura 4.2: Zoom di Figura 4.1 relativo alla clusterizzazione indotta dai pattern presenti nella classe `<type>`. Si nota una differente frequenza di utilizzo dei sotto-tipi oltre che ad un numero di cluster sensibilmente differente. V_4 non compare nell'immagine in quanto non possiede una classe `<type>`.

(a) V_3 (baseline) `packed-Klone` (b) V_1

(c) V_2 (d) V_4

Figura 4.3: Esempio di come i differenti vendor clusterizzano gli stessi malware a livello di taglio $\mathcal{C}_{*/family}$. Si nota anche la differenza della granularità dell'informazione codificata nei label.

caso più ottimistico ci si può aspettare una serie di inconsistenze deboli ed una serie di consistenze, mentre nel caso più pessimistico ci si può aspettare un'unica inconsistenza forte costituita dalla totalità dei cluster.

Nella figura successiva, si vuole invece far vedere come antivirus differenti, a partire dallo stesso insieme di malware, inducono clusterizzazioni completamente differenti. Più precisamente, in Figura 4.3, sono selezionati come insieme di partenza i malware contenuti nel cluster $\mathcal{C}^{V_3}_{klone} \in \mathcal{C}^{V_3}_{*/family}$, ed è visualizzato come tale insieme di malware si sparpaglia sui cluster $\mathcal{C}^{V_*}_{x} \in \mathcal{C}^{V_*}_{*/family}$ degli altri vendor. In figura, per motivi di spazio, non è visualizzato il percorso completo che porta dalla radice dell'albero ai cluster in questione ma, tale percorso è sintetizzato nell'etichette dei cluster.

Anche in quest'immagine è possibile vedere la differenza della granularità dell'informazione codificata nei label da parte dei differenti vendor. V_4 presenta cluster direttamente collegati alla radice, mentre V_2 presenta dei percorsi che variano da un minimo di un passo ad un massimo di tre (escludendo il cluster `not-detected` direttamente collegato alla radice), mentre V_1 continua a presentare cluster non direttamente collegati alla radice, la cui distanza però, è di al massimo due passi.

4.3 Distanza tra modelli

In questa sezione sono proposti i risultati ricavati applicando la definizione di distanza, formalizzata nel Capitolo 3 in Sezione 3.2.1, ai modelli dei sistemi di naming.

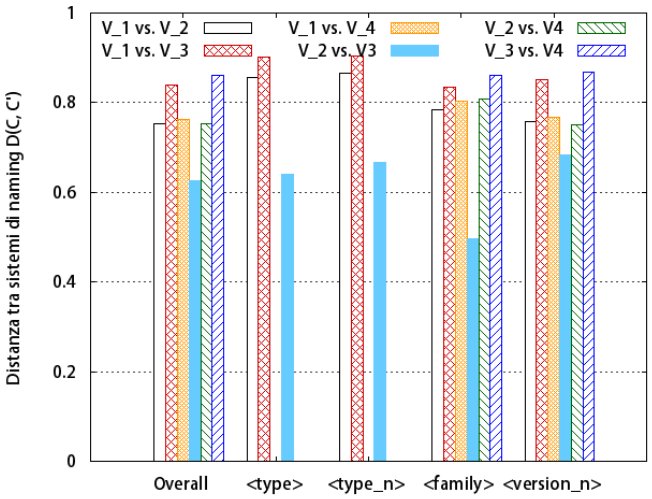
La distanza è stata calcolata per differenti livelli di taglio dell'albero dei cluster. Più precisamente sono stati presi in considerazione i seguenti tagli:

- *Overall* racchiude tutti i cluster presenti nel modello.
- $\mathcal{C}_{<type>}$ racchiude tutti i cluster generati dai pattern contenuti nella classe `<type>`.
- $\mathcal{C}_{<type_n>}$ contiene i cluster indotti dall'ultimo *pattern semplice* contenuto nelle *gerarchie di pattern semplici* appartenenti alla classe `<type>`. Tale taglio permette di valutare quanto sono consistenti i sistemi di naming utilizzando solamente l'informazione relativa al tipo.
- $\mathcal{C}_{<family>}$ contiene i cluster indotti dai pattern contenuti nelle classi `<type>` e `<family>`. Dato che per un vendor è estremamente raro che esistano due label aventi `family` uguale e `type` differente[1] si può ritenere, con lieve approssimazione, che il taglio $\mathcal{C}_{<family>}$ permetta di valutare l'inconsistenza tra i sistemi di naming utilizzando solamente l'informazione contenuta nella classe `<family>`.
- $\mathcal{C}_{<version_n>}$ raggruppa le foglie dell'albero ed è da considerarsi il taglio più importante, in quanto è realizzato utilizzando tutta l'informazione contenuta nei label e di conseguenza è quello che definisce il livello di inconsistenza tra i sistemi di naming.

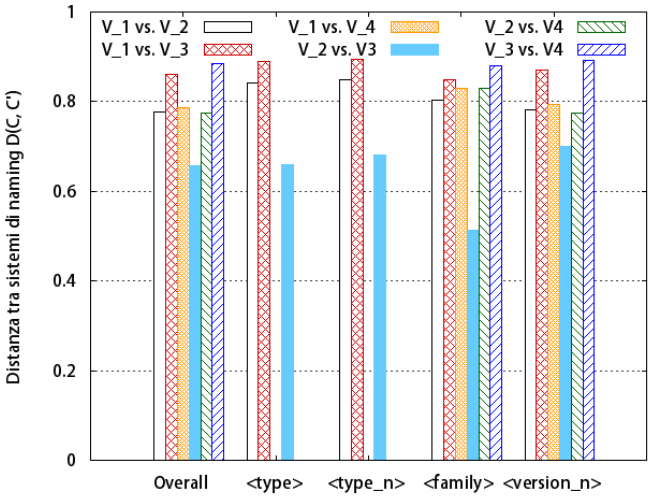
Per riferimenti ed esempi relativi ai tagli dell'albero di cluster si rimanda al Capitolo 3 Sezione 3.1.3.

In Tabella 4.1 sono riportate, per le due differenti definizioni (d_J, d_H), le distanze tra i modelli dei sistemi di naming, calcolate ai differenti tagli. Tali distanze sono sintetizzate in Figura 4.4.

[1] I casi in cui questo accede si riflettono fondamentale in due cluster, dei quali uno è di dimensioni molto ridotte rispetto all'altro



(a) Distanza basata sull'entropia, i.e., utilizzando d_H.



(b) Distanza basata sulla jaccard, i.e., utilizzando d_J

Figura 4.4: Distanza tra sistemi di naming $D(\mathcal{C}, \mathcal{C}')$ calcolata nelle due varianti d_H e d_J. I valori ricavati tramite i due differenti tipi di distanza sono consistenti e generalmente alti. In generale si può notare una maggiore consistenza ai livelli `<family>` e `<version_n>` rispetto al livello `<type>`.

Si può notare che i due tipi di distanza forniscono dati tra loro consistenti e che generalmente l'inconsistenza tra i sistemi di naming è elevata. Più in dettaglio si può osservare che utilizzare solamente l'informazione relativa al tipo di threat, fornisce risultati maggiormente inconsistenti rispetto all'utilizzo dalla sola informazione relativa alla famiglia, o rispetto all'utilizzo della totalità dell'informazione contenuta nei label. Tra tutte le coppie analizzate, la coppia V_2-V_3 fornisce i risultati maggiormente consistenti, soprattutto a livello `<family>` dove i valori della distanza sfiorano lo 0.5. Il fatto che

questa coppia presenti un sensibile degrado della consistenza passando dal livello `<family>` al livello `<version_n>`, è da imputarsi alla maggior quantità d'informazione, relativa alle versioni dei malware, codificata nei label di V_2 rispetto a quelli di V_3. Si può quindi ipotizzare che eliminando l'informazione relativa alla piattaforma e alle versioni dei malware V_2 e V_3 tendano a fornire informazioni consistenti.

4.4 Scatter measure

In questa sezione sono esposti i risultati relativi alla scatter measure $S(\mathcal{C}^A, \mathcal{C}^B)$ (definita in Capitolo 3 Sezione 3.2.2). L'analisi è stata effettuata per i tagli $\mathcal{C}_{type_n}$, $\mathcal{C}_{family}$, $\mathcal{C}_{version_n}$ e per differenti valori di T_σ. I risultati organizzati per tagli sono riportati rispettivamente in Tabella 4.2 4.3 4.4 e visualizzati in Figura 4.5.

Valori di T_σ molto bassi (e.g., 1%) rappresentano casi molto ottimistici, in quanto l'insieme di scatter $\widehat{\Gamma}(c, \mathcal{C}')$ deve coprire c solo per una piccola parte, mentre valori molto elevati di T_σ (e.g., 100%) rappresentano casi realistici in cui è richiesto che $\widehat{\Gamma}(c, \mathcal{C}')$ copra c in maniera molto significativa.

I risultati visualizzati in Figura 4.5 confermano in generale la miglior consistenza del taglio $\mathcal{C}_{version_n}$ (Figura 4.5c) rispetto al taglio $\mathcal{C}_{family}$ (Figura 4.5b) ed al taglio $\mathcal{C}_{type_n}$ (Figura 4.5a). Anche in quest'analisi la coppia $V_2 - V_3$ risulta la più consistente e continua a fornire i migliori risalutati per il taglio $\mathcal{C}_{family}$.

In media si può affermare che per i tagli $\mathcal{C}_{family}$ e $\mathcal{C}_{version_n}$, la scatter measure non ha fornito valori negativi, in quanto un insieme di scatter è costituito al massimo da sei cluster, prelevati da un insieme la cui cardinalità varia da circa 500 a 4200 cluster (la dimensione dipende dalla coppia in analisi ed dal livello considerato). Non si può affermare lo stesso per il taglio $\mathcal{C}_{type_n}$ che presenta insiemi di scatter che variano da un minimo di quattro elementi ad un massimo di nove selezionati da un insieme che varia tra i 30 ed i 61 cluster.

Si può quindi affermare che in generale un label generato da un sistema di naming non è in relazione con molti altri label generati da un sistemi di naming differenti. Questo risultato che appare ottimistico sarà decisamente peggiorato da alcune considerazioni fornite assieme all'analisi successiva, che dimostrerà la criticità del problema.

4.5 Analisi inconsistenze

In questa sezione sono forniti i risultati ricavati dal processo di estrazione delle inconsistenze, definito in Capitolo 3 Sezione 3.2.3. I risultati dell'analisi sono presentati separatamente, in paragrafi differenti, in base all'insieme di cluster preso in considerazione (i.e., $\mathcal{C}_{type_n}$, $\mathcal{C}_{family}$, $\mathcal{C}_{version_n}$). Le conclusioni generali relative a quest'analisi sono tratte in un paragrafo separato.

Taglio	$\mathbf{V_1 - V_2}$	$\mathbf{V_1 - V_3}$	$\mathbf{V_1 - V_4}$	$\mathbf{V_2 - V_3}$	$\mathbf{V_2 - V_4}$	$\mathbf{V_3 - V_4}$
Overall	0.7523	0.8402	0.7621	0.6252	0.7519	0.8615
type	0.8555	0.9014	/	0.6407	/	/
type_n	0.8657	0.9043	/	0.6664	/	/
family	0.7845	0.8348	0.8031	0.4972	0.8090	0.8615
version_n	0.7579	0.8506	0.7683	0.6838	0.7516	0.8688

(a) Distanza basata sulla jaccard, i.e., utilizzando d_J

Taglio	$\mathbf{V_1 - V_2}$	$\mathbf{V_1 - V_3}$	$\mathbf{V_1 - V_4}$	$\mathbf{V_2 - V_3}$	$\mathbf{V_2 - V_4}$	$\mathbf{V_3 - V_4}$
Overall	0.7763	0.8613	0.7867	0.6566	0.7758	0.8857
type	0.8415	0.8899	/	0.6585	/	/
type_n	0.8501	0.8948	/	0.6802	/	/
family	0.8029	0.8486	0.8295	0.5141	0.8294	0.8803
version_n	0.7807	0.8703	0.7930	0.6992	0.7754	0.8930

(b) Distanza basata sull'entropia, i.e., utilizzando d_H

Tabella 4.1: Distanza tra i modelli

$\mathbf{T_{\sigma\%}}$	$\mathbf{V_1 - V_2}$	$\mathbf{V_1 - V_3}$	$\mathbf{V_2 - V_3}$
1	1.0000	1.0000	1.0000
5	1.0000	1.0000	1.0000
10	1.0000	1.0000	1.0000
15	1.0000	1.0000	1.0000
20	1.0161	1.0000	1.0000
25	1.0484	1.0161	1.0000
30	1.1371	1.0370	1.0208
35	1.1936	1.0635	1.0208
40	1.2823	1.0901	1.0417
45	1.3710	1.2426	1.0706
50	1.4758	1.3441	1.1075
55	1.7177	1.4987	1.2621
60	1.7984	1.6258	1.3441
65	2.0806	1.7376	1.4560
70	2.2500	1.8693	1.5726
75	2.5000	2.0437	1.6153
80	2.9274	2.2606	1.7228
85	3.4677	2.6038	1.8474
90	4.1855	3.0749	2.1052
95	5.6532	3.8464	2.5084
100	8.6210	5.6999	3.9194

Tabella 4.2: Risultati scatter measure per il taglio $\mathcal{C}_{type_n}$ al variare di T_σ

$T_{\sigma\%}$	$V_1 - V_2$	$V_1 - V_3$	$V_1 - V_4$	$V_2 - V_3$	$V_2 - V_4$	$V_3 - V_4$
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
10	1.0010	1.0000	1.0000	1.0000	1.0000	1.0000
15	1.0058	1.0011	1.0031	1.0000	1.0035	1.0005
20	1.0232	1.0063	1.0185	1.0000	1.0129	1.0053
25	1.0548	1.0211	1.0411	1.0021	1.0401	1.0155
30	1.1041	1.0536	1.0888	1.0052	1.0943	1.0493
35	1.1880	1.1210	1.1782	1.0172	1.1826	1.1161
40	1.2416	1.1643	1.2326	1.0245	1.2406	1.1567
45	1.3421	1.2421	1.3311	1.0413	1.3465	1.2428
50	1.4074	1.2893	1.3882	1.0564	1.4224	1.2884
55	1.6643	1.5273	1.6675	1.1981	1.6734	1.5312
60	1.7874	1.6215	1.7882	1.2526	1.7926	1.6177
65	1.9808	1.7820	1.9710	1.3192	1.9672	1.7723
70	2.2731	2.0295	2.2488	1.4247	2.2566	2.0143
75	2.5248	2.2402	2.4789	1.4715	2.4790	2.1900
80	2.8916	2.5535	2.8205	1.5589	2.8268	2.4518
85	3.3680	2.9764	3.2838	1.6768	3.3052	2.8598
90	3.9487	3.4930	3.8711	1.8208	3.9157	3.3749
95	4.7595	4.2143	4.7790	2.0561	4.9088	4.2075
100	5.5837	5.0062	6.0406	2.5312	6.2195	5.4327

Tabella 4.3: Risultati scatter measure per il taglio $\mathcal{C}_{family}$ al variare di T_σ

Prima di fornire i risultati dell'analisi, è però necessario analizzare due casi particolari che si sono venuti a creare durante l'analisi: *nodi isolati* e *cluster* $\mathcal{C}_{not-detected}$.

Nodi isolati sono quei cluster che, a causa della rimozione degli archi del grafo, non sono in relazione con nessun altro cluster. Tali cluster sono ambigui, in quanto per un certo valore $\widehat{T}_w$ possono collegarsi ad altri *nodi isolati*, o ad altre componenti connesse, andando a creare rispettivamente una consistenza o un 'inconsistenza debole o forte. Per non sfalsare le misure, tali cluster, data la loro amibiguità, non sono considerati ne consistenze ne inconsistenze e non prendono parte al conteggio nei risultati finali.

Cluster $\mathcal{C}_{\mathbf{not-detected}}$ sono i cluster che raggruppano i campioni non considerati malevoli in accordo all'analisi di un determinato antivirus. Tali cluster sono caratterizzati dall'avere dimensioni molto grandi e dal propagare le inconsistenze, portandole ad includere la quasi totalità dei nodi presenti nel grafo. Dato che la relazione tra un qualunque cluster e i

$T_{\sigma\%}$	$V_1 - V_2$	$V_1 - V_3$	$V_1 - V_4$	$V_2 - V_3$	$V_2 - V_4$	$V_3 - V_4$
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
10	1.0014	1.0004	1.0000	1.0000	1.0000	1.0000
15	1.0051	1.0020	1.0027	1.0004	1.0027	1.0013
20	1.0199	1.0079	1.0116	1.0017	1.0073	1.0067
25	1.0395	1.0274	1.0300	1.0070	1.0226	1.0228
30	1.0751	1.0608	1.0634	1.0183	1.0472	1.0516
35	1.1440	1.1231	1.1352	1.0485	1.1012	1.1031
40	1.1763	1.1575	1.1700	1.0649	1.1254	1.1313
45	1.2423	1.2323	1.2417	1.1021	1.1799	1.2056
50	1.2846	1.2728	1.2759	1.1161	1.2118	1.2384
55	1.5394	1.5361	1.5429	1.2895	1.4434	1.4732
60	1.6168	1.6252	1.6151	1.3303	1.5036	1.5405
65	1.7356	1.7707	1.7323	1.4094	1.6026	1.6592
70	1.9434	2.0046	1.9374	1.5475	1.7857	1.8536
75	2.0851	2.1809	2.0590	1.6268	1.9096	1.9660
80	2.3067	2.4692	2.2803	1.7825	2.1167	2.1688
85	2.5627	2.8190	2.5527	1.9828	2.3865	2.4487
90	2.8270	3.2169	2.8516	2.2263	2.6809	2.7589
95	3.1358	3.7311	3.2446	2.5233	3.0529	3.1913
100	3.3706	4.1755	3.6447	2.8560	3.4090	3.6940

Tabella 4.4: Risultati scatter measure per il taglio $\mathcal{C}_{version_n}$ al variare di T_σ

cluster $\mathcal{C}_{not-detected}$ più che una relazione tra label è una relazione tra le differenti capacità di scansione dei due prodotti[2], per evitare di sfalsare i risultati dell'analisi a causa dell'introduzione di enormi componenti connesse, tali cluster sono stati rimossi dall'insieme dei cluster in analisi. La rimozione dei cluster $\mathcal{C}_{not-detected}$ può generare *nodi isolati*, in quanto è possibile che vi siano dei cluster che sono un sottoinsieme di $\mathcal{C}_{not-detected}$ e che di conseguenza siano collegati solamente a loro, tali cluster per rimanere coerenti con la decisione precedente, relativa ai *nodi isolati*, non prendono parte al conteggio nei risultati finali.

Analisi a livello $\mathcal{C}_{\mathbf{type_n}}$ I risultati relativi all'analisi di questo livello sono riportati in Tabella 4.5 e visualizzati in Figura 4.8. Si nota che, a questo livello, la consistenza tra i sistemi di naming è molto bassa. Le coppie $V_1 - V_2$

[2]La relazione si può verificare sia quando un antivirus fornisce falsi positivi, sia quando un antivirus non è in grado di riconosce un malware, quindi la relazione non è da considerare come indice di maggiore bontà dell'antivirus che riconosce il campione come malevolo rispetto all'altro.

e $V_1 - V_3$ per $T_w = 0_{\%}$ presentano un unica enorme inconsistenza di tipo forte che coinvolge tutti i cluster appartenenti al grafo, mentre la coppia $V_2 - V_3$, che continua a fornire i risultati migliori, per $T_w = 0_{\%}$ oltre ad un unica grande inconsistenza di tipo forte presenta anche due consistenze (i.e., Figura 4.8b). Per $T_w = 10_{\%}$, a differenza delle altre coppie che presentano risultati quasi invariati, i risultati di $V_2 - V_3$ migliorano nettamente, dall'unica grande inconsistenza forte si generano: inconsistenze forti di dimensioni inferiori, inconsistenze deboli e alcune consistenze (i.e., Figura 4.8c).

Il fatto che il numero di inconsistenze forti per la coppia $V_1 - V_2$ scenda a zero per $T_w = 40_{\%}$ (i.e., Figura 4.8e), non è da considerarsi come indice di maggiore consistenza di $V_1 - V_2$ rispetto a $V_2 - V_3$, in quanto, quest'ultima coppia presenta uno svariato numero di consistenze rispetto a $V_1 - V_2$, che invece, tende a generare solamente un grande numero di *nodi isolati*. Questo porta ad affermare che il grafo generato dalla coppia $V_1 - V_2$ è costituito nella sua totalità da relazioni poco significative.

	$\mathbf{T_w = 0_{\%}}$			$\mathbf{T_w = 10_{\%}}$			$\mathbf{T_w = 20_{\%}}$			$\mathbf{T_w = 40_{\%}}$		
Coppia	**IF**	**ID**	**C**	**IF**	**ID**	**C**	**IF**	**ID**	**C**	**IF**	**ID**	**C**
$\mathbf{V_1 - V_2}$	1	0	0	1	0	0	2	4	5	0	5	11
$\mathbf{V_1 - V_3}$	1	0	0	1	0	0	3	1	2	1	1	9
$\mathbf{V_2 - V_3}$	1	0	2	5	5	13	4	6	19	2	6	24

(a) Numero di componenti connesse classificate come: inconsistenze forti (IF), inconsistenze deboli (ID), consistenze (C).

$\mathbf{T_w}$%	$\mathbf{V_1 - V_2}$	$\mathbf{V_1 - V_3}$	$\mathbf{V_2 - V_3}$
0	84.0	72.0	96.0
10	79.0	65.0	10.6
20	24.5	15.7	9.25
40	0.0	10.0	12.0

(b) Dimensione media delle Inconsistenze Forti

Tabella 4.5: Risultati relativi all'estrazione delle inconsistenze a livello $\mathcal{C}_{type}$. Si nota la presenza di inconsistenze di tipo forte molto grandi per bassi valori di T_W

Analisi a livello $\mathcal{C}_{\mathbf{family}}$ I risultati relativi all'analisi di questo livello sono riportati in Tabella 4.6 e visualizzati in Figura 4.9. In generale si nota che: per $T_W = 0\%$ e per $T_W = 10\%$ (i.e., Figura 4.9b e Figura 4.9c) sono presenti pochissime inconsistenze di tipo forte di enormi dimensioni e che il numero di consistenze rimane contenuto. La coppia $V_2 - V_3$ si distingue dalle altre ancora un volta, presentando risultati molto più consistenti. Il numero di consistenze rilevate per questa coppia e nettamente superiore al numero di consistenze estratte per le altre coppie, inoltre la dimensione media delle in-

consistenze di tipo forte di $V_2 - V_3$ risulta nettamente inferiore a quella delle altre coppie (i.e., Figura 4.9a).

A questo livello di analisi si può concludere non vi è consistenza tra le coppie che sono caratterizzate dal presentare poche inconsistenze di grandi dimensioni.

	$\mathbf{T_w = 0\%}$			$\mathbf{T_w = 10\%}$			$\mathbf{T_w = 20\%}$			$\mathbf{T_w = 40\%}$		
Coppia	**IF**	**ID**	**C**	**IF**	**ID**	**C**	**IF**	**ID**	**C**	**IF**	**ID**	**C**
$\mathbf{V_1 - V_2}$	4	4	88	5	8	109	11	30	158	40	81	251
$\mathbf{V_1 - V_3}$	2	4	69	5	7	94	15	36	144	28	67	222
$\mathbf{V_1 - V_4}$	1	3	34	1	4	38	3	11	62	16	38	139
$\mathbf{V_2 - V_3}$	12	7	392	35	24	484	38	41	579	28	52	671
$\mathbf{V_2 - V_4}$	1	1	39	1	2	43	3	21	74	21	37	162
$\mathbf{V_3 - V_4}$	1	0	32	1	1	39	3	15	78	12	35	133

(a) Numero di componenti connesse classificate come: inconsistenze forti (IF), inconsistenze deboli (ID), consistenze (C).

$\mathbf{T_w\%}$	$\mathbf{V_1 - V_2}$	$\mathbf{V_1 - V_3}$	$\mathbf{V_1 - V_4}$	$\mathbf{V_2 - V_3}$	$\mathbf{V_2 - V_4}$	$\mathbf{V_3 - V_4}$
0	412.6	724.0	1293.0	93.7	1413.0	1196.0
10	315.0	270.6	1275.0	24.9	1393.0	1159.0
20	120.0	68.7	385.3	15.7	397.3	315.7
40	15.5	14.8	43.2	9.9	30.1	40.2

(b) Dimensione media delle Inconsistenze Forti

Tabella 4.6: Risultati relativi all'estrazione delle inconsistenze a livello $\mathcal{C}_{family}$. Si nota la presenza di inconsistenze di tipo forte molto grandi per bassi valori di T_W, la coppia $V_2 - V_3$ si distingue nuovamente come la migliore.

Analisi a livello $\mathcal{C}_{\mathbf{version_n}}$ I risultati relativi all'analisi di questo livello sono riportati in Tabella 4.7 e visualizzati in Figura 4.10. Anche in questo caso, per valori bassi di T_W (i.e., $T_W = 0\%$ Figura 4.10b e $T_W = 10\%$ Figura 4.10c), si nota la presenta di poche inconsistenze di tipo forte di grosse dimensioni. La coppia $V_2 - V_3$ continua a presentarsi come la coppia più consistente, fornendo un maggior numero di consistenze rispetto alle altre coppie (per $T_W \leq 20\%$) e fornendo inconsistenze di tipo forte aventi una dimensione nettamente minore (Figura 4.10a).

Considerazioni generali L'analisi ha evidenziato una carenza generale di consistenza a tutti i livelli presi in esame, tale carenza è dovuta principalmente alla costante presenza di poche inconsistenze di tipo forte di grosse dimensioni. Il livello $\mathcal{C}_{version}$ si può considerare in generale il più consistente, infatti,per valori bassi di T_W, dimostra una dimensione media delle inconsi-

	$T_w = 0\%$			$T_w = 10\%$			$T_w = 20\%$			$T_w = 40\%$		
Coppia	**IF**	**ID**	**C**	**IF**	**ID**	**C**	**IF**	**ID**	**C**	**IF**	**ID**	**C**
$V_1 - V_2$	16	49	290	21	63	300	47	119	378	95	214	616
$V_1 - V_3$	4	22	92	9	28	105	33	73	149	47	128	249
$V_1 - V_4$	11	24	149	13	27	157	26	50	206	51	124	396
$V_2 - V_3$	48	153	338	89	169	351	112	223	386	115	276	494
$V_2 - V_4$	8	22	286	12	32	296	19	66	354	58	141	563
$V_3 - V_4$	5	12	52	8	15	63	23	37	99	29	74	213

(a) Numero di componenti connesse classificate come: inconsistenze forti (IF), inconsistenze deboli (ID), consistenze (C).

$T_w\%$	$V_1 - V_2$	$V_1 - V_3$	$V_1 - V_4$	$V_2 - V_3$	$V_2 - V_4$	$V_3 - V_4$
0	290.0	676.3	318.8	70.6	577.0	491.8
10	217.6	291.3	267.3	36.7	380.0	300.3
20	86.8	66.1	124.3	26.0	222.6	91.7
40	28.8	26.0	43.3	18.1	53.6	45.7

(b) Dimensione media delle Inconsistenze Forti

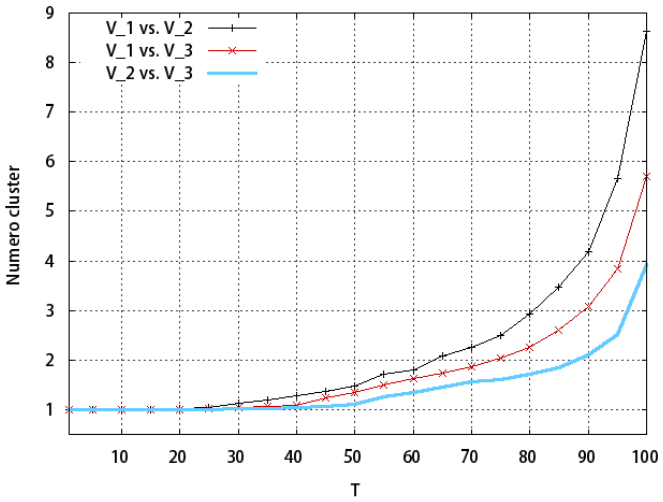
Tabella 4.7: Risultati relativi all'estrazione delle inconsistenze a livello $\mathcal{C}_{version_n}$. $V_2 - V_3$ continua ad essere la coppia che fornisce i risultati migliori. Per T_W bassi si nota la presenza di grosse inconsistenze di tipo forte.

stenze più bassa [3] ed un numero maggiore di consistenze. La coppia $V_2 - V_3$ si dimostra la coppia più consistente ad ogni livello, ed in particolare fornisce il valore massimo di consistenza per il livello $\mathcal{C}_{family}$, confermando tutti i risultati delle analisi precedenti.

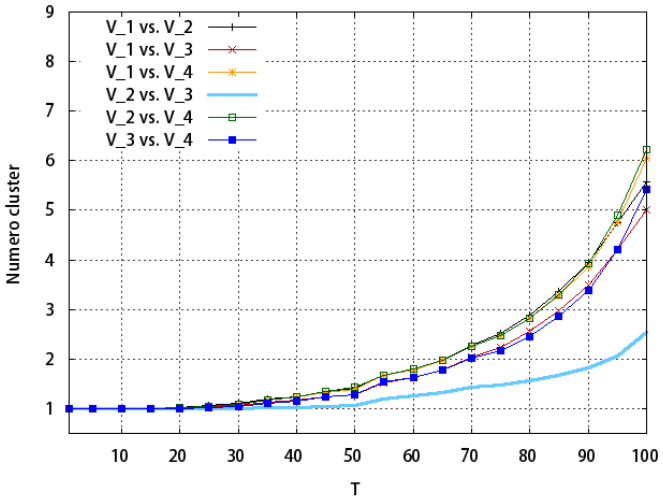
I risultati di quest'analisi permettono di guardare con una criticità differente i buoni risultati forniti dalla scatter measure, infatti è vero che un cluster in generale è sparpagliato su pochi cluster, quindi data un label ad esso ne sono associati pochi della controparte, però partendo da un label l ed estraendo l'insieme dei label della controparte a cui l è associato, effettuare l'associazione inversa di questo insieme tende a propagare l'inconsistenza.

In Figura 4.6 sono proposte due delle inconsistenze estratte per la coppia $V_2 - V_3$, si nota che l'inconsistenza debole (Figura 4.6a) è costituita da label di V_2 che sono effettivamente specializzazioni del label di V_3. In Figura 4.7 è visualizzata una delle inconsistenze forti di grandi dimensioni, il numero di interconnessioni è estremamente elevato e lascia intuire che nessun tipo di relazione tra i label costituirebbe una soluzione efficace al problema del naming.

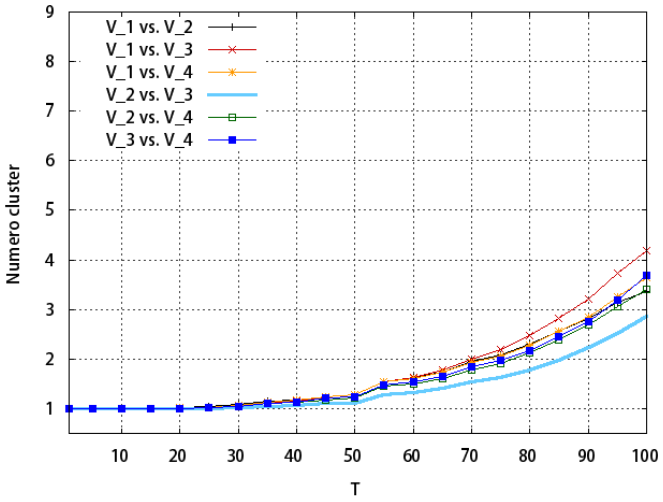
[3] Il livello $\mathcal{C}_{type}$ presenta dimensioni medie delle inconsistenze forti molto inferiori, però è da considerare il fatto che i cluster presi in esame per tale livello sono molto pochi e che generalmente l'unica inconsistenza forte ingloba tutto il grafo.



(a) Scatter measure $\mathcal{C}_{type_n}$

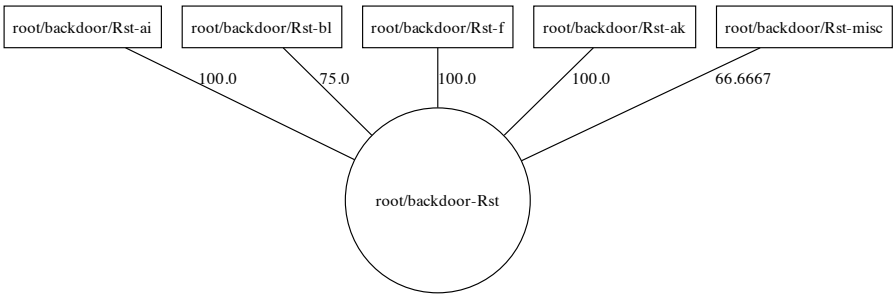


(b) Scatter measure $\mathcal{C}_{family}$

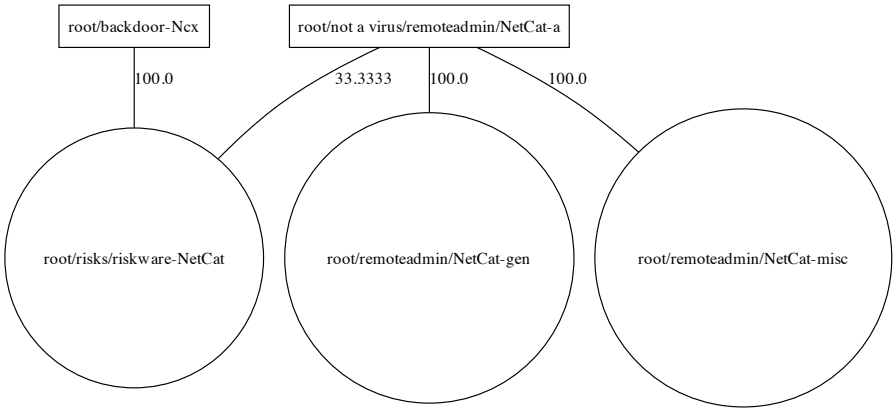


(c) Scatter measure $\mathcal{C}_{version_n}$

Figura 4.5: Scatter measure valutata per differenti valori di T_σ. $T_\sigma = 1\%$ può essere considerato un caso ottimistico in quanto la copertura del cluster richiesta da parte dell'insieme di scatter $\widehat{\Gamma}(c, \mathcal{C}')$ è quasi nulla, mentre $T_\sigma = 100\%$ rappresenta il caso realistico in cui è richiesta la totale copertura di c da parte di $\widehat{\Gamma}(c, \mathcal{C}')$. La coppia $V_2 - V_3$ si conferma nuovamente la coppia più consistente, in particolare a livello `<family>`.



(a) Inconsistenza debole



(b) Inconsistenza forte

Figura 4.6: Esempio reale di inconsistenza debole Figura (a) e forte Figura (b) tra V_2 (rettangoli) e V_3 (cerchi).É interessante notare come una selezione casuale di un inconsistenza debole sia proprio un caso di specializzazione sintattica oltre che semantica

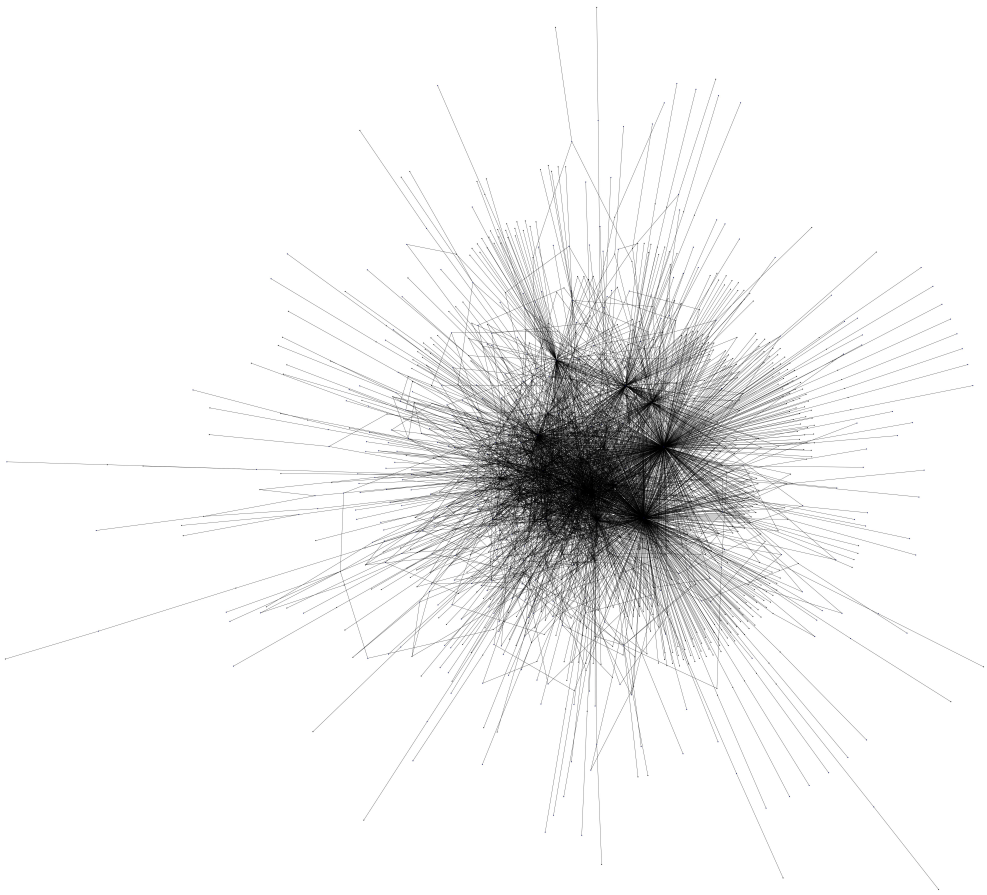
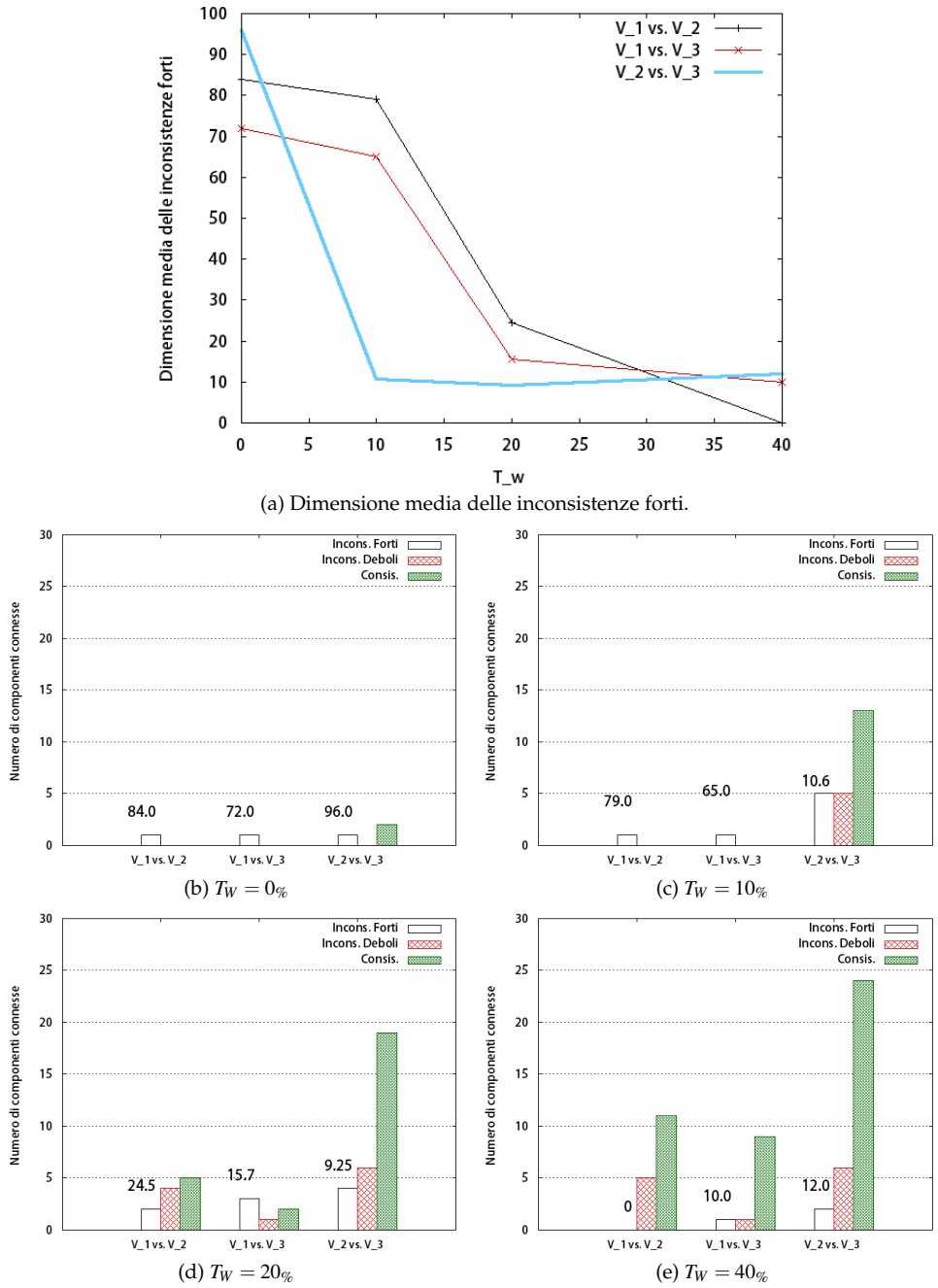


Figura 4.7: Inconsistenza di tipo forte estratta dalla coppia V_3-V_4. A differenza dell'inconsistenza presentata in Figura 4.6b la dimensione di questa componetene connessa è estremamente elevata (i.e., 1293 nodi) e presenta un livello di complessità e numerosità delle interconnessioni davvero critico.



(a) Dimensione media delle inconsistenze forti.

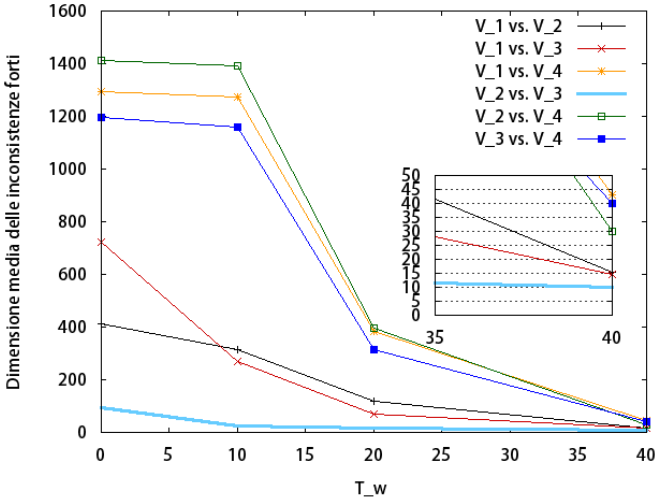
(b) $T_W = 0_{\%}$

(c) $T_W = 10_{\%}$

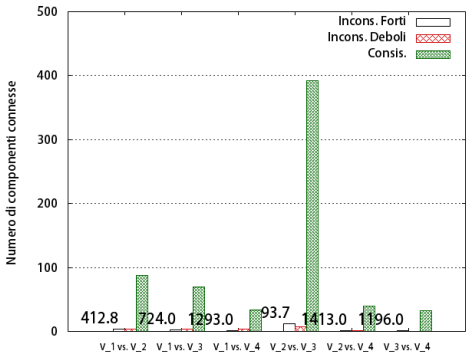
(d) $T_W = 20_{\%}$

(e) $T_W = 40_{\%}$

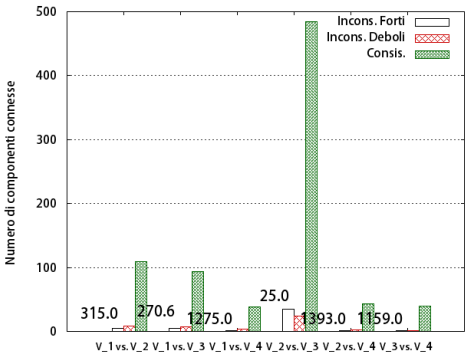
Figura 4.8: Dimensione media delle inconsistenze forti e numero di componenti fortemente connesse estratte al variare di T_W a livello $\mathcal{C}_{type_n}$. Per le componenti connesse classificate come inconsistenze di tipo forte è riportato il numero medio di cluster che le costituiscono sia sull'istogramma sia nel relativo grafico.



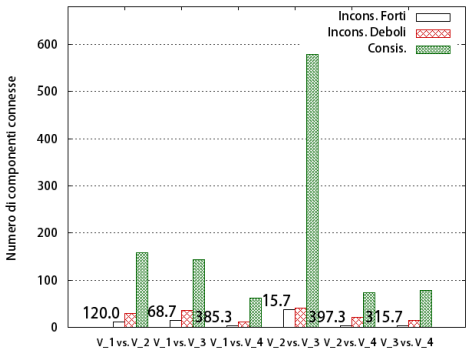
(a) Dimensione media delle inconsistenze forti.



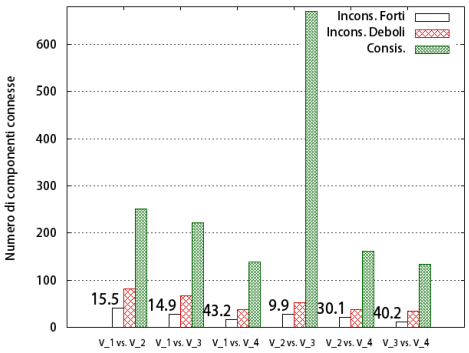
(b) $T_W = 0\%$



(c) $T_W = 10\%$

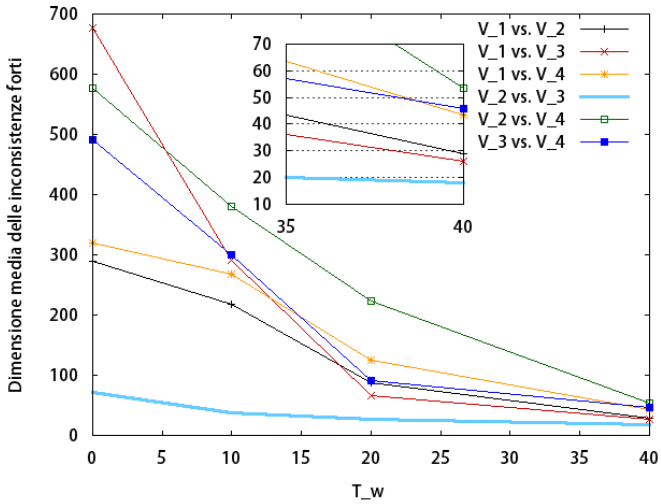


(d) $T_W = 20\%$

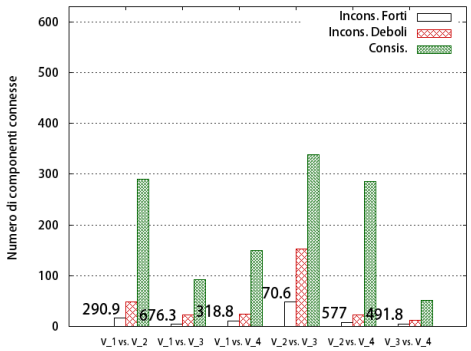


(e) $T_W = 40\%$

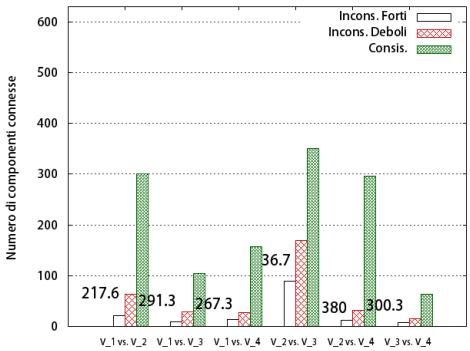
Figura 4.9: Dimensione media delle inconsistenze forti e numero di componenti fortemente connesse estratte al variare di T_W a livello $\mathcal{C}_{family}$. Per le componenti connesse classificate come inconsistenze di tipo forte è riportato il numero medio di cluster che le costituiscono, sia sull'istogramma, sia nel relativo grafico.



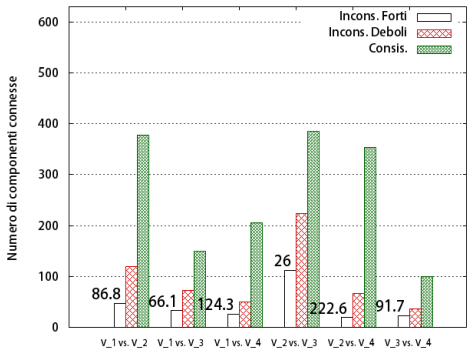
(a) Dimensione media delle inconsistenze forti.



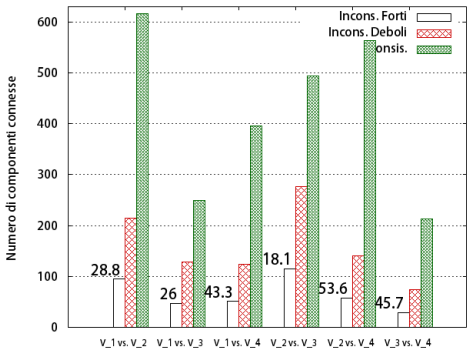
(b) $T_W = 0\%$



(c) $T_W = 10\%$



(d) $T_W = 20\%$



(e) $T_W = 40\%$

Figura 4.10: Dimensione media delle inconsistenze forti e numero di componenti fortemente connesse estratte al variare di T_W a livello $\mathcal{C}_{version_n}$. Per le componenti connesse classificate come inconsistenze di tipo forte è riportato il numero medio di cluster che le costituiscono, sia sull'istogramma, sia nel relativo grafico.

Capitolo 5

Conclusioni e sviluppi futuri

In questa tesi è stato proposto un approccio che permette di analizzare in maniera sistematica le inconsistenze tra i sistemi di naming adottati da antivirus differenti. Tale approccio può essere suddiviso in due fasi, la prima si preoccupa di modellare i sistemi di naming come grafi a partire da un insieme di malware, mentre la seconda fase confronta i modelli generati, estraendo le inconsistenze strutturali e due indici che descrivono il livello globale di inconsistenza.

L'approccio proposto è stato implementato ed utilizzato per analizzare l'eventuale inconsistenza presente tra sei coppie di antivirus reali, utilizzando come dataset un insieme di circa 100.000 differenti malware reali.

I risultati ricavati dall'analisi evidenziano anzitutto la presenza di porzioni di modelli fortemente inconsistenti tra loro, dimostrazione del fatto che il problema del naming è un problema di tipo semantico-strutturale e non una mera differenza sintattica. Un altro risultato, direttamente collegato ai problemi strutturali, è la dimostrazione che quando un cluster di un vendor è associato in maniera inconsistente ai cluster di un altro vendor, effettuare l'associazione inversa di tali cluster tende a propagare l'inconsistenza invece che a racchiuderla in un piccolo sottoinsieme di cluster. Questo sminuisce il risultato ottimistico della scatter measure, in quanto è vero che un label è associato a pochi altri label, ma è anche vero, che se si tenta di effettuare l'associazione inversa a partire da quei label si va a creare una rete di collegamenti completamente inutile ai fini della risoluzione del problema del naming.

La distanza tra i modelli, in generale elevata, dimostra che il problema del naming esiste, e che, contrariamente alle aspettative, per la maggior parte dei casi, utilizzare la totalità dell'informazione contenuta nei label tende a generare risultati più consistenti rispetto ai casi in cui ne è utilizzata solamente una parte.

Infine, il grado di consistenza maggiore, dimostrato dalla coppia V_2-V_3,

rispetto alla altre coppie, permette di affermare che l'inconsistenza non è distribuita equamente su ogni coppia, e che i risultati ricavati dall'analisi di una coppia non permettono di effettuare congetture su altre coppie.

Limitazioni e sviluppi futuri L'approccio proposto è affetto da alcune limitazioni che saranno oggetto di futuri sviluppi ed implementazioni. Innanzitutto l'estrazione dei `pattern` in **Fase 1** richiede l'intervento manuale di un analista, in modo da associare i `pattern` estratti alle corrette classi (e.g., `<type>`, `<family>`). Per superare questa limitazione è necessario uno sforzo della comunità di ricerca, oppure un sostanziale aiuto da parte dei vendor che potrebbero rendere pubbliche le euristiche con cui generano i label, rendendo necessaria la revisione manuale solamente per quei label che non seguono nessun tipo di convenzione. In ogni caso anche senza l'aiuto dei vendor l'approccio proposto è completamente fattibile. Per quanto riguarda la fase sperimentale di questa tesi è stato necessario intervenire manualmente solamente sui `pattern` appartenei alla classe `<type>`, impiegando circa 30 minuti per vendor.

Un altra limitazione riguarda la vista statica fornita dal dataset sui sistemi di naming. Come esplicitato in [Canto et al., 2008] le convenzioni di naming possono variare nel tempo, inoltre ogni giorno sono prodotti numerosi nuovi malware, a cui sono associati nuovi label. Tali label possono rendere i sistemi più o meno consistenti. Questa limitazione può essere superata modificando il modello, ed introducendo un concetto di evoluzione temporale, che permetterebbe di tenere traccia dell'andamento del livello di inconsistenza nel tempo, permettendo così di prevedere eventuali trend futuri.

L'ultima limitazione, anche se non direttamente collegata al metodo proposto, riguarda il dataset su cui cui sono state effettuate le analisi. VirusTotal, per motivazioni relative al design del sistema, esegue gli antivirus con cui effettua lo scan dei malware da linea di comando. Un antivirus eseguito da linea di comando presenta differenti capacità di rilevazione rispetto alla sua versione Desktop. Questo può quindi portare, in base alla modalità con cui è eseguito l'antivirus, alla generazione di label differenti per uno stesso malware. In ogni caso, l'importanza di VirusTotal nella fase sperimentale di questa tesi, deriva dal fatto che permette di effettuare delle query sullo storico delle sue scansioni; in particolare permette di ricavare una lista di malware ordinata per il numero di scansioni richieste, che identificano in un certo senso la popolarità del malware e l'interesse delle comunità alla sua analisi.

Appendice A

Distanza $d_H(c, c')$

Lo scopo di questo appendice è quello di fornire una descrizione dettagliata della distanza $d_H(c, c')$ e di come tale distanza può essere utilizzata per calcolare il grado di similitudine di due cluster.

Dato il vendor A, ad ogni malware w presente nel dataset è possibile associare un vettore binario, $\overline{\mathcal{V}_A}$ costituito da $|\mathcal{C}^A|$ elementi, definito come segue:

$$\overline{\mathcal{V}_A}[i] = \begin{cases} 1 & \text{se } w \in c_{a_i} \\ 0 & \text{se } w \notin c_{a_i} \end{cases}$$

dove c_{a_i} è l'i-esimo cluster del vendor A (i.e., $c_{a_i} \in \mathcal{C}^A$), mentre l'i-esima colonna di $\overline{\mathcal{V}_A}$ può essere considerata la realizzazione di una variabile casuale binomiale $X_{c_{a_i}} \sim Bin(\frac{n_i}{|\mathcal{M}|})$, che modella il cluster, dove $n_i = |c_{a_i}|$ e $|\mathcal{M}|$ è il numero di malware presenti nel dataset. Lo stesso ragionamento può essere ripetuto per un ipotetico vendor B, il quale assocerà ad ogni malware w un altro vettore $\overline{\mathcal{V}_B}$, costituito da $|\mathcal{C}^B|$ realizzazioni delle variabili casuali $X_{c_{b_j}} \sim Bin(\frac{m_j}{|\mathcal{M}|})$ dove $m_j = |c_{b_j}|$.
Per ogni possibile coppia di variabili casuali $X_{c_{a_i}}$, $X_{c_{b_j}}$ è possibile definire la distribuzione congiunta:

$$p_{X_{c_{a_i}}, X_{c_{b_j}}} = \begin{cases} \frac{|c_{ai} \bigcap c_{bj}|}{|\mathcal{M}|} & \text{per } X_{c_{a_i}} = 1, X_{c_{b_j}} = 1 \\ \frac{|c_{ai}| - |c_{ai} \bigcap c_{bj}|}{|\mathcal{M}|} & \text{per } X_{c_{a_i}} = 1, X_{c_{b_j}} = 0 \\ \frac{|c_{bj}| - |c_{ai} \bigcap c_{bj}|}{|\mathcal{M}|} & \text{per } X_{c_{a_i}} = 0, X_{c_{b_j}} = 1 \\ \frac{|\mathcal{M}| - (|c_{ai} \bigcup c_{bj}| - |c_{ai} \bigcap c_{bj}|)}{|\mathcal{M}|} & \text{per } X_{c_{a_i}} = 0, X_{c_{b_j}} = 0 \end{cases}$$

Si può quindi definire quanto sono distanti tra loro $\mathcal{C}^A$ e $\mathcal{C}^B$, in termini di mancanza di informazione su $\mathcal{C}^A$ dato $\mathcal{C}^B$ e viceversa, calcolando la mancanza di informazione su $X_{c_{a_i}}$ dato $X_{c_{b_j}}$ e viceversa.

La mancanza di informazione su $X_{c_{a_i}}$ dato $X_{c_{b_j}}$ può essere calcolata tramite l'entropia condizionale:

$$H(X_{c_{a_i}}|X_{c_{b_j}}) = H(X_{c_{a_i}}, X_{c_{b_j}}) - H(X_{c_{b_j}}) \tag{A.1}$$

Fissato $X_{c_{a_i}}$ tra tutti i possibili $X_{c_{b_j}}$ (variabili casuali che modellano i cluster di $\mathcal{C}^B$), esiste $\widehat{X}_b$ tale che $H(X_{c_{a_i}}|\widehat{X}_b) \leq H(X_{c_{a_i}}|X_{c_{b_j}})$. Il valore $H(X_{c_{a_i}}|\widehat{X}_b)$ può essere normalizzato dividendolo per $H(X_{c_{a_i}})$.

Si può quindi ottenere la mancanza di informazione necessaria a descrivere $\mathcal{C}^A$ dato $\mathcal{C}^B$ come:

$$H(\mathcal{C}^A|\mathcal{C}^B) = \frac{1}{|\mathcal{C}^A|} \sum_{c_{a_i} \in \mathcal{C}^A} \frac{H(X_{c_{a_i}}|\widehat{X}_b)}{H(X_{c_{a_i}})} \tag{A.2}$$

Analogamente si può definire $H(\mathcal{C}^B|\mathcal{C}^A)$ come

$$H(\mathcal{C}^B|\mathcal{C}^A) = \frac{1}{|\mathcal{C}^B|} \sum_{c_{b_j} \in \mathcal{C}^B} \frac{H(X_{c_{b_j}}|\widehat{X}_a)}{H(X_{c_{a_i}})} \tag{A.3}$$

la distanza tra $\mathcal{C}^A$ e $\mathcal{C}^B$ è definita dalla media tra (A.2) e (A.3):

$$D(\mathcal{C}^A, \mathcal{C}^B) = \frac{1}{2}\left(\frac{1}{|\mathcal{C}^A|} \sum_{c_{ai} \in \mathcal{C}^A} \frac{H(X_{c_{a_i}}|\widehat{X}_b)}{H(X_{c_{a_i}})} + \frac{1}{|\mathcal{C}^B|} \sum_{c_{b_j} \in \mathcal{C}^B} \frac{H(X_{c_{b_j}}|\widehat{X}_a)}{H(X_{c_{b_j}})}\right) \tag{A.4}$$

sostituendo in (A.4) $\frac{H(X_{c_{a_i}}|\widehat{X}_b)}{H(X_{c_{a_i}})}$ con $\delta(c_{ai}, \mathcal{C}^B) = \min_{c_{bj} \in \mathcal{C}^B} d_H(c_{a_i}, c_{b_j}), \forall c_{b_j} \in \mathcal{C}^B$ e $\frac{H(X_{c_{b_j}}|\widehat{X}_a)}{H(X_{c_{b_j}})}$ con $\delta(c_{b_j}, \mathcal{C}^A) = \min_{c_{a_i} \in \mathcal{C}^A} d_H(c_{ai}, c_{b_j}), \forall c_{a_i} \in \mathcal{C}^A$ si riottiene la formula per il calcolo della distanza (3.1) presentata nel Capitolo 3 in Sezione 3.2.1. Il problema di $d_H(c, c')$ è che non assume il valore 0 solamente se $c \equiv c'$ ma più in generale se $X_c = \mathcal{F}(X_{c'})$ (e.g., $c \bigcap c' = \varnothing$ e $c \bigcup c' = \mathcal{M}$). Per risolvere il problema [Lancichinetti et al., 2009] pongono la seguente condizione sul calcolo di d_H[1]:

$$d_H(c, c') = \begin{cases} \frac{H(X_c|X_{c'})}{H(X_c)} & \text{se } h(P(1,1)) + h(P(0,0)) > h(P(1,0)) + h(P(0,1)) \\ 1 & \text{altrimenti} \end{cases} \tag{A.5}$$

[1] con $h(P(u,y))$ si intende $h(P(X_c = u, X_{c'} = y)$

Bibliografia

Frank Dessmann Andreas Marx. The wildlist is dead, long live the wildlist! In *Virus Bullettin conference 2007*, 2007. Available online at http://www.sunbelt-software.com/ihs/alex/vb_2007_wildlist_paper.pdf.

Ulrich Bayer, Paolo M. Comparetti, Clemens Hlauschek, Christopher Krügel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, 2009.

Murray Brand. Forensic analysis avoidance techniques of malware. paper presented at the. In *University, Mount Lawley Campus, Western Australia*, 2007.

Julio Canto, Marc Dacier, Engin Kirda, and Corrado Leita. Large scale malware collection: Lessons learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing System*, 2008.

Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium (Security'03)*, pages 169–186. USENIX Association, USENIX Association, August 2003.

Marius Gheorghescu. An automated virus classification system. 2009.

Sarah Gordon. Virus and vulnerability classification schemes: Standards and integration. Technical report, Symantec Security Response, 2003.

David Harley. The game of the name malware naming, shape shifters and sympathetic magic. In *CEET 3rd International Conference on Cybercrime Forensics Education & Training*, San Diego, CA, 2009.

infosecurity.com. Malware threat reports fail to add up. Available online at http://www.infosecurity-us.com/view/6314/malware-threat-reports-fail-to-add-up/, 2010.

Desiree Beck-Robert Martin Ivan Kirillov, Penny Chase. Malware attribute enumeration and characterization.

Tom Kelchner. The (in)consistent naming of malcode. *Computer Fraud & Security*, 2010(2):5–7, 2010. ISSN 1361-3723.

Andrea Lancichinetti, Santo Fortunato, and Janos Kertesz. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009. URL `http://stacks.iop.org/1367-2630/11/i=3/a=033015`.

David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. Available online at `http://www.inference.phy.cam.ac.uk/itprnn/book.pdf`.

Malik Magdon-Ismail Mark K.Goldberg, Mykola Hayvanovych. Measuring similarity between sets of overlapping clusters. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, Minneapolis, MN, Aug 2010.

Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *In Proceedings of the Annual Computer Security Applications Conference (ACSAC*, 2007.

Microsoft. Microsoft malware protection center glossary. Available online at `http://www.microsoft.com/security/portal/Threat/Encyclopedia/Glossary.aspx`, 2011.

David Harley Pierre-Marc Bureau. A dose by any other name. In *Virus Bullettin conference October 2008*, San Diego, CA, October 2008.

Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. Available online at `http://dutta.csc.ncsu.edu/csc791_spring07/wrap/dfs.pdf`.

VirusBullettin. Vgrep, system designed to help clear up some of the confusion surrounding the naming of viruses. Available online at `http://www.virusbtn.com/resources/vgrep/index`, 2011.

VirusTotal. Virustotal, free online virus and malware scan. Available online at `http://www.virustotal.com`, 2011.

H. Wolda. Similarity indices, sample size and diversity. *Oecologia*, 50(3): 296–302, 1981. ISSN 0029-8549.