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

ANALISI SPERIMENTALE DI CLUSTER DI MALWARE BASATI SU CARATTERISTICHE STATICHE E DINAMICHE

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

Tesi di Laurea di:
Alessia CREMONESI
Matr. 734666
Marco CRESSERI
Matr. 720843

Anno Accademico 2010–2011

Ringraziamenti

Desideriamo esprimere un sentito ringraziamento al Prof. Stefano Zanero per la sua cortesia, competenza e disponibilità che ci ha dimostrato, e per tutto l'aiuto fornito durante la stesura di questo lavoro. Ringraziamo inoltre l'Ing. Guido Salvaneschi per l'interesse e il tempo che ci ha dedicato fin dall'inizio di questo percorso. Non possiamo dimenticare l'Ing. Federico Maggi che con i suoi consigli ci ha accompagnati in questo periodo. Grazie a tutti per averci permesso di affrontare una problematica così interessante e attuale.

Un ringraziamento particolare va anche al/alla mio/a compagno/a di tesi per la sua amicizia e le avventure affrontate insieme, per tutte quelle volte che siamo usciti indenni dalle difficoltà supportandoci a vicenda e per aver alleggerito le giornate passate interamente davanti al computer.

Un immenso grazie va anche a tutte quelle persone che ci sono state vicine, sostenendoci ed incoraggiandoci durante tutti questi anni universitari.

Sommario

Il problema del malware costituisce una severa minaccia per la sicurezza dei dispositivi informatici. La categorizzazione del codice malevolo è di fondamentale importanza per riconoscere le *famiglie* di malware esistenti e per identificare le nuove minacce. Per molti anni questa operazione è stata svolta manualmente dagli analisti. Attualmente, si stima che i campioni rilevati quotidianamente siano nell'ordine delle decine di migliaia, pertanto sono necessarie tecniche automatiche di analisi e classificazione.

In questa tesi analizziamo sperimentalmente tre metodi di clustering (impiegabili per classificare campioni sconosciuti) basati su *feature dinamiche* e *statiche*, ricavando i parametri che garantiscono i risultati migliori rispetto alla classificazione per famiglie (attribuita da VirusTotal) scelta come riferimento. Proponiamo inoltre due nuove feature costruite a partire dalle informazioni ricavate tramite analisi statica, allo scopo di rappresentare la struttura del campione a differenti livelli di dettaglio. Infine confrontiamo i cluster prodotti dai vari metodi con la classificazione basata sulle famiglie e quelli generati dal metodo basato su feature dinamiche con quelli ottenuti a partire dalle feature statiche.

I risultati sperimentali evidenziano che i cluster prodotti a partire dalle feature statiche non presentano correlazioni significative con quelli basati su feature dinamiche. Una spiegazione di questo risultato è che queste ultime sono legate in maniera differente rispetto alle feature statiche, implicando così la creazione di cluster differenti.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Con il termine *malware*, che deriva dalla contrazione delle parole inglesi *malicious* e *software*, si identificano quei programmi il cui principale scopo è, in generale, causare danni più o meno gravi al computer infettato.

Internet è una parte essenziale della società moderna, ma è anche un perfetto mezzo di diffusione per malware, scritto e spesso controllato da vere e proprie organizzazioni criminali. Infatti, mentre i primi codici malevoli furono scritti come esperimento o per scherzo e avevano lo scopo di divertire piuttosto che causare danni ai sistemi di computer, lo scopo dei malware moderni è quello di arricchirne l'autore. Ad esempio, il malware *Gpcode* cifra alcune tipologie di documenti chiedendo poi alla vittima di contattare l'autore per acquistare un'applicazione in grado di ripristinare i file.

I malware si differenziano per scopo e metodo di propagazione. Le prime vittime furono i computer UNIX che, tra gli anni '70 e '80. La prima categoria di software malevoli per personal computer furono i virus. Nel lontano 1982 uno studente universitario scrisse *Elk Cloner* per i computer Apple II, probabilmente la piattaforma più diffusa del momento. Nel 1986 molti virus nacquero nelle università: si installavano nei settori di avvio del disco rigido e la loro propagazione avveniva principalmente tramite lo scambio di floppy disk infetti. Dalla metà degli anni '90 furono colpite le aziende tramite i virus in forma di macro, la cui propagazione è facilitata dalla diffusione degli applicativi per ufficio (es., le macro per Microsoft Office). Nel frattempo i metodi di propagazione si sono evoluti passando dai CD-ROM infetti alla rete Internet. In particolare, la diffusione della posta elettronica, comprensiva di allegati, favorì la moltiplicazione dei virus basati su macro. L'avvento della banda larga creò le condizioni ideali per la proliferazione dei worm, tipi di malware caratterizzati dalle capacità di auto-propagarsi. Al termine della prima decade del nuovo millennio la diffusione del Web, delle applicazioni web e, recentemente, del Web 2.0 ha spostato la superficie di attacco dagli utilizzatori di programmi per ufficio agli utenti di applicazioni web. Di conseguenza, abbiamo assistito negli ultimi tre anni ad un'esplosione dei

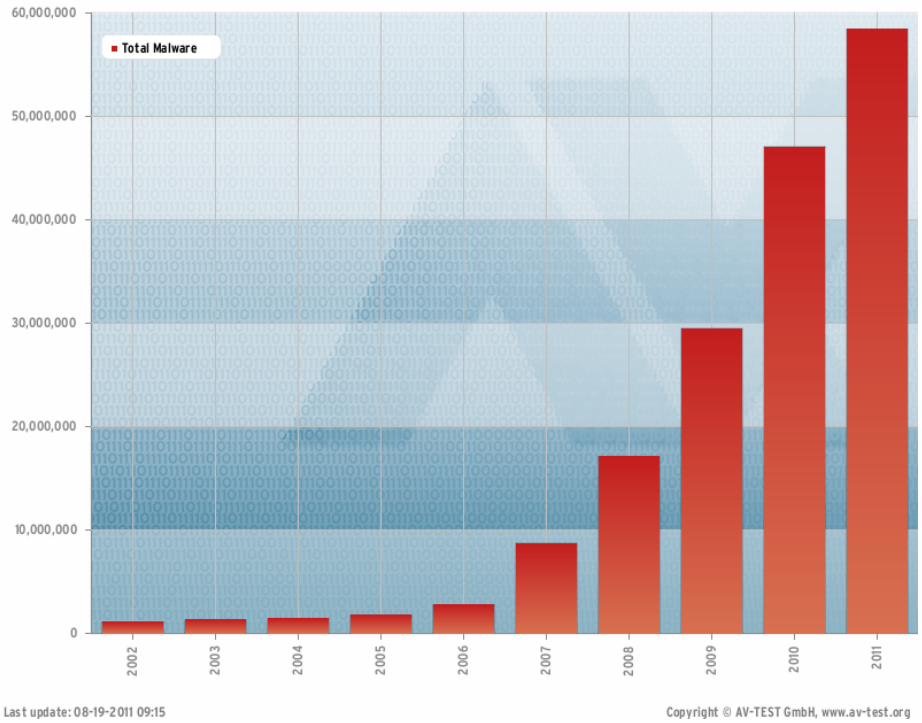

Figura 1.1: Crescita del numero dei campioni di malware rilevati tra il 2002 e il 2011. Fonte AV-TEST [1].

malware che sfruttano il Web per propagarsi. I criminali informatici iniziano a mettere a punto meccanismi di ricerca automatica di siti vulnerabili. Una volta individuati, li compromettono al fine di installarvi dei veri e propri "kit" di infezione destinati a colpire i visitatori di tali siti.

Negli anni, il numero di malware è aumentato esponenzialmente e continua a crescere, come evidenziato in Figura 1.1. A partire dal 2007 si nota un incremento dei malware e tale crescita non accenna a diminuire. Attualmente si stima che il numero di nuovi campioni rilasciati sia tra i 30.000 e i 50.000 al giorno. Il motivo di questa crescita è da ricercarsi nel maggiore utilizzo di motori polimorfici, packer, tecniche di offuscamento e ricompilazione. Pertanto, i nuovi virus individuati spesso risultano mutazioni di malware già conosciuti.

Gli analisti di malware ricevono quotidianamente migliaia di nuovi campioni, pertanto, è di fondamentale importanza determinare quali azioni possono essere eseguite da questi software malevoli. A questo scopo, durante l'analisi, i malware sono raggruppati in famiglie con caratteristiche simili e poche differenze comportamentali. Ad esempio, la famiglia *Beagle* comprende worm il cui scopo è generare spam, i campioni della famiglia *Msnpass* sottraggono dati sensibili da programmi di messaggistica istantanea, mentre

quelli della famiglia *Sinowal* consentono l'accesso a un computer remoto da parte di malintenzionati. A ciascuna famiglia è associato un nome, ed eventualmente una variante, che la identifica in modo univoco. Questa classificazione aiuta gli analisti a capire se un particolare campione è una variante di un malware noto o se si tratta di una nuova specie. Per molti anni questa classificazione è stata eseguita manualmente dagli analisti, ma il continuo aumento dei campioni da analizzare ha reso questo procedimento infattibile a causa del tempo richiesto e delle risorse necessarie. Sono state così sviluppate tecniche di classificazione automatica che permettono di raggruppare i campioni in famiglie, in modo da supportare le fasi di analisi. Tuttavia, quelle che consentono di ottenere dei buoni risultati non sempre sono scalabili in quanto non sono in grado di gestire dataset di grandi dimensioni.

In questa tesi ci occupiamo di indagare la possibilità di classificare i malware in base alle loro caratteristiche comportamentali e strutturali. I campioni sono sottoposti ad analisi dinamica e statica per ottenere una serie di *feature*, che identificano particolari caratteristiche presenti o meno nei campioni analizzati, come, ad esempio, l'azione "creazione del processo X" o la presenza di una determinata sequenza di istruzioni nel codice. A questo scopo effettuiamo una completa analisi sperimentale utilizzando:

- tre metodi di clustering gerarchico tradizionale, di tipo agglomerativo, basati su quattro definizioni di distanza adatte al confronto di vettori binari;
- un metodo probabilistico basato su hash, utilizzato in letteratura per il clustering di malware basato su modelli di comportamento;
- un metodo utilizzato nell'ambito dell'information retrieval[1] per la classificazione e la ricerca di documenti.

Nella prima parte ci focalizziamo sulle *feature dinamiche* (es., "scrittura sul file X" e "apertura del socket Y"). Sottoponiamo ciascun campione ad analisi dinamica per identificare azioni eseguite e comportamenti manifestati. Costruiamo quindi un insieme di cluster basandoci su tali caratteristiche e ne valutiamo la qualità confrontandolo con il clustering di riferimento corrispondente alle famiglie.

Nella parte centrale ci focalizziamo sull'analisi statica proponendo due nuove *feature statiche*, derivanti dai basic block e dal control flow graph, al fine di migliorare i risultati forniti dagli algoritmi di clustering. Confrontiamo quindi tra loro le classificazioni ottenute, cercando i parametri che consentono di ottenere la migliore qualità dei risultati, e le paragoniamo al clustering di riferimento corrispondente alle famiglie. Infine analizziamo come queste feature si adattano al contesto della classificazione.

[1] Insieme di tecniche utilizzate per il recupero mirato delle informazioni in formato elettronico.

Nell'ultima parte confrontiamo sperimentalmente i risultati del clustering basato su feature dinamiche con quelli ottenuti a partire dalle feature statiche, cercando una possibile relazione ed evidenziandone le inconsistenze.

I risultati descritti in questa tesi, ricavati sperimentalmente, evidenziano che gli algoritmi di clustering tradizionali non sono in grado di operare con l'elevato numero di feature. L'algoritmo probabilistico basato su hash risente dell'eccessiva specificità delle caratteristiche non riuscendo a fornire una copertura significativa. L'ultimo metodo applicato, invece, fornisce risultati migliori dei precedenti anche se presenta delle inconsistenze rispetto la classificazione delle famiglie presa come riferimento. Altresì, dai risultati non è possibile evidenziare alcuna relazione significativa che permetta di legare il clustering basato su feature dinamiche con quelli prodotti dalle feature statiche.

Capitolo 2

Definizione del problema

Il problema del malware costituisce una severa minaccia per la sicurezza dei dispositivi informatici. Conseguentemente molte comunità di ricerca hanno focalizzato la loro attenzione su quest'area e hanno proposto numerose tecniche per analizzare e classificare il malware.

La classificazione dei malware consiste nel raggruppamento di campioni simili in uno stesso insieme. La comunità ha proposto il concetto di *famiglie* per identificare i campioni di malware che presentano caratteristiche comuni. Una famiglia di malware è un gruppo di campioni molto simili, solitamente nati da piccole variazioni dello stesso frammento di codice sorgente, caratterizzati da comportamenti analoghi. Piccole variazioni nel comportamento tra due varianti possono causare differenze considerevoli nella loro classificazione. Ad esempio, alcune varianti di tipo virus infettano file, mentre altre di tipo worm si auto-replicano. Tale concetto, però, non ha una definizione precisa, di conseguenza, ad una stessa famiglia, possono appartenere campioni di tipologia diversa.

I prodotti antivirus solitamente assegnano lo stesso nome ai membri della stessa famiglia, contrassegnando le varianti con un codice che segue il nome. Tuttavia, i criteri di assegnazione dei nomi non sono uniformi e mostrano delle inconsistenze tra i vari antivirus. I malware possono anche essere classificati sulla base di caratteristiche proprie quali feature dinamiche e statiche. Tutti questi metodi di classificazione sono indipendenti e producono risultati differenti, pertanto trovare una relazione tra di essi è un problema di non facile soluzione.

L'obiettivo principale di questo lavoro consiste nella verifica sperimentale della possibilità di classificare i campioni di malware in base alle proprie feature dinamiche e statiche, analizzandone le relazioni con le famiglie identificate dai principali antivirus. Purtroppo, non sempre tutti gli antivirus rilevano i campioni come appartenenti alla stessa famiglia, pertanto può capitare che un campione sia attribuito ad una famiglia errata. In secondo luogo, si indaga sperimentalmente l'esistenza di una relazione tra la

classificazione prodotta a partire dall'analisi dinamica con quelle ottenute basandosi sull'analisi statica.

In questo capitolo, prima di presentare una classificazione prodotta da alcuni antivirus (Sezione 2.2) e spiegare come abbiamo costruito il dataset (Sezione 2.3), presentiamo una panoramica sui principali tipi di malware esistenti (Sezione 2.1) con lo scopo di facilitare il lettore.

2.1 Tipi di malware

Il codice malevolo è solitamente classificato in base ai suoi obiettivi e al metodo di propagazione [2]. Presentiamo di seguito alcuni dei tipi di malware più diffusi. Vista la rapida evoluzione del software malevolo la classificazione non è da ritenersi esaustiva.

Virus programmi che si autopropagano infettando altri file, di solito eseguibili ma anche altri file con macro, o particolari sezioni del disco fisso, in modo da essere eseguiti ogni volta che il file infetto viene aperto. Si trasmettono da un computer all'altro tramite spostamento dei file infetti da parte dell'utente.

- **Batch** sono i cosiddetti "virus amatoriali". Si tratta in generale di file batch (es., `autoexec.bat` o `/etc/rc.local`) contenenti istruzioni dannose (es., comandi di formattazione o cancellazione di file). Normalmente gli antivirus non rilevano i file batch come dannosi. L'uso di questo tipo di "virus" è spesso ricorrente nel cyberbullismo.
- **Exploit** sono frammenti di codice malevolo o dati costruiti appositamente per sfruttare una specifica vulnerabilità del sistema bersaglio, con lo scopo di causare danni al sistema stesso o prenderne possesso, eventualmente per scatenare ulteriori attacchi di tipo *denial-of-service*.

Worm programmi che si propagano automaticamente da un computer all'altro, modificando la configurazione del sistema operativo della macchina ospite in modo da essere eseguiti automaticamente e tentano di replicarsi sfruttando principalmente Internet. Per indurre gli utenti ad eseguirli utilizzano tecniche di persuasione tipiche del *social engineering* [3], oppure sfruttano delle vulnerabilità di alcuni programmi per diffondersi automaticamente. Il loro scopo è rallentare il sistema con operazioni inutili o dannose.

- **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.

Rabbit programmi che esauriscono le risorse del computer creando rapidamente copie di se stessi (in memoria o su disco).

Keylogger sono programmi in grado di registrare tutto ciò che un utente digita su una tastiera o negli appunti, rendendo così possibile il furto di password o dati potenzialmente sensibili. Generalmente i keylogger sono installati sul computer attraverso *trojan* o *worm*, in altri casi invece è necessario l'accesso fisico o remoto al computer della vittima.

Browser modifier programma che modifica le impostazioni del browser, come la pagina iniziale, senza il consenso dell'utente.

Trojan horse software che, oltre ad esporre funzionalità "lecite" per indurne l'utilizzo, contengono parti dannose in grado di attaccare il sistema o sottrarre dati che vengono eseguite all'insaputa dell'utilizzatore. Non sono in grado di replicarsi automaticamente, quindi per diffondersi devono essere consapevolmente inviati alla vittima. Oggi i trojan horse sono impiegati per installare altri malware, come ad esempio keylogger, backdoor. Sono infatti presenti alcune tipologie di trojan detti *dropper/downloader* che hanno lo scopo di installare altro codice malevolo, oppure come *proxy* che permettono di configurare il computer infettato in modo da rendere possibile l'intercettazione del traffico, l'impersonificazione, e-banking fraudolento.

Dialer programma per stabilire una connessione ad Internet. Utilizzati in modo illecito, i dialer permettono di modificare il numero telefonico composto dal modem. I numeri impiegati a scopo illecito sono numeri a tariffazione speciale, allo scopo di trarne illecito profitto all'insaputa dell'utente.

Scareware sono così chiamati quei programmi che ingannano l'utente facendo credere di avere rilevato gravi problemi sul computer, allo scopo di far loro installare particolari malware, chiamati in gergo *rogue antivirus*, caratterizzati dal fatto di spacciarsi per antivirus veri e propri o tool di sicurezza, talvolta innocui ma a pagamento.

Rogue security software software che appare utile per migliorare la sicurezza del sistema, generando una serie di allarmi erronei e ingannevoli, e che tramite social engineering fa partecipare l'utente a transazioni fraudolente. Ad esempio, malware che si fingono programmi per la sicurezza del pc, spingendo gli utenti ad acquistare una licenza dello stesso [4].

Backdoor programmi che consentono l'accesso non autorizzato al sistema su cui sono in esecuzione da parte di entità esterne. Solitamente modificano le politiche di sicurezza locali in modo da permettere il completo accesso remoto tramite la rete. Tipicamente si diffondono

in abbinamento ad un *trojan* o ad un *worm*, oppure costituiscono una forma di accesso di emergenza al sistema per permettere, ad esempio, il recupero di una password dimenticata.

Spyware software usato per raccogliere informazioni dall'utente e dal sistema per poi trasmetterle a un destinatario interessato. Le informazioni carpite possono andare dalle abitudini di navigazione, siti web visitati, fino alle password e alle chiavi crittografiche. 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.

Rootkit solitamente sono composti da un driver (kernel space) e, a volte, da alcune copie modificate di programmi normalmente presenti nel sistema. Non sono dannosi di per sé, ma hanno la funzione di nascondere, sia all'utente o all'amministratore del sistema che a programmi tipo antivirus la presenza di particolari processi, file, funzioni o impostazioni dannosi. Sono principalmente utilizzati per mascherare *spyware* e *trojan*.

Adware programmi che visualizzano pubblicità. Mentre alcuni possono essere utili in quanto permettono di sponsorizzare un servizio, altri forniscono una visualizzazione continua di messaggi pubblicitari senza il consenso dell'utente. Possono causare danni quali rallentamenti del pc e rischi per la privacy in quanto comunicano le abitudini di navigazione ad un server remoto.

Le categorie elencate, oltre a presentare svariate sottocategorie, sono tra loro combinabili, ad esempio un *worm* può contenere un payload che installa una *backdoor*, permettendo così la creazione di una rete di computer accessibile in maniera remota, dove è possibile lanciare attacchi di *distributed denial-of-service* (DDoS) o altro tipo. La Figura 2.1 mostra come si suddividono i codici malevoli nelle differenti categorie.

2.2 Naming

Quando un analista di malware rileva un file come malevolo vi associa un'etichetta che 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.

Il contenuto e la struttura delle etichette sono fortemente dipendenti dalle convenzioni adottate dall'antivirus che le ha generate. Per uno stesso malware, antivirus differenti, possono generare etichette diverse con strutture ed informazioni differenti. Alcune etichette possono contenere dati riguardanti il tipo di codice malevolo (es., worm, trojan, backdoor), la piattaforma per

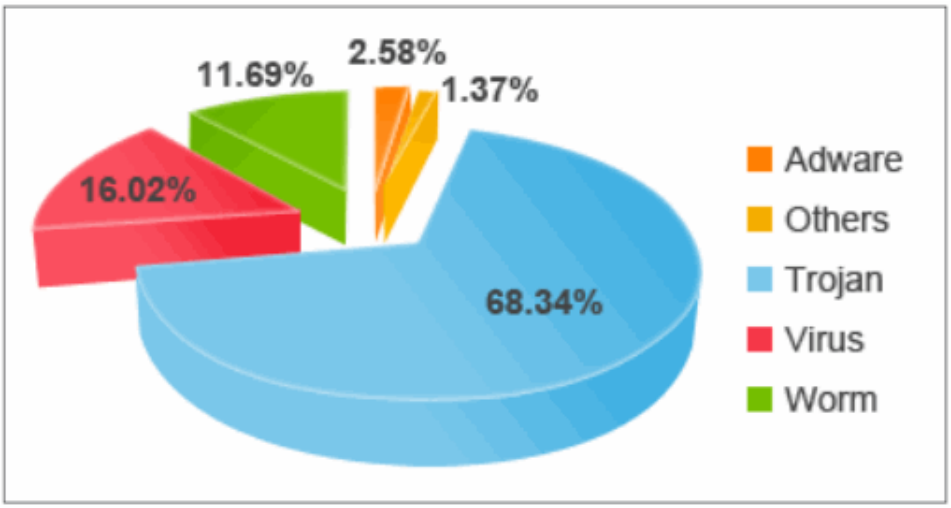

Figura 2.1: Suddivisione del malware per categorie (Aprile - Giugno 2011). Fonte PandaLabs [5].

cui il malware è stato progettato (es., Win32, VBS, JS) oltre che al nome o versione di codice.

Non è mai stata seguita o stabilita un'unica convenzione per l'assegnazione dei nomi ai malware. Ricercatori e produttori di antivirus erano soliti assegnare i nomi in base a caratteristiche ritenute interessanti. Per l'utente medio, conoscere il nome che il proprio antivirus assegna al campione in esame risulta di poca rilevanza, infatti, nel caso volesse informazioni più dettagliate, può consultare gli archivi gestiti dalla quasi totalità dei produttori di antivirus. L'assegnazione dei nomi, però, risulta un vero problema quando si cerca di correlare dati forniti da antivirus differenti.

Il sempre maggior numero di malware rilevati ogni giorno e l'utilizzo di tecniche di offuscamento ha costretto i principali produttori di antivirus ad evolvere i propri strumenti. Le tecniche di identificazione esatte sono diventate inutilizzabili poiché il tempo necessario per effettuare la scansione di ogni singolo campione ha reso il processo infattibile. Ha così preso piede l'utilizzo di *signature* generiche ed euristiche avanzate, potenzialmente diverse tra i vari produttori, per riconoscere intere famiglie di malware.

Ogni antivirus ha diverse capacità di rilevazione dei codici malevoli oltre che differenze nell'assegnazione dei nomi e ne è stata recentemente dimostrata l'inconsistenza dei metodi di classificazione [6]. È quindi uno sforzo inutile cercare di confrontarne i risultati.

Non sono immuni da questo problema i servizi, quale VirusTotal [7], che consentono di eseguire parallelamente i differenti tipi di antivirus su un campione di malware. Anche in questo caso cercare una correlazione tra i vari report diventa un problema di non facile soluzione. Pertanto la

classificazione del malware effettuata basandosi sulle etichette è alquanto disomogenea ed inconsistente.

2.3 Generazione del dataset

Per svolgere le analisi sperimentali presentate in questo lavoro di tesi abbiamo generato un dataset con l'ausilio del portale VirusTotal. Questo strumento consente di analizzare un file ritenuto sospetto con i principali 42 antivirus. Inoltre offre la possibilità, a particolari tipologie di utenti, di eseguire ricerche, visualizzare statistiche e scaricare campioni che soddisfino i criteri di ricerca impostati.

I malware che compongono il dataset sono quelli più diffusi appartenenti alle famiglie *Ackantta*, *Beagle* [8], *Conficker* [9, 10], *Fujacks* [11], *Msnpass*, *MyDoom*, *SillyFDC*, *Sinowal*, *Virut* e *ZBot*. La famiglia attribuita da VirusTotal ad un campione corrisponde a quella rilevata dalla maggioranza degli antivirus. Tuttavia, i vari antivirus potrebbero non essere concordi sul nome della famiglia come dimostrato in [6].

Nel presente lavoro utilizziamo due strumenti, ANUBIS [12] e DISASM [13], che eseguono rispettivamente analisi dinamica e statica. Il secondo, come qualsiasi altro metodo di analisi statica, non è in grado di processare software packed, poiché il codice del campione è cifrato o compresso e quindi non accessibile fino all'istante di esecuzione. Per tale motivo abbiamo ristretto la ricerca ai soli campioni privi di packer o elaborati con UPX [14], di cui è liberamente disponibile l'unpacker. Il dataset costruito comprende, in totale, 934 campioni divisi nelle 10 famiglie prima elencate. Nella Tabella 2.1 evidenziamo la composizione dettagliata indicando anche il numero di campioni di ogni famiglia.

Famiglia	Unpacked	UPX	Totali
Ackantta	140	2	142
Beagle	52	26	78
Conficker	43	74	117
Fujacks	12	59	71
Msnpass	41	19	60
MyDoom	31	62	93
SillyFDC	78	3	81
Sinowal	100	0	100
Virut	95	3	98
ZBot	91	3	94
Totali	683	251	934

Tabella 2.1: Composizione del dataset

Nonostante gli accorgimenti usati durante la costruzione del dataset, alcuni campioni hanno causato errori durante l'analisi con ANUBIS e DISASM. Tali errori possono essere dovuti a svariati motivi, tra cui la presenza di file corrotti, packed in modo ricorsivo o con un algoritmo non rilevato. La Tabella 2.2 riassume il dataset riportando il numero di campioni validi e gli errori riscontrati. Escludiamo dalle successive analisi tutti i malware che non dispongono di entrambi i report prodotti dai due strumenti. Di conseguenza i campioni su cui eseguiamo le analisi sono quelli riportati nella colonna *validi*.

Famiglia	Validi	Errori ANUBIS	Errori DISASM	Errori comuni	Totali
Ackantta	139	3	0	0	142
Beagle	47	7	22	2	78
Conficker	107	3	7	0	117
Fujacks	63	1	7	0	71
Msnpass	57	2	0	1	60
MyDoom	81	2	0	10	93
SillyFDC	64	6	8	3	81
Sinowal	97	3	0	0	100
Virut	92	4	2	0	98
ZBot	89	3	1	1	94
Totali	836	34	47	17	934

Tabella 2.2: Errori nel dataset

La Tabella 2.3 riporta i cluster prodotti da alcuni dei principali antivirus impiegati da VirusTotal. La *copertura*, in questo caso, indica la percentuale di campioni appartenenti alla famiglia, attribuita da VirusTotal, che sono stati rilevati dall'antivirus; nella colonna *cluster* si indica, invece, il numero di cluster in cui si divide la famiglia considerata. La bassa copertura rilevata per alcune famiglie (es., Beagle e Sinowal) evidenzia la difficoltà di rilevare l'appartenenza o meno di un campione alla famiglia considerata.

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	Avast5		Ikarus	
Ackantta	95,77	8	93,66	13
Beagle	32,05	9	42,31	7
Conficker	87,18	7	80,34	8
Fujacks	54,93	7	74,65	8
Msnpass	58,33	9	48,33	11
MyDoom	82,80	4	82,80	1
SillyFDC	55,56	10	51,85	14
Sinowal	46,00	9	46,00	4
Virut	90,82	2	91,84	5
ZBot	91,49	7	88,30	2

	Microsoft		Panda	
Ackantta	87,32	13	90,14	10
Beagle	29,49	8	35,90	11
Conficker	87,18	5	88,03	8
Fujacks	50,70	6	59,15	9
Msnpass	38,33	6	68,33	7
MyDoom	82,80	3	84,95	5
SillyFDC	38,27	7	61,73	10
Sinowal	42,00	6	53,00	5
Virut	89,80	6	86,73	2
ZBot	93,62	3	82,98	5

	PcTools		Symantec	
Ackantta	95,07	4	90,14	6
Beagle	47,44	7	34,62	7
Conficker	86,32	9	88,03	7
Fujacks	60,56	7	57,75	7
Msnpass	53,33	11	58,33	10
MyDoom	83,87	2	81,72	4
SillyFDC	79,01	2	79,01	1
Sinowal	44,00	4	43,00	4
Virut	93,88	2	92,86	4
ZBot	86,17	2	84,04	4

Tabella 2.3: Clustering prodotto da alcuni dei principali antivirus impiegati da VirusTotal

Capitolo 3

Clustering basato su feature dinamiche

Esistono vari modi per analizzare i malware. In questo capitolo ci concentriamo sulle tecniche di analisi dinamica, in particolare consideriamo come feature dinamiche le azioni eseguite dai campioni e registrate da ANUBIS. Descriviamo e applichiamo un metodo di clustering basato su tali caratteristiche e ne analizziamo i risultati.

Dopo aver descritto le caratteristiche dell'analisi dinamica (Sezione 3.1) dettagliamo il metodo basato sulle feature dinamiche (Sezione 3.2), descrivendo la costruzione e riduzione della matrice dei dati e gli algoritmi di clustering impiegati, e infine analizziamo i risultati prodotti (Sezione 3.3) valutandone le inconsistenze rispetto alle famiglie.

3.1 Analisi dinamica

L'analisi dinamica dei programmi consiste nell'esecuzione del software in una *sandbox*, un ambiente isolato e monitorato. Durante l'esecuzione vengono registrate varie attività tra cui le chiamate al sistema operativo, il traffico di rete, l'accesso al file system e al registro, ecc. Questa tecnica fornisce risultati di buona qualità anche in presenza di codice realizzato con metodi atti a complicare o impedire l'analisi statica. La principale limitazione del metodo è la dipendenza dalla traccia di esecuzione. I risultati sono basati sui comportamenti registrati durante una o più esecuzioni. Sfortunatamente, alcune attività del campione potrebbero manifestarsi solo in specifiche condizioni, ad esempio:

- le *bombe logiche* mostrano il loro comportamento malevolo soltanto in particolari istanti temporali;
- malware destinati al furto d'identità o di dati sensibili possono eseguire determinate azioni quando interagiscono con l'utente;

- altri potrebbero attivarsi in seguito alla ricezione di un determinato comando o alla presenza di alcuni oggetti all'interno del file system.

Dato che i sistemi di analisi dinamica eseguono automaticamente il codice, in un ambiente preimpostato e senza l'intervento dell'utente, questi comportamenti non verranno rilevati. Inoltre, alcuni malware sono in grado di identificare l'ambiente di esecuzione e di conseguenza modificare il proprio comportamento.

Queste limitazioni potrebbero portare ad un'incompleta panoramica dell'attività del campione.

In un recente lavoro di analisi [15] sono stati osservati alcuni comportamenti tipici di un insieme di malware, individuando alcune attività che caratterizzano azioni malevole:

Attività su file system i malware possono interagire con il file system, creando o modificando file e directory. Spesso creano copie polimorfiche di se stessi, altri file eseguibili, librerie DLL e script.

Attività su registro i malware possono creare e modificare le chiavi di registro di Windows. Tali modifiche solitamente alterano i parametri di configurazione delle schede di rete, le impostazioni di alcuni componenti software e di sistema, consentono al malware di essere eseguito automaticamente all'avvio di Windows. Inoltre, modificando le *security policy*, sono in grado di impedire all'utente di agire sulle impostazioni di sicurezza e, in generale, di avere il pieno controllo sul sistema operativo.

Attività di rete i malware possono scaricare da internet altri file, effettuare richieste a server DNS, connettersi a server remoti per inviare dati sensibili rubati all'utente, tentare di propagarsi sfruttando alcune vulnerabilità delle macchine raggiungibili, attendere comandi e connessioni.

Interazione con utenti molti malware mostrano all'utente alcuni messaggi, principalmente segnalando presunti errori, con lo scopo di ridurre i sospetti sull'attività del software malevolo.

Identificazione della sandbox alcuni malware sono in grado di stabilire se sono in esecuzione in ambienti di analisi dinamica, modificando il proprio comportamento o terminando le attività.

3.1.1 ANUBIS

ANUBIS [12, 16, 17] è uno strumento che consente di analizzare il comportamento dei file eseguibili di Windows in formato PE (Portable Executable) [18], focalizzandosi principalmente sull'analisi di malware. È basato su

una versione opportunamente modificata dell'emulatore QEMU [19] su cui è installato Windows XP Service Pack 2. Una volta lanciato l'eseguibile, ANUBIS registra tutte le azioni svolte dal software malevolo nell'ambiente emulato. Per poter valutare il comportamento del campione, vengono monitorate tutte le chiamate di sistema alle API e quelle native di Windows, contenute nella libreria *ntdll.dll*.

Solitamente i programmatori non utilizzano direttamente le chiamate di sistema native, poiché non sono completamente documentate e possono cambiare tra versioni differenti del sistema operativo. Al loro posto usano le Windows API, documentate e presenti in tutte le versioni. Gli autori di malware, invece, spesso impiegano le chiamate di sistema native al fine di eliminare le dipendenze dalle librerie DLL e confondere così gli antivirus.

Al termine dell'analisi, ANUBIS genera un file di report che contiene sufficienti informazioni da fornire all'utente una buona panoramica circa lo scopo e le azioni effettuate dal campione analizzato. Tale documento include i seguenti dettagli:

Informazioni generali questa sezione contiene informazioni circa l'invocazione di TTAnalyze [20], il motore interno ad ANUBIS, gli argomenti della riga di comando ed alcune informazioni generali riguardanti il campione in esame (es., dimensione del file, valore di ritorno del programma, tempo di esecuzione);

Attività su file questa sezione copre le attività sui file da parte del campione in esame (es., file creati, modificati, cancellati);

Attività di registro questa sezione descrive tutte le modifiche apportate al registro di Windows e tutti i valori delle chiavi letti dal campione in esame;

Attività di servizio questa sezione documenta tutte le interazioni tra il campione in esame ed il Windows Service Manager (es., avvio o interruzione di un servizio di Windows);

Attività di processo questa sezione contiene tutte le informazioni circa la creazione o terminazione di processi (e thread) e la comunicazione tra processi;

Attività di rete questa sezione fornisce un collegamento ad un file di log contenente tutto il traffico inviato o ricevuto dal campione in esame.

3.2 Uso di feature dinamiche

Questo metodo ha l'obiettivo di costruire un clustering di malware basato sulle attività svolte durante l'esecuzione e si articola nelle seguenti fasi:

Fase 1 sottoponiamo l'intero dataset costruito ad ANUBIS e otteniamo i report contenenti le informazioni comportamentali, in formato XML, relative ad ogni campione analizzato, che ne descrivono nel dettaglio tutte le attività svolte. Trascuriamo nel prosieguo dell'analisi i file che generano degli errori, e che conseguentemente non dispongono di nessun report.

Fase 2 elaboriamo i report estraendo le informazioni essenziali e costruiamo la matrice dei dati. Questo processo è dettagliato nelle Sezioni 3.2.1, 3.2.2.

Fase 3 costruiamo il clustering partendo dalla matrice prodotta al passo precedente. Questa fase è dettagliata nelle Sezioni 3.2.3 e 3.2.4.

3.2.1 Costruzione della matrice dei dati

Dobbiamo elaborare il report XML, ottenuto nella fase precedente, per rimuovere tutte le informazioni non rilevanti dal punto di vista comportamentale, quali le statistiche sull'esito dell'analisi dinamica, e quelle ridondanti o trascurabili. Selezioniamo i tag e gli attributi interessanti tramite una trasformazione XSLT, appositamente definita per estrarre da ogni elemento una stringa che descrive in modo completo una singola azione manifestata dal campione. Per ciascuna stringa calcoliamo un hash, utilizzato in seguito per identificarla in maniera più compatta. Un campione presenta la *feature dinamica* j, se la relativa azione è stata rilevata da ANUBIS. Per ciascun malware ne registriamo le occorrenze.

Raccogliamo tutti i dati in una *matrice sparsa binaria campioni-caratteristiche* $\mathbf{X_{m,n}}$, dove m è il numero dei campioni analizzati e n è il numero delle feature dinamiche estratte dalla trasformazione XSLT.

$$\mathbf{X_{m,n}} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,n} \end{bmatrix} \tag{3.1}$$

L'elemento $x_{i,j}$ assume valore 1 solo se il campione i presenta la caratteristica j, 0 altrimenti. La matrice risultante contiene 836 righe, 14.995 colonne e presenta un fattore di riempimento pari all'$1,8430\%$.

3.2.2 Riduzione delle dimensioni della matrice dei dati

Analizzando la matrice campioni-caratteristiche, osserviamo che la maggior parte delle colonne contengono un solo elemento, come evidenziato dall'istogramma a sinistra della Figura 3.1. Ciò significa che esistono malware

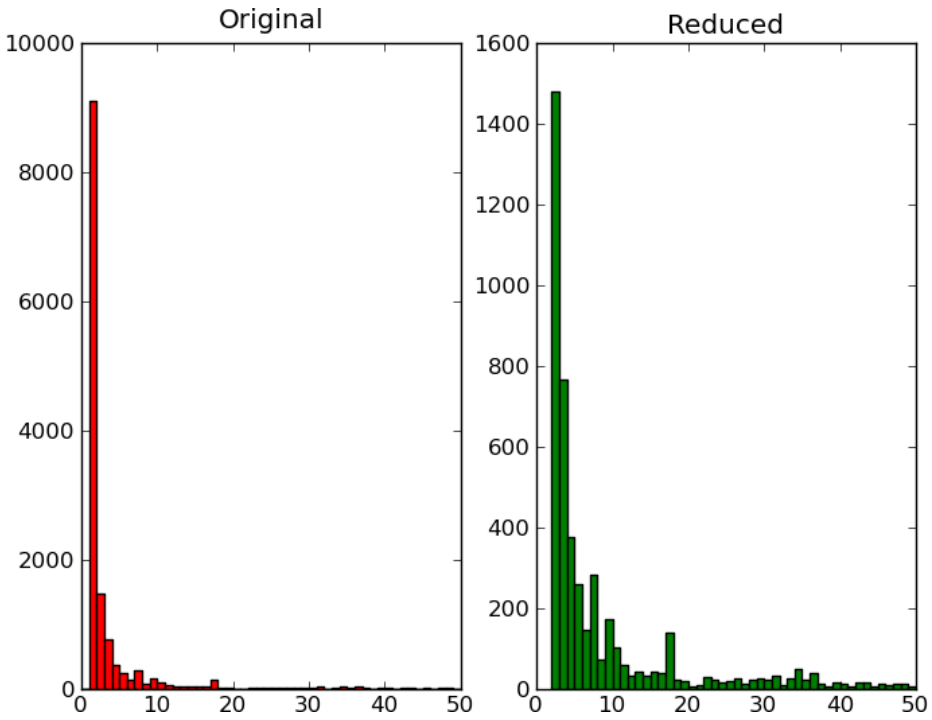

Figura 3.1: Istogrammi raffiguranti il numero di feature dinamiche presenti in un numero di campioni pari al valore dell'ascissa. In altre parole, ogni barretta dell'istogramma rappresenta il numero di colonne della matrice campioni-caratteristiche con un numero di elementi non nulli pari all'indice dell'ascissa della barretta nell'istogramma. I diagrammi sono relativi alle matrici campioni-caratteristiche, originale a sinistra e ridotta a destra.

che presentano feature dinamiche univoche, ossia eseguono azioni che li contraddistinguono dagli altri.

L'informazione univoca può essere utile per discriminare due campioni molto simili. Tali campioni infatti potrebbero differenziarsi solo per le feature dinamiche univoche e condividere le altre. Eliminando questi dettagli tali malware risulterebbero identici. Un esempio di questo problema si può verificare quando alcuni campioni scrivono su un file il cui nome viene generato casualmente, oppure quando si cerca di stabilire connessioni a server con indirizzi IP differenti. Ciascuna variante porta alla generazione di una nuova feature dinamica. Per contro, trascurando le feature dinamiche univoche si riducono notevolmente le dimensioni della matrice campioni-caratteristiche prima descritta.

A partire dalla matrice originale $\mathbf{X_{m,n}}$ costruiamo la ridotta $\mathbf{X'_{m,p}}$, con $p < n$, eliminando tutte le feature dinamiche univoche e mantenendo quelle condivise da almeno due campioni.

$$\mathbf{X'_{m,p}} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,p} \end{bmatrix} \tag{3.2}$$

Questa nuova matrice contiene lo stesso numero di righe di quella originale, 5.896 colonne e presenta un fattore di riempimento pari al $4,5010\%$.

3.2.3 Algoritmi di clustering tradizionali

Applicando gli algoritmi tradizionali di clustering gerarchico di tipo agglomerativo, sulle matrici completa e ridotta, ed utilizzando le distanze Jaccard (A.1), Coseno (A.2), Tanimoto (A.3) e Dice (A.4) otteniamo risultati non indicativi.

L'elevato numero di colonne della matrice campioni-caratteristiche rientra nella casistica dei dati ad alta dimensionalità dettagliato nell'Appendice A.3. Inoltre i tempi di costruzione della matrice delle distanze risultano proibitivi, poiché devono essere calcolate le distanze tra tutte le coppie di vettori riga.

3.2.4 Algoritmi LSH e LSI

Gli algoritmi LSH e LSI, descritti rispettivamente nell'Appendice A.3.1 e A.3.2, nascono nell'ambiente dell'*information retrieval* e hanno vari settori applicativi, tra cui il clustering di dati di grandi dimensioni.

Applichiamo entrambi gli algoritmi sia alla matrice completa, sia a quella ridotta, con lo scopo di compattarne le dimensioni e consentire, in seguito, l'applicazione del metodo di clustering gerarchico tradizionale.

Metodo basato sull'algoritmo LSH

Il primo passo di questo metodo consiste nel calcolo dei *minhash* corrispondenti a ciascun campione. Per poter generare le funzioni di hash lineari richieste è necessario calcolare il più piccolo numero primo maggiore del numero di colonne della matrice campioni-caratteristiche. Nel caso di quella completa questo numero è 15.013, mentre per quella ridotta è 5.897.

Il secondo passo consiste nel calcolo probabilistico della similarità tra i campioni. In un recente lavoro [21] sono stati calcolati i valori ottimali dei parametri coinvolti nell'algoritmo: soglia di similarità $T_s = 0,7$, numero di funzioni minhash per ogni hash lsh $k = 10$ e numero di iterazioni $l = 90$.

Infine, applicando l'algoritmo di clustering gerarchico tradizionale con criterio di collegamento *single linkage* (A.5) otteniamo i cluster.

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	Matrice completa		Matrice ridotta	
Ackantta	20,14	8	33,09	14
Beagle	8,51	2	34,04	6
Conficker	0	0	84,11	11
Fujacks	0	0	9,52	2
Msnpass	22,81	5	29,82	7
MyDoom	0	0	0	0
SillyFDC	3,13	1	25,00	7
Sinowal	2,06	1	75,26	7
Virut	0	0	17,39	2
ZBot	75,28	1	91,01	4

Tabella 3.1: Clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSH

L'algoritmo LSH fornisce risultati significativi solamente utilizzando la matrice ridotta, poiché in quella completa la maggior parte delle feature dinamiche non aiutano a trovare campioni che si comportano in modo simile. LSH seleziona casualmente e confronta alcuni vettori colonna. Due campioni ritenuti distanti sono attribuiti a cluster differenti. Nel seguito si concentra l'attenzione sul clustering ottenuto a partire dalla matrice ridotta.

I risultati dell'esecuzione di LSH sono mostrati in Figura 3.2 e presentati in Tabella 3.1. La *copertura* indica la percentuale di campioni appartenenti alla famiglia, attribuita da VirusTotal, che sono stati aggregati in cluster non unitari. Tali cluster possono includere anche campioni appartenenti a famiglie differenti. La copertura è definita in (3.3), dove $\mathcal{F}$ è la famiglia considerata e $C_i^{\mathcal{F}}$ sono i cluster generati dall'algoritmo contenenti almeno un campione della famiglia $\mathcal{F}$.

$$Copertura = \frac{\sum_i |C_i^{\mathcal{F}}|}{|\mathcal{F}|} \cdot 100 \quad t.c. \quad |C_i^{\mathcal{F}}| > 1 \tag{3.3}$$

Questo valore ci permette di capire la frazione del dataset che può essere rilevata e classificata dall'algoritmo. Trascuriamo i cluster unitari poiché non sono da ritenersi significativi. Nella colonna *cluster* si indica, invece, il numero di cluster in cui si divide la famiglia considerata. Riportiamo solamente i cluster contenenti due o più elementi.

Questo algoritmo, essendo probabilistico, applicato al contesto delle feature dinamiche, produce risultati molto differenti tra loro ad ogni esecuzione.

Analizzando nel dettaglio i cluster generati osserviamo che LSH produce una pluralità di cluster spuri, ossia contenenti un singolo elemento, ed un elevato numero cluster piccoli. Inoltre la copertura è piuttosto bassa.

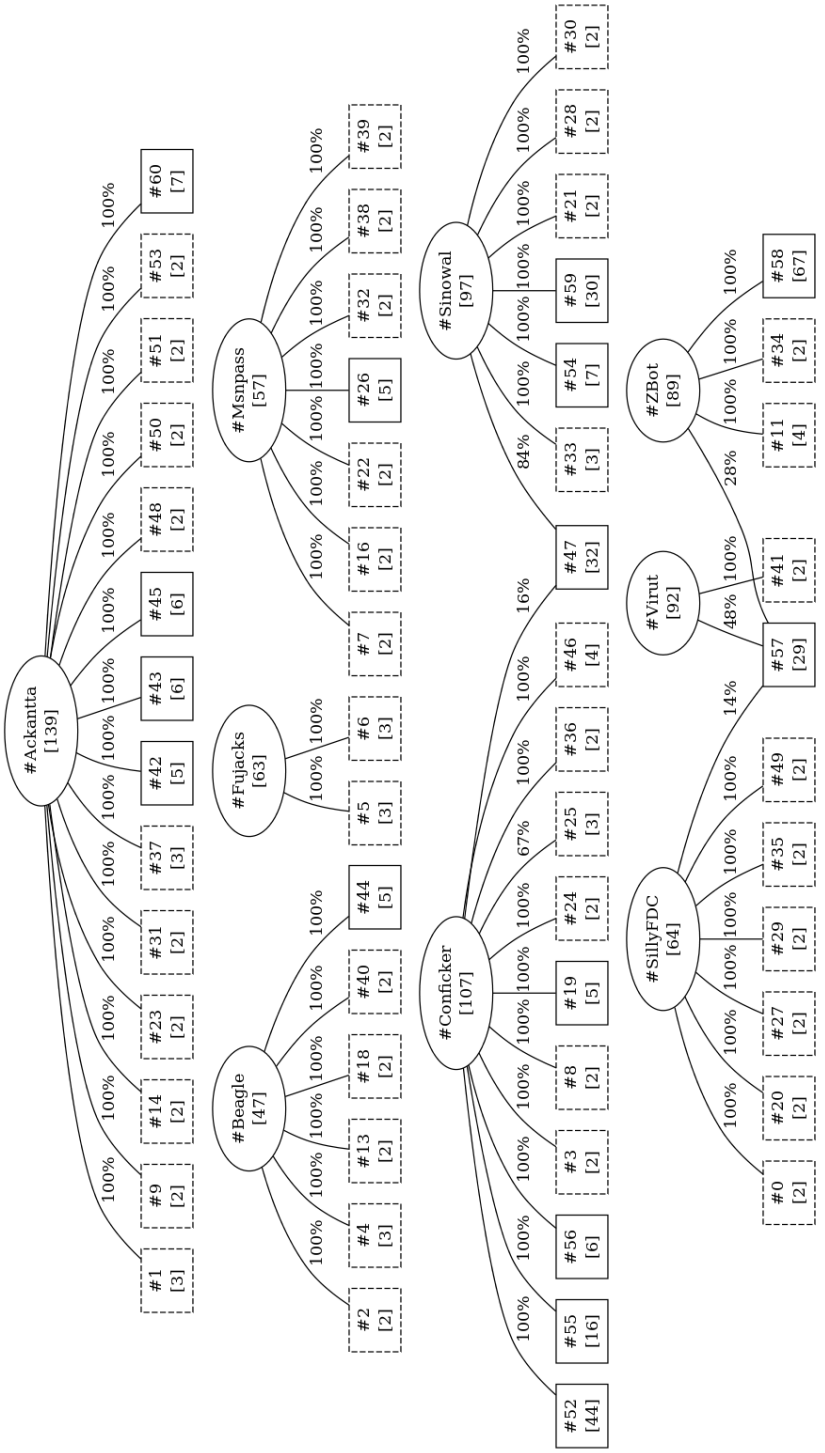

Figura 3.2: Clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSH. Sono state rimosse alcune associazioni di scarsa entità tra cluster e famiglie, allo scopo di rendere più leggibile il diagramma.

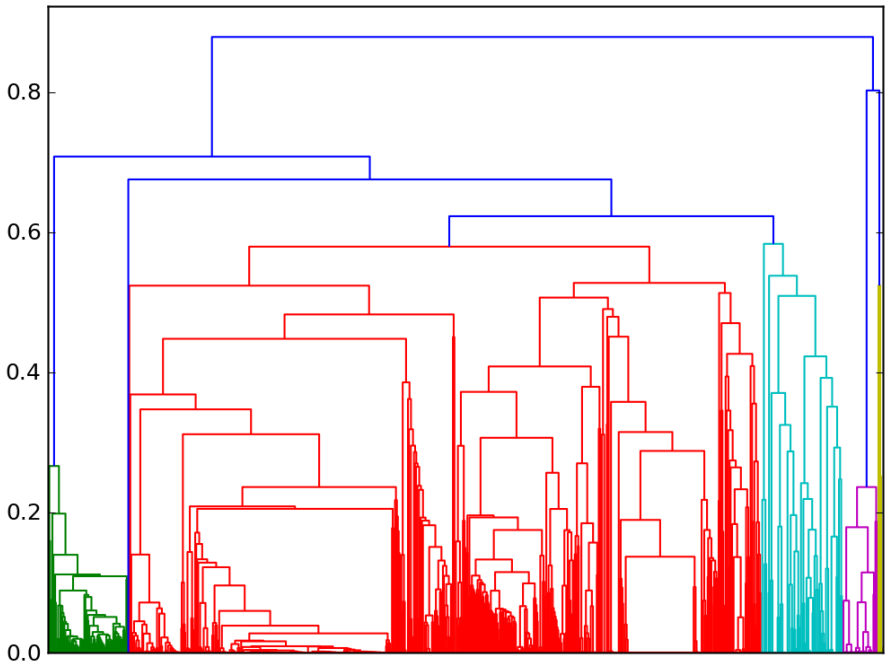

Figura 3.3: Dendrogramma corrispondente al clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSI

Metodo basato sull'algoritmo LSI

Il primo passo di questo metodo consiste nel ridurre le dimensioni della matrice utilizzando la decomposizione SVD troncata. In letteratura [22] si trova citato più volte il valore $k = 300$ come numero ideale di dimensioni atto a rappresentare una base di conoscenza.
Dopo aver ricavato le matrici U, Σ e V^T, costruiamo la matrice delle similarità campioni-campioni. In seguito utilizziamo la matrice complementare delle distanze, che contiene tutti i valori della metrica coseno (A.2).

L'ultimo passo richiede di applicare l'algoritmo di clustering gerarchico tradizionale alla matrice appena creata. Utilizziamo il metodo basato sulla *distanza coseno*, e il criterio di collegamento *average linkage* (A.7). L'algoritmo appena descritto produce un albero di cluster, raffigurato tramite dendrogramma in Figura 3.3, che viene appiattito in corrispondenza della soglia di distanza $T_d = 0,3$.

Questo metodo fornisce risultati significativi solamente utilizzando la matrice completa, poiché eliminando alcune colonne si elimina gran parte dell'informazione semantica contenuta nei dati. LSI trasforma la matrice portandola dallo spazio vettoriale delle feature dinamiche ad un nuovo spazio vettoriale di dimensione inferiore. Nel seguito concentriamo l'attenzione sul clustering ottenuto a partire dalla matrice completa.

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	Matrice completa		Matrice ridotta	
Ackantta	99,28	10	97,12	13
Beagle	89,36	10	87,23	11
Conficker	99,07	2	99,07	2
Fujacks	90,48	8	87,30	11
Msnpass	85,96	7	84,21	9
MyDoom	95,06	1	95,06	2
SillyFDC	89,06	9	89,06	11
Sinowal	96,91	6	95,88	6
Virut	97,83	2	97,83	2
ZBot	100	5	98,88	5

Tabella 3.2: Clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSI

I risultati dell'esecuzione di LSI sono mostrati in Figura 3.4 e presentati in Tabella 3.2. La copertura è calcolata come definito in (3.3). Riportiamo solamente i cluster contenenti due o più elementi.

Analizzando nel dettaglio il clustering generato osserviamo che LSI produce un numero sensibilmente inferiore di cluster spuri rispetto a LSH. Tuttavia, molti cluster risultano associati a più famiglie. Questo fatto deriva principalmente da due aspetti:

- la definizione di famiglia di malware è molto generica, come descritto nel capitolo precedente. All'interno della stessa possono trovarsi campioni appartenenti a tipi differenti (es., worm e trojan possono coesistere). Capita sovente che campioni dello stesso tipo, ma appartenenti a famiglie diverse (es., dropper appartenenti alle famiglie Conficker e Sinowal), abbiano comportamenti più simili rispetto a campioni di una stessa famiglia ma di tipologia diversa.

- LSI trasforma una combinazione di feature dinamiche in un unico *concetto*. All'interno dello stesso concetto possono essere riassunte azioni malevoli completamente differenti, legate tuttavia da qualche forma di correlazione nei dati. In questo modo è altamente probabile che campioni aventi feature dinamiche distinte, che vengono mappate in un unico concetto, siano assegnati allo stesso cluster.

Per valutare la qualità dei cluster ottenuti tramite questo metodo, calcoliamo i valori delle metriche *precision*, *recall* ed *f-measure*, definite in Appendice A.2. Il grafico di queste grandezze, al variare della soglia di distanza T_d, è rappresentato in Figura 3.5.

In corrispondenza del valore $T_d = 0.3$, la f-measure è massima. L'elevata precision garantisce che nei cluster non vengano inseriti malware che

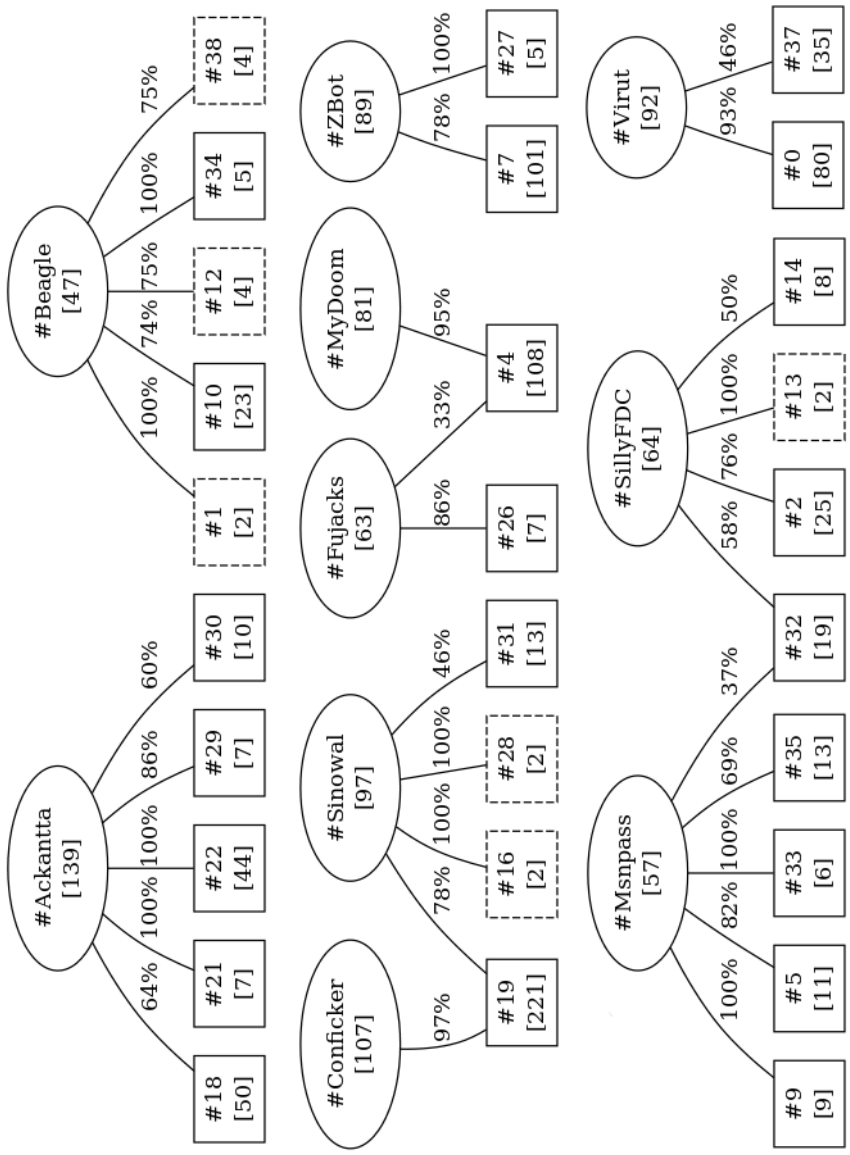

Figura 3.4: Clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSI. Sono state rimosse alcune associazioni di scarsa entità tra cluster e famiglie, allo scopo di rendere più leggibile il diagramma.

presentano comportamenti troppo diversi. La recall invece assicura che campioni simili vengano inseriti nello stesso cluster. Osservando il grafico relativo alla copertura, sempre in Figura 3.5, notiamo che la percentuale di malware clusterizzati è molto alta anche per valori di distanza relativamente bassi.

3.3 Analisi dei risultati

Ispezionando manualmente il contenuto di ciascun cluster, rappresentato in Figura 3.4, osserviamo che LSI è in grado di individuare la maggior parte delle varianti dei malware appartenenti alla stessa famiglia. Spesso tali varianti sono attribuite a cluster diversi anche se non sempre questa suddivisione corrisponde alle tipologie.

Confrontando tra loro i cluster corrispondenti ad una stessa famiglia osserviamo che differiscono solamente per poche azioni svolte dai campioni. Di conseguenza tali insiemi non sono disgiunti sulla base dei tipi di malware, ma in funzione dei comportamenti manifestati durante l'analisi. Ad esempio, la famiglia Virut viene suddivisa in due cluster distinti (#0 e #37). Il cluster #0 diversamente dal #37 contiene malware che modificano le impostazioni di configurazione per Internet, creano nuovi processi, scrivono nelle cartelle di sistema e non terminano la loro esecuzione entro i tempi previsti da ANUBIS. Invece, i cluster corrispondenti a due famiglie distinte manifestano caratteristiche diverse, ma non sempre le differenze risultano marcate.

In generale, dati due cluster arbitrari prodotti da LSI non possiamo attribuire, basandosi unicamente sull'analisi delle azioni svolte, la famiglia di appartenenza. Osserviamo che il cluster #19 raccoglie campioni appartenenti alle famiglie Conficker e Sinowal. Tale associazione resta evidente anche trascurando le relazioni poco significative. Possiamo trovare una spiegazione in un recente articolo [10] che evidenzia una forte correlazione comportamentale tra le due famiglie, dovuta al fatto che probabilmente sono state sviluppate dallo stesso team.

Analizziamo le inconsistenze tra la suddivisione in famiglie, considerata come clustering di riferimento, ed i cluster prodotti dalla categorizzazione comportamentale applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7103$, ci si aspetta quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 3.3

La consistenza tra le famiglie e il cluster prodotto è molto bassa. Per valori di $T_w \leq 10\%$ si presenta un'unica inconsistenza forte che coinvolge tutti i nodi appartenenti al grafo. Aumentando il valore della soglia T_w si

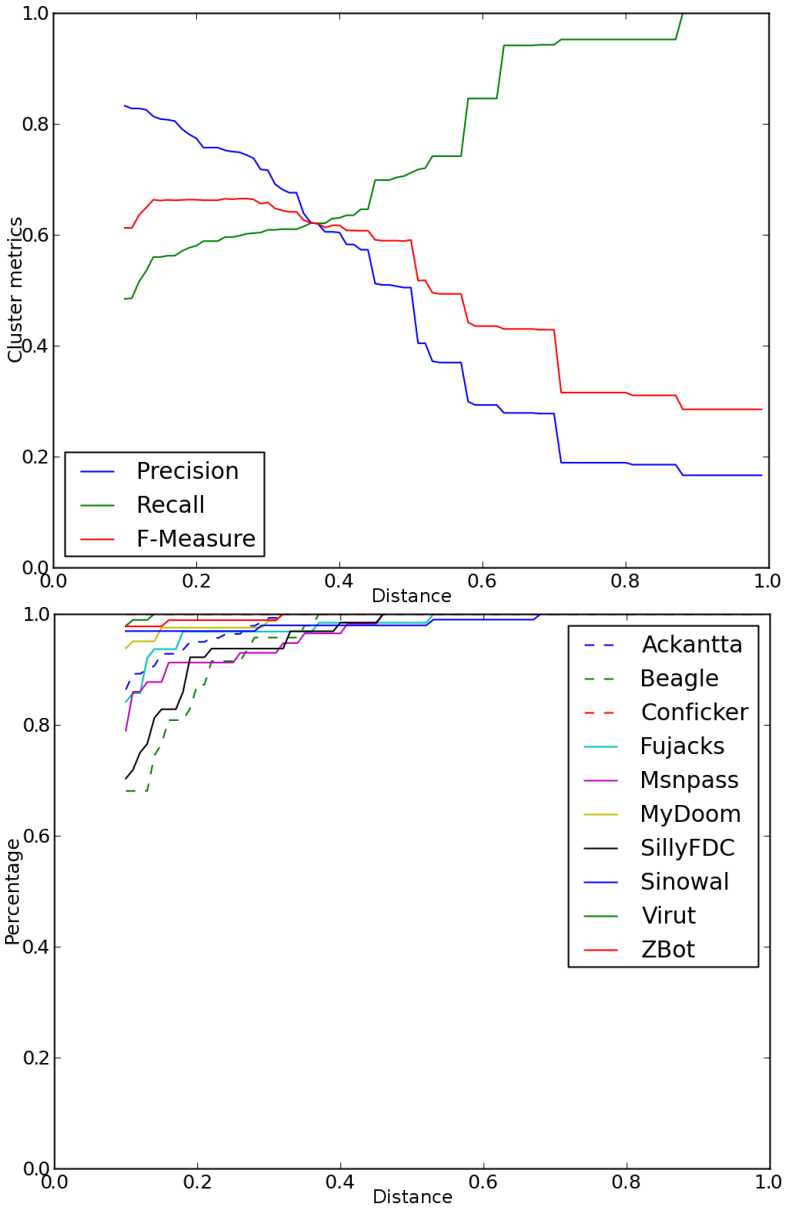

Figura 3.5: Diagrammi di precision-recall (in alto) e copertura (in basso) relativi al clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSI

T_w	IF	ID	C
0%	1	-	-
5%	1	-	-
10%	1	-	-
20%	2	-	-
30%	3	4	-
40%	1	6	2

Tabella 3.3: Analisi delle inconsistenze tra il clustering basato su feature dinamiche, ottenuto tramite l'algoritmo LSI, e quello di riferimento

riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Per $T_w = 30\%$ le famiglie si iniziano a mappare su un insieme di cluster comportamentali maggiormente definito, come raffigurato in 3.4. Aumentando ulteriormente la soglia T_w rimane l'unica inconsistenza forte tra le famiglie Conficker e Sinowal giustificata precedentemente.

Capitolo 4

Clustering basato su feature statiche

In questo capitolo ci concentriamo sulle tecniche di analisi statica. Descriviamo e applichiamo dei metodi di clustering basati su feature statiche e ne analizziamo i risultati.

Come feature statiche consideriamo inizialmente le fingerprint estratte da DISASM, corrispondenti a sequenze di 10 basic block, dopodiché aumentiamo il dettaglio considerando i singoli blocchi ed infine ci concentriamo su sequenze di dimensione arbitraria chiamate super-fingerprint.

Per identificare i basic block consideriamo dapprima il colore associato ad ognuno di essi, dipendente dalle classi di istruzioni contenute. Tuttavia, questo identificatore è troppo generico, di conseguenza, associamo a ciascun blocco "colorato", l'insieme di tutte le fingerprint che lo attraversano.

Dopo aver descritto le caratteristiche dell'analisi statica e i problemi da cui è afflitta (Sezione 4.1) dettagliamo i metodi basati sulle feature statiche (Sezione 4.2), descrivendo la costruzione e riduzione della matrice dei dati e gli algoritmi di clustering impiegati e analizzando i risultati prodotti da ciascun metodo. Infine confrontiamo i risultati ottenuti (Sezione 4.3).

4.1 Analisi statica

L'analisi statica è il processo di analisi del codice di un programma senza lanciarlo in esecuzione. In questo processo, il binario viene solitamente disassemblato, ossia trasformato nelle corrispondenti istruzioni assembler. In seguito possono essere applicate tecniche di analisi del flusso di controllo e del flusso dati, per ottenere informazioni circa le funzionalità del programma.

Per poter confrontare il campione con altri, si cercano particolari sequenze di istruzioni (chiamate *firme*, *signature* o *fingerprint*) che possano essere utilizzate per identificare il campione. Se viene riconosciuta una sequenza

riconducibile alla firma di un malware noto, è altamente probabile che il software in esame sia infetto o malevolo.

L'analisi statica ha il vantaggio che può coprire l'intero codice del programma, ed è solitamente più rapida della sua controparte dinamica.

4.1.1 Contromisure

L'approccio basato sul riconoscimento di sequenze di istruzioni è efficace finché il codice originale dei malware non subisce sostanziali modifiche, dato che, per identificarlo, il codice che lo caratterizza deve rimanere immutato. Gli autori di malware tendono ad implementare codice difficilmente analizzabile tramite le tecniche di analisi statica. In particolare tendono ad applicare contromisure basate sull'offuscamento del binario complicandone così sia la fase di disassemblamento che l'analisi vera e propria.

Con il termine *offuscamento* ci si riferisce a tecniche che mantengono immutate la semantica e le funzionalità del programma, ma allo stesso tempo ne rendono molto più difficile l'estrazione del codice assembler e la comprensione della struttura. Nel contesto del disassemblamento, l'offuscamento si riferisce alla trasformazione del binario in modo tale che il parsing delle istruzioni sia difficile. Inoltre il codice stesso può essere offuscato in modo da rendere più complicata l'estrazione del flusso di controllo di un programma e l'analisi del flusso dati. Queste contromisure possono essere applicate in modo automatico, ma difficilmente invertite anche se la trasformazione è nota. Le tecniche di offuscamento si dividono in due grosse categorie: *polimorfiche* [23, 24] e *metamorfiche*.

Un malware polimorfo presenta il proprio payload criptato e lo decripta, tramite un'apposita routine, al momento dell'esecuzione. Quando un malware polimorfo si replica, infettando un nuovo sistema, cripta il suo payload con una chiave differente e modifica il codice della routine di decifrazione con l'inserimento di istruzioni *nop*[1], trasposizione di codice o salti incondizionati. Per riferirsi ai malware il cui payload è compresso oppure cifrato si utilizza il termine *packed*. Per decomprimere il payload la routine di unpaking può essere invocata una sola volta, in questo 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 momoria in momenti diversi. Ciò significa che il malware deve essere eseguito in una sandbox per poter analizzare il payload e tale analisi deve essere posticipata fino a quando la routine di unpacking ha decifrato il payload.

L'utilizzo di packer permette di aggirare le tecniche basate su fingerprint. L'approccio tradizionale per gestire i malware polimorfi è quello di usare le specifiche routine di unpacking, diverse per ogni algoritmo conosciuto,

[1] "No Operation Performed": istruzione che non ha alcun effetto sul codice ma si limita ad incrementare di uno il program counter.

prima di effettuare l'analisi del codice. Questo approccio non fornisce ottimi risultati poiché i malware possono possedere più strati di packing e non è possibile conoscere tutti gli algoritmi.

Un malware di tipo metamorfico invece, quando si replica, modifica il suo codice, mantenendo le medesime funzionalità, in molteplici modi: tramite l'inserimento di nop, con la trasposizione di codice, riassegnazione dei registri, ... Dettagliamo maggiormente le principali tecniche di offuscamento impiegate dai codici metamorfici.

Inserimento di "codice morto" aggiunge codice al programma senza modificarne il comportamento. L'esempio più semplice è l'inserimento di istruzioni nop. Offuscamenti più complessi sono costituiti da sequenze di codice e istruzioni che modificano lo stato del programma per ripristinarlo subito dopo (es., $x = x + 2$ immediatamente seguito da $x = x - 2$) oppure parti di codice che non saranno mai eseguite ma inserite appositamente per disturbare l'analisi statica. Non tutte le sequenze di codice morto possono essere rilevate ed eliminate, dato che, il problema è analogo a quello di *decidere se una sequenza di istruzioni è equivalente ad un programma vuoto*, notoriamente indecidibile.

Trasposizione di codice riorganizza la sequenza delle istruzioni all'interno del codice in modo tale che l'ordine di esecuzione delle stesse sia differente, ma le funzionalità restino inalterate. Un primo esempio di offuscamento può essere fatto ordinando in modo casuale le istruzioni e poi ricostruendo la corretta sequenza tramite salti incondizionati, mentre un secondo può consistere nello scambio di due istruzioni indipendenti. La prima tecnica è relativamente semplice da implementare, mentre la seconda è più complicata perché le dipendenze tra le istruzioni vanno accertate.

Riassegnazione dei registri scambia i nomi dei registri e non ha nessun altro effetto sul comportamento dinamico del programma. Tale modifica tuttavia causa variazioni nel codice oggetto.

Sostituzione di istruzioni sostituisce una o più istruzioni con una sequenza semanticamente equivalente (es., l'istruzione $x = x \times 2$ potrebbe essere sostituita con $x = x \ll 1$).

Le tecniche polimorfiche non sono efficaci nei confronti dell'analisi dinamica, poiché il codice stesso del malware provvederà a rimuovere queste contromisure durante l'esecuzione. Inoltre, buona parte delle tecniche metamorfiche vengono vanificate, in quanto non influenzano i comportamenti del campione stesso.

4.1.2 DISASM

DISASM [13, 25] è uno strumento di analisi statica che implementa una tecnica di costruzione delle fingerprint basata sulle informazioni di controllo di flusso, e consente di rilevare similarità strutturali tra varianti polimorfiche di codice malevolo.

Alcune parti di malware contengono codice eseguibile. È possibile che alcune parti di codice siano criptate, mentre altre direttamente eseguibili sulla macchina vittima (per decifrare il codice è necessario che all'interno del campione di malware ci sia un'apposita procedura). Campioni che non contengono codice eseguibile, come malware scritti in codici di scripting non compilati, non possono essere individuati da DISASM. Per tutte le parti eseguibili del codice vengono estratte delle fingerprint.

Per individuare il codice polimorfico DISASM genera fingerprint ad un più alto livello di astrazione rispetto al codice assembler. In questo modo si evita che semplici modifiche o riordinamenti di codice compromettano l'esito dell'analisi statica. Le fingeprint generate godono delle seguenti proprietà:

Unicità parti di codice eseguibile diverse vengono mappate su fingerprint differenti. Se il sistema non soddisfacesse questa proprietà potrebbero essere generati dei falsi positivi.

Robustezza a inserimenti e cancellazioni quando vengono aggiunte delle istruzioni a parti già esistenti di codice eseguibile, le fingerprint corrispondenti al codice originale non dovrebbero cambiare. Inoltre, a seguito di una cancellazione, la parte rimanente del codice dovrebbe essere identificata come parte dell'eseguibile originale. Questa proprietà è necessaria per contrastare semplici tentativi di evasione.

Robustezza a modifica il meccanismo di generazione delle fingerprint deve essere robusto contro determinate modifiche al codice. Questa proprietà è necessaria per identificare varianti di un singolo malware polimorfico.

Grazie a queste proprietà, DISASM è in grado di superare alcune contromisure all'analisi statica, quali inserimento di codice sporco, ridenominazione dei registri, trasposizione di codice e sostituzione di istruzioni.

Internamente a DISASM, la struttura dell'eseguibile viene descritta tramite il suo *grafo di controllo di flusso*, nel seguito CFG ("Control Flow Graph"). I nodi del CFG sono i basic block, ossia sequenze di istruzioni con un solo punto di ingresso e un solo punto di possibile diramazione, che, se esiste, coincide con il punto di uscita. Un arco dal blocco A al blocco B rappresenta un possibile flusso di controllo da A a B.

Le sottostrutture comuni a due CGF possono essere identificate verificando l'isomorfismo dei sottografi connessi di dimensione k, chiamati

k-sottografi, contenuti in tutti i CFG. Due regioni di codice sono correlate se condividono alcuni k-sottografi. In Figura 4.1 se ne mostra un esempio.

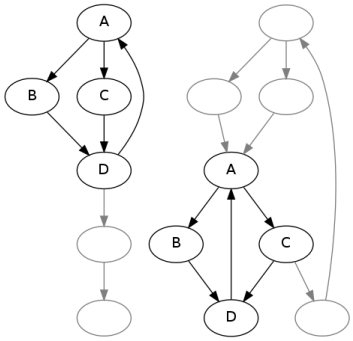

Figura 4.1: Sottografo di dimensione 4 comune a due CFG

Sottografi differenti devono mapparsi su fingerprint diverse per soddisfare la proprietà di unicità. Operazioni quali ridenominazione dei registri e sostituzione di istruzioni non hanno effetti sul CFG. Inoltre anche il riordino delle istruzioni all'interno di un basic block e la modifica del layout dei blocchi nel codice eseguibile lasciano inalterato il CFG. Questo rende la rappresentazione tramite CFG più robusta alle modifiche del codice rispetto ad altri approcci. Tuttavia, un avversario può introdurre modifiche che alterano il CFG stesso del malware. Nell'esempio in Figura 4.2 si osserva che l'inserimento di un nuovo blocco contenente una sola istruzione di salto incondizionato porta alla modifica del CFG originale.

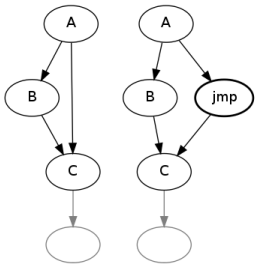

Figura 4.2: Grafo con flusso di controllo originale (sinistra) e modificato (destra)

Per rappresentare le informazioni inerenti i basic block viene assegnato un *colore* ad ogni nodo del CFG, generato in base alle istruzioni contenute nel blocco. Considerando questa nuova informazione, due k-sottografi sono isomorfi solo se i vertici sono connessi allo stesso modo e il colore di ognuno di essi corrisponde. Questo approccio consente di ridurre i falsi positivi.

DISASM utilizza come colore un valore a 14 bit, ciascuno dei quali corrisponde ad una certa classe di istruzioni riportate in Tabella 4.1.

Classe	Descrizione
Data transfer	Istruzioni mov
Arithmetic	Istruzioni aritmetiche, shift e rotazioni
Logic	Istruzioni logiche, operazioni su bit e byte
Test	Istruzioni test e compare
Stack	Istruzioni push e pop
Branch	Istruzioni di controllo di flusso condizionato
Call	Istruzioni di invocazione di funzione
String	Operazioni x86 per le stringhe
Flags	Accesso ai registri di flag x86
LEA	Istruzioni Load Effective Address
Float	Operazioni in virgola mobile
Syscall	Interrupt e chiamate di sistema
Jump	Istruzioni di controllo di flusso incondizionato
Halt	Istruzione di fermo esecuzione

Tabella 4.1: Classi di istruzioni impiegate da DISASM

Quando una o più istruzioni di una certa classe appaiono in un basic block il corrispondente bit del colore viene posto a 1. Se non sono presenti istruzioni appartenenti ad una certa classe il corrispondente bit è posto a 0.

I tentativi di offuscamento dovuti a sostituzione di istruzioni non possono essere completamente prevenuti utilizzando le classi di colore, ma vengono resi molto più difficili per l'attaccante. Il motivo è dato dal fatto che ci sono meno possibilità di trovare istruzioni semanticamente equivalenti appartenenti a classi diverse. In certi casi è praticamente impossibile sostituire un'istruzione con una semanticamente equivalente.

4.2 Uso di feature statiche

Questi metodi hanno l'obiettivo di costruire un clustering di malware basato su caratteristiche strutturali del CFG del programma, in seguito chiamate feature statiche, e si articolano nelle seguenti fasi:

Fase 1 sottoponiamo l'intero dataset costruito a DISASM e raccogliamo l'output per ciascun campione analizzato. Trascuriamo i file che generano degli errori nel prosieguo dell'analisi.

Fase 2 per ciascun campione elaboriamo l'output di DISASM, estraendone le feature statiche, e costruendo la matrice dei dati. Ogni metodo, successivamente descritto, si differenzia per la natura delle feature statiche prese in esame.

Fase 3 costruiamo il clustering partendo dalla matrice prodotta al passo precedente.

Vediamo nel dettaglio come si articola la seconda fase di ciascun metodo proposto, illustrando le caratteristiche strutturali utilizzate come feature statiche, e i risultati prodotti.

4.2.1 Fingerprint

In questo metodo utilizziamo come feature statiche le fingerprint estratte da DISASM, ossia hash dei sottografi di dimensione 10 estratti dal CFG del campione analizzato. Un campione presenta la *feature statica* j, se la corrispondente fingerprint è stata estratta da DISASM. Per ciascun malware se ne registrano le occorrenze.

Costruzione delle matrici dei dati

Raccogliamo tutti i dati nella matrice *campioni-caratteristiche* $\mathbf{X_{m,n}}$ (3.1), dove m è il numero dei campioni analizzati e n è il numero delle feature statiche. L'elemento $x_{i,j}$ assume valore 1 solo se il campione i presenta la caratteristica j, 0 altrimenti. La matrice risultante contiene 836 righe, 2.767.565 colonne e presenta un fattore di riempimento pari allo $0,4429\%$. Tale matrice, anche se molto sparsa, risulta difficile da utilizzare a causa delle dimensioni. Per le successive elaborazioni ci siamo dovuti appoggiare ad un calcolatore dotato di almeno 12GB di memoria centrale.

Eliminando le feature statiche univoche, ossia le caratteristiche strutturali che contraddistinguono un campione da un altro, si riesce a ridurre il numero delle colonne. La matrice ridotta $\mathbf{X'_{m,p}}$ (3.2) contiene lo stesso numero di righe di quella originale, 1.078.881 colonne e presenta un fattore di riempimento pari allo $0,9488\%$.

Clustering

Gli algoritmi tradizionali di clustering gerarchico di tipo agglomerativo non sono applicabili, poiché date le dimensioni della matrice, i tempi di calcolo delle distanze tra gli elementi risultano proibitivi.

I numeri primi necessari per il calcolo di LSH sono 2.767.571 per la matrice completa e 1.078.919 per quella ridotta. Omettiamo i risultati in quanto sono poco significativi data la quasi totalità di cluster spuri e la bassissima copertura.

Per l'algoritmo LSI riportiamo solamente i dati relativi alla matrice completa perché la ridotta non fornisce risultati significativi. Questo algoritmo prevede l'uso della decomposizione SVD troncata. Il valore di $k = 300$, più volte citato in letteratura, male si adatta a questo contesto perché l'informazione contenuta nel nuovo spazio vettoriale è ancora troppo selettiva e non permette di ottenere risultati apprezzabili. Abbiamo quindi progressivamente ridotto il valore del parametro k analizzando di volta in volta i risultati.

Osservando i dendrogrammi in Figura 4.3 possiamo notare che per un valore di $k \geq 100$ si forma un elevato numero di cluster spuri, rappresentati dalle linee verticali di colore blu, inoltre i campioni vengono aggregati per valori di distanza molto alti. Al contrario, per $k = 50$ la maggior parte dei campioni vengono aggregati per bassi valori di distanza facendo ipotizzare che ci sia molta informazione condivisa. Un buon compromesso lo troviamo in corrispondenza di $k = 75$.

I diagrammi *precision-recall* riportati in Figura 4.4 confermano quanto stabilito a partire dai dendrogrammi. L'alta precision che caratterizza i grafici ottenuti per $k \geq 100$ giustifica l'elevato numero di cluster unitari, mentre il valore della recall corrispondente a $k = 50$ concorda con l'aggregazione dei campioni. Per $k = 75$ si osserva che la *f-measure* rimane pressoché costante lungo tutto l'asse. Scegliamo quindi $T_d = 0,3$ come valore di soglia per la distanza in corrispondenza del quale verrà appiattito il dendrogramma.

Analisi dei risultati

I risultati dell'esecuzione di LSI, sono presentati in Tabella 4.2 e mostrati in Figura 4.6 per $k = 75$. La copertura è calcolata come definito in (3.3). Riportiamo solamente i cluster contenenti due o più elementi. Ispezionando manualmente il contenuto di ciascun cluster osserviamo che nella maggior parte dei casi si riescono a distinguere le varie famiglie e per ognuna di esse le tipologie diverse di malware sono raggruppate in insiemi differenti.

Confrontando tra loro i cluster corrispondenti ad una stessa famiglia osserviamo che si differenziano anche per una sola caratteristica comportamentale. Ad esempio, i cluster #22 e #19 della famiglia Ackantta differiscono per la capacità di avviarsi automaticamente; i cluster #32 e #66 della famiglia Conficker si diversificano per la causa di terminazione dei processi; un'altra ragione che sovente porta alla distinzione tra i cluster è il compilatore: i campioni appartenenti ai cluster #3 e #64 della famiglia Msnpass sono compilati rispettivamente con Microsoft Visual Basic e Borland Delphi.

Analizziamo ora le inconsistenze tra la suddivisione in famiglie, considerata come clustering di riferimento, ed i cluster prodotti dalla categorizzazione strutturale basata su fingerprint, applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7468$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 4.3. La consistenza tra le famiglie e il cluster prodotto è molto bassa. Per valori di $T_w \leq 20\%$ si presenta un'unica inconsistenza forte che coinvolge quasi tutti i nodi appartenenti al grafo ed un'inconsistenza debole tra la famiglia MyDoom e i relativi due cluster prodotti da LSI. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative

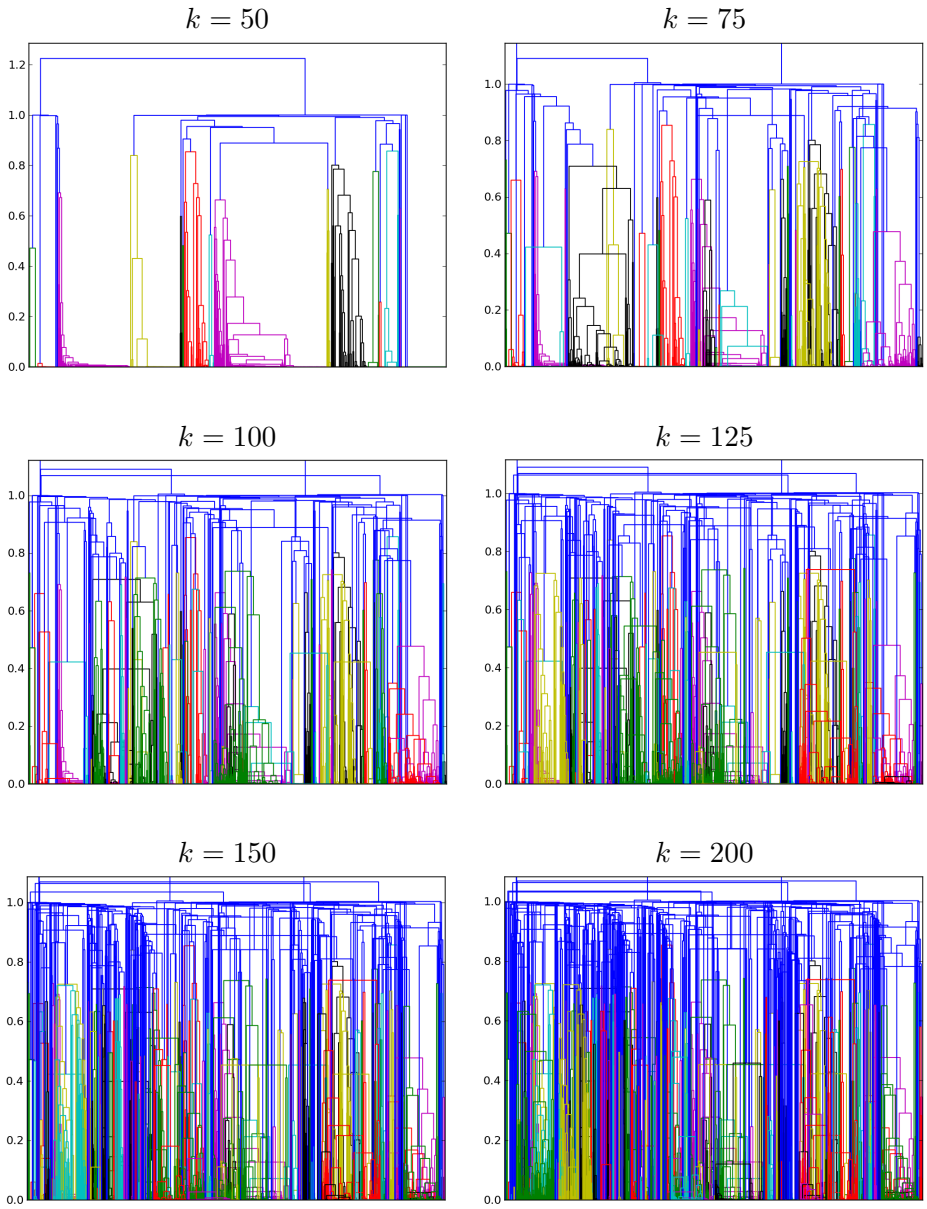

Figura 4.3: Dendrogrammi relativi al clustering delle feature basato su per il metodo basato su fingerprint, ottenuto tramite l'algoritmo LSI, al variare del parametro k

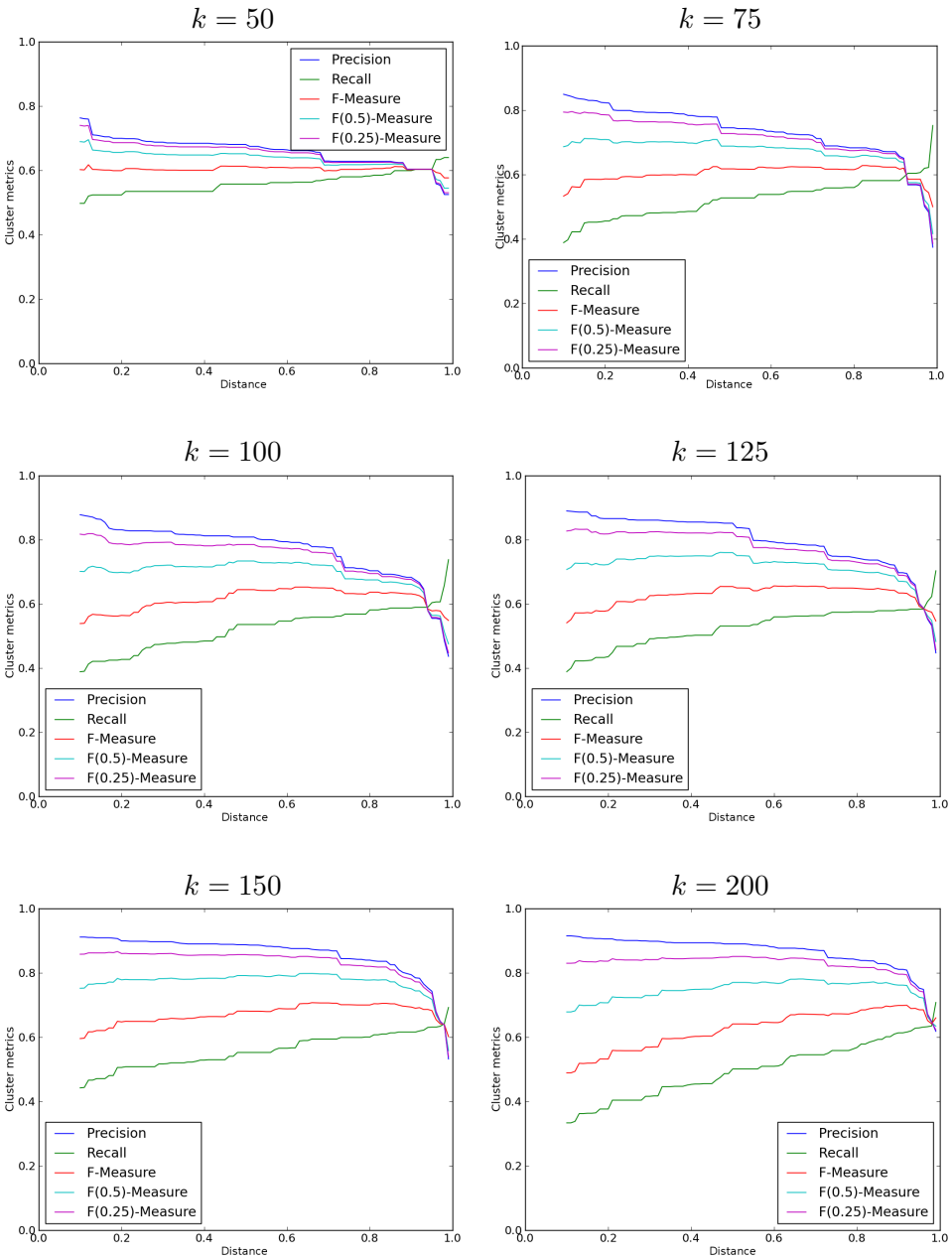

Figura 4.4: Diagrammi precision-recall relativi al clustering basato su feature statiche per il metodo basato su fingerprint, ottenuto tramite l'algoritmo LSI, al variare del parametro k

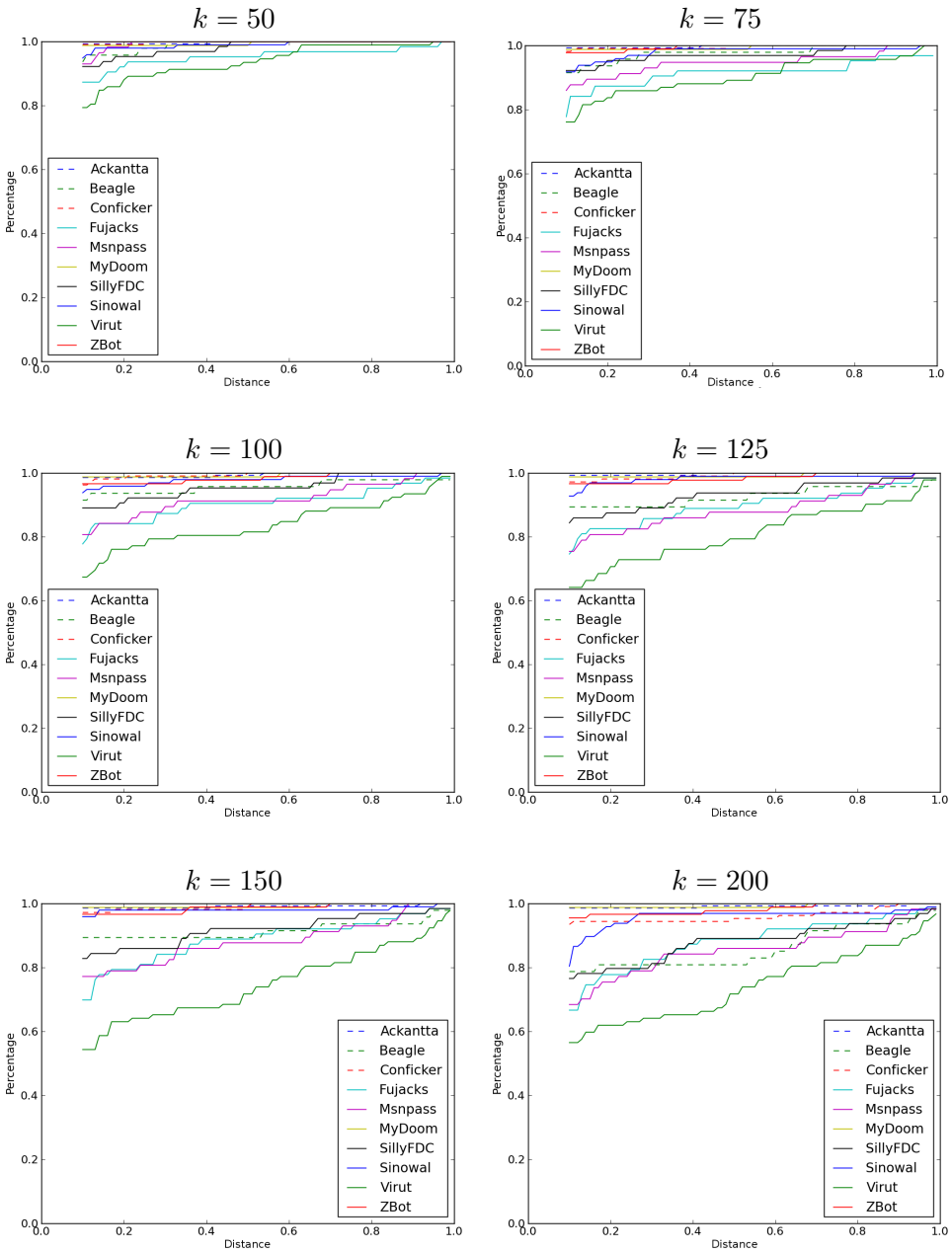

Figura 4.5: Diagrammi di copertura relativi al clustering basato su feature statiche per il metodo basato su fingerprint, ottenuto tramite l'algoritmo LSI, al variare del parametro k

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	k = 50		k = 75	
Ackantta	99,28	10	99,28	10
Beagle	95,74	5	85,11	8
Conficker	99,07	5	98,13	5
Fujacks	85,71	8	82,54	8
Msnpass	85,96	6	80,70	6
MyDoom	98,77	1	98,77	2
SillyFDC	90,63	6	84,38	8
Sinowal	94,85	3	95,88	9
Virut	88,04	16	82,61	15
ZBot	100	3	97,75	3

	k = 100		k = 125	
Ackantta	98,56	10	98,56	10
Beagle	89,36	9	87,23	6
Conficker	98,13	5	97,20	5
Fujacks	82,54	10	80,95	9
Msnpass	78,95	8	77,19	8
MyDoom	98,77	2	98,77	3
SillyFDC	81,25	7	81,25	10
Sinowal	95,88	9	96,91	8
Virut	76,09	16	69,57	15
ZBot	95,51	3	95,51	4

	k = 150		k = 200	
Ackantta	98,56	10	98,56	10
Beagle	87,23	7	78,72	7
Conficker	97,20	8	94,39	13
Fujacks	79,37	10	79,37	10
Msnpass	75,44	8	73,68	9
MyDoom	98,77	3	98,77	4
SillyFDC	79,69	10	75,00	9
Sinowal	96,91	4	95,88	16
Virut	64,13	13	63,04	13
ZBot	95,51	5	95,51	5

Tabella 4.2: Clustering basato su feature statiche (fingerprint), ottenuto utilizzando l'algoritmo LSI, al variare del parametro k

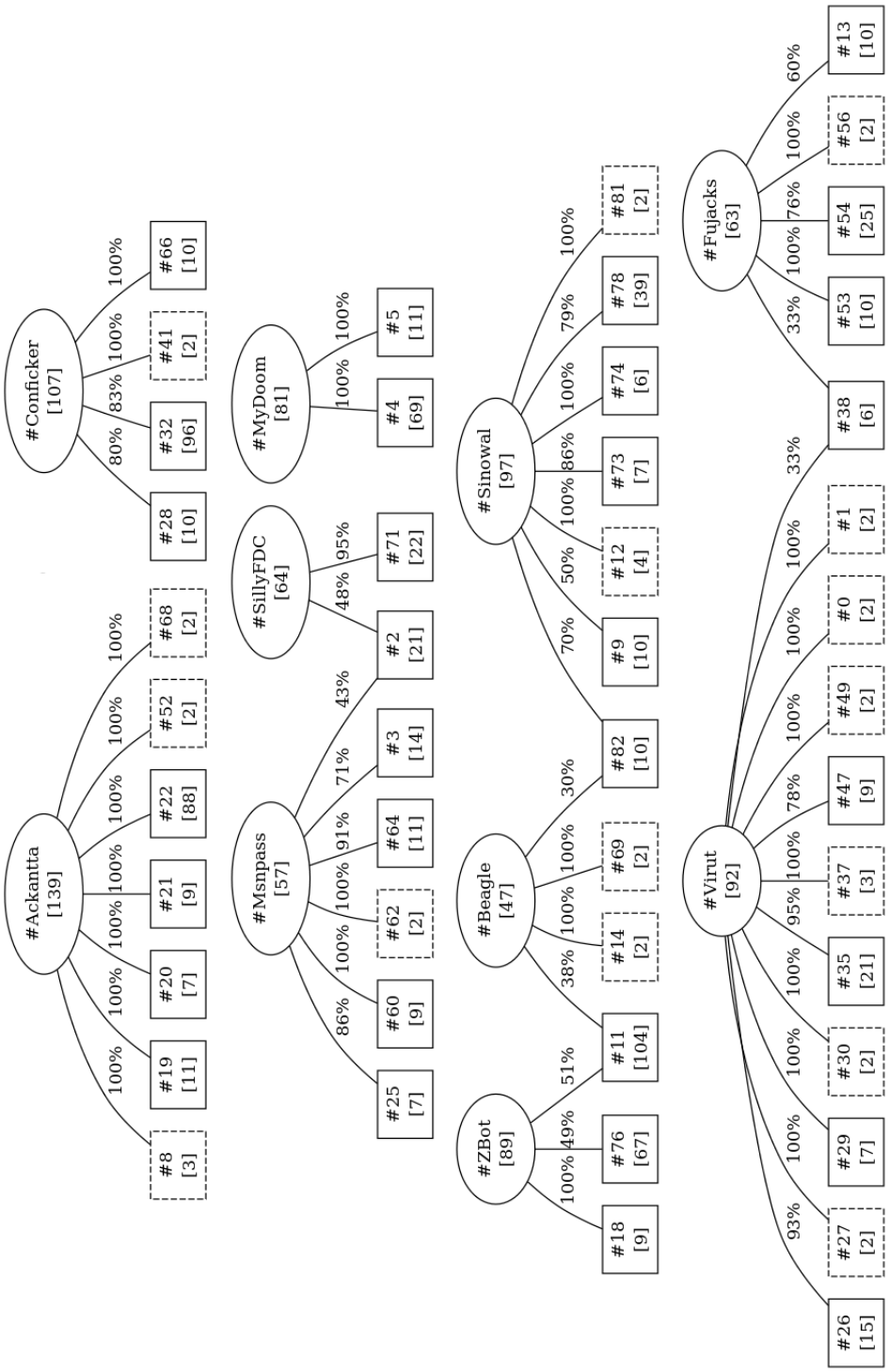

Figura 4.6: Clustering basato su feature statiche (fingerprint), ottenuto tramite l'algoritmo LSI con $k = 75$. Sono state rimosse alcune associazioni di scarsa entità tra cluster e famiglie, allo scopo di rendere più leggibile il diagramma.

T_w	IF	ID	C
0%	1	1	-
5%	1	1	-
10%	1	1	-
20%	1	1	-
30%	3	3	-
40%	1	8	-

Tabella 4.3: Analisi delle inconsistenze tra il clustering basato su feature statiche (fingerprint), ottenuto tramite l'algoritmo LSI, e quello di riferimento

e il grafo si divide in più componenti connesse. Per $T_w = 30\%$ le famiglie si iniziano a mappare su un insieme di cluster strutturali maggiormente definito, come rappresentato in Figura 4.6. Aumentando ulteriormente la soglia T_w rimane l'unica inconsistenza forte tra le famiglie Msnpass e SillyFDC. Queste ultime sono composte per la maggior parte da malware di tipo *downloader*. Il cluster #2 infatti, contiene campioni che scaricano da Internet altri malware e li lanciano in esecuzione.

Le fingerprint estratte da DISASM costituiscono un modo efficace per confrontare due campioni di malware, basandosi sull'informazione contenuta nel CFG. Tuttavia, i risultati ottenuti a partire da questi dati non consentono di distinguere né le varianti dei malware, né le tipologie. Non sempre infatti si riesce ad identificare un unico vero motivo che causa l'assegnamento di due campioni a cluster diversi.

4.2.2 Basic block, identificati tramite colore

In questo metodo utilizziamo come feature statiche i basic block, identificati tramite il colore assegnato da DISASM, ossia una maschera binaria che evidenzia la presenza di determinate classi di istruzioni all'interno del blocco. Un campione presenta la *feature statica* j, se il corrispondente blocco colorato è stato estratto da DISASM. Per ciascun malware se ne registrano le occorrenze.

Estrazione delle feature statiche

Le informazioni relative al colore dei basic block non sono incluse direttamente nell'output di DISASM, ma possono essere ricavate da un file ausiliario generato ad ogni esecuzione, chiamato `fast.dot`, contentente la descrizione del CGF ricostruito, come mostrato in Figura 4.7.

Ciascun basic block è rappresentato tramite un nodo, la cui etichetta contiene la posizione in memoria espressa come offset, il colore e le istruzioni assembler che lo compongono. Ogni etichetta rispetta il formato descritto

```
digraph message {
    node259767 [shape=box, label=" 3f6b7:   85e04\n
      add   0x869f7e45(%ecx),%cl\n
      or    $0xe23bf9f6,%eax\n
      ...
    node259739 [shape=box, label=" 3f69b:   88210\n
      mov   $0x747dfe3e,%ebx\n
      mov   $0x47a3fde4,%esi\n
      ...
    node259643 [shape=box, label=" 3f63b:   85704\n
      pop   %edi\n
      neg   %edx\n
      ...
    node259592 [shape=box, label=" 3f608: 1000004\n
      ljmp  *%eax\n"];
    ...
}
```

Figura 4.7: Frammento del file fast.dot. Il codice mostrato, a differenza di quello generato, è stato indentato e spaziato per facilitarne la comprensione.

dall'espressione regolare `<offset>: <colore>\n { <istruzione>\n}+`. Estraiamo queste informazioni leggendo la stringa contenuta nell'attributo *label* di ciascun nodo appartenente al grafo.

Molti autori di malware inseriscono "codice morto" per rendere più complesso il processo di analisi. Analizzando manualmente il codice assembler, che costituisce ciascun blocco, osserviamo che alcuni di essi terminano con istruzioni che non causano variazioni nel flusso di esecuzione, in disaccordo con la definizione di basic block. Altri sono composti da singole istruzioni senza evidenti scopi pratici (es., $x = x + 0$). Eliminiamo tutti questi blocchi sospetti prima di procedere con le fasi successive.

Consideriamo come validi tutti i blocchi, identificati tramite il colore, che terminano con un'istruzione di salto, chiamata o ritorno da procedura, o altre istruzioni che causano una variazione nel flusso di esecuzione.

Costruzione delle matrici dei dati

Raccogliamo tutti i dati nella matrice *campioni-caratteristiche* $\mathbf{X_{m,n}}$ (3.1), dove m è il numero dei campioni analizzati e n è il numero delle feature statiche. L'elemento $x_{i,j}$ assume valore 1 solo se il campione i presenta la caratteristica j, 0 altrimenti. La matrice risultante contiene 836 righe, 10.250 colonne e presenta un fattore di riempimento pari al $6,0893\%$.

Famiglia	Copertura [%]	Cluster
Ackantta	5,76	4
Beagle	10,64	2
Conficker	6,54	3
Fujacks	9,52	1
Msnpass	17,54	2
MyDoom	25,93	8
SillyFDC	0	0
Sinowal	0	0
Virut	0	0
ZBot	2,25	1

Tabella 4.4: Clustering basato su feature statiche (basic block identificati tramite colore), ottenuto utilizzando l'algoritmo LSH con la matrice ridotta

Eliminiamo le feature statiche univoche riducendo così il numero delle colonne. La matrice ridotta $\mathbf{X'_{m,p}}$ (3.2) contiene lo stesso numero di righe di quella originale, 8.262 colonne e presenta un fattore di riempimento pari al $7,5257\%$.

Clustering

I numeri primi necessari per il calcolo di LSH sono 10.253 per la matrice completa e 8.263 per quella ridotta. Ad ogni esecuzione il clustering ottenuto cambia completamente, a causa del fatto che alcuni colori risultano estramente frequenti, mentre altri sono molto rari. L'algoritmo probabilistico non fornisce buoni risultati su dati così poco uniformi. I risultati del clustering sulla matrice ridotta sono riportati in Tabella 4.4. La copertura è calcolata come definito in (3.3). Osserviamo che questo approccio fornisce una copertura del dataset limitata.

Come per il metodo basato su fingerprint, l'algoritmo LSI applicato alla matrice completa non fornisce risultati apprezzabili per il valore $k = 300$. Riduciamo quindi progressivamente il valore del parametro k analizzando di volta in volta i risultati ottenuti.

Osserviamo in Tabella 4.5 che per $k = 50$ la copertura è pressoché totale, mentre per $k \geq 150$ comincia a diminuire, come evidenziato dalle famiglie Sinowal e ZBot. Il valore $k = 75$ invece fornisce un buon compromesso.

I diagrammi *precision-recall* in Figura 4.8 confermano quanto appena osservato. Per bassi valori di k la recall è molto elevata e ciò giustifica la presenza di pochi cluster. La bassa precision invece non garantisce che nello stesso cluster siano presenti solamente campioni simili. Al contrario, per alti valori di k la precision cresce, ma il crollo della recall è in accordo con la riduzione della copertura. Scegliamo quindi $k = 75$ come dimensioni del

Famiglia	Copertura [%] k = 50	Cluster	Copertura [%] k = 75	Cluster
Ackantta	97,84	11	95,68	11
Beagle	100	4	93,62	6
Conficker	100	11	98,13	12
Fujacks	100	3	98,41	4
Msnpass	100	4	100	6
MyDoom	98,77	1	98,77	1
SillyFDC	100	3	100	7
Sinowal	100	2	100	2
Virut	98,91	6	98,91	8
ZBot	100	2	100	3

	k = 100		k = 150	
Ackantta	95,68	17	83,45	15
Beagle	82,98	5	76,60	7
Conficker	98,13	14	96,26	15
Fujacks	98,41	5	95,24	5
Msnpass	98,25	7	94,74	7
MyDoom	98,77	1	100	2
SillyFDC	100	9	93,75	11
Sinowal	98,97	2	47,42	13
Virut	97,83	8	93,48	9
ZBot	97,75	3	98,88	4

	k = 200		k = 300	
Ackantta	79,14	15	72,66	13
Beagle	74,47	8	61,70	9
Conficker	90,65	16	70,09	19
Fujacks	95,24	6	90,48	7
Msnpass	92,98	7	91,23	7
MyDoom	98,77	1	98,77	1
SillyFDC	81,25	9	68,75	8
Sinowal	19,59	1	17,53	3
Virut	92,39	11	88,04	14
ZBot	91,01	18	12,36	2

Tabella 4.5: Clustering basato su feature statiche (basic block identificati tramite colore), ottenuto utilizzando l'algoritmo LSI con la matrice completa

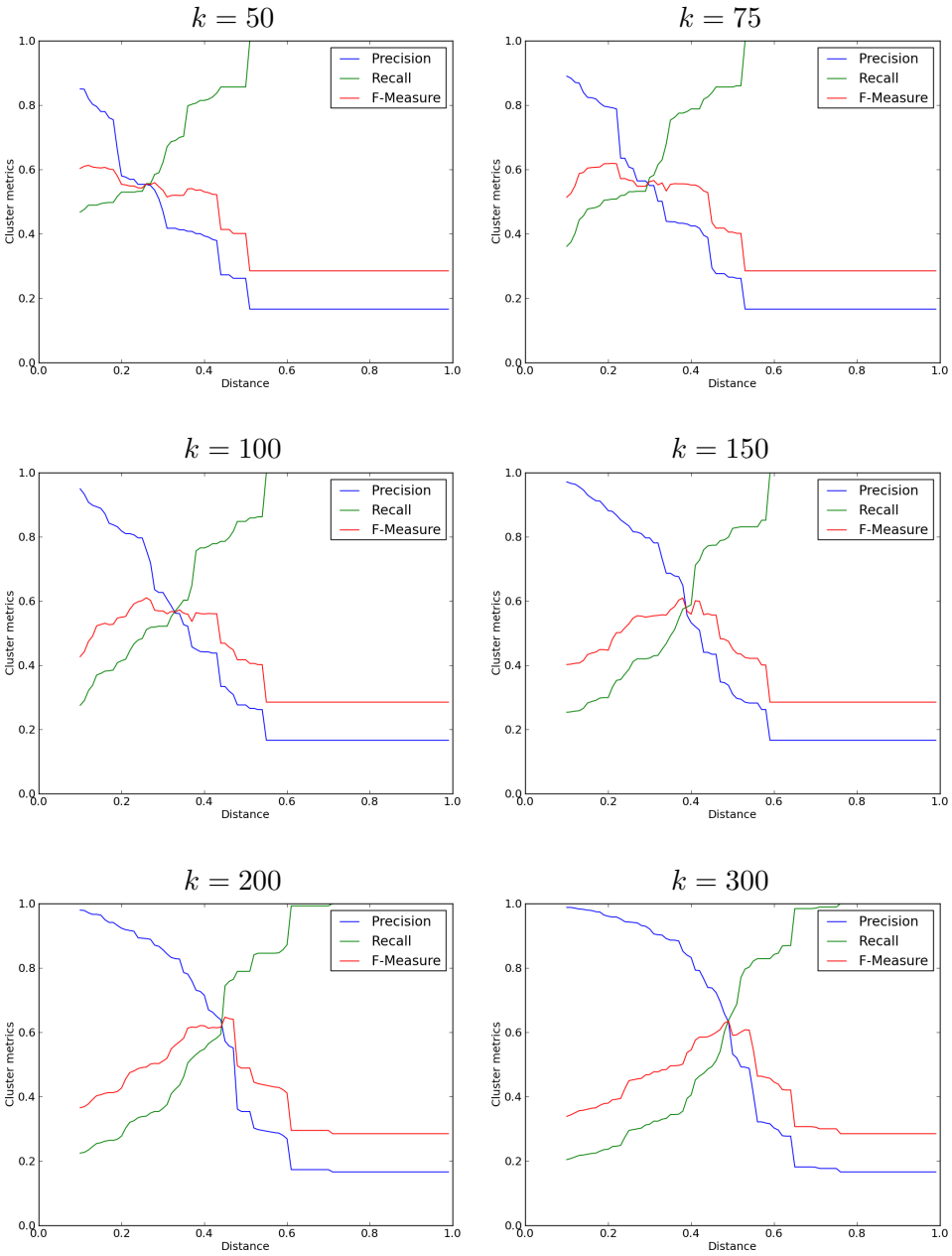

Figura 4.8: Diagrammi precision-recall relativi al clustering basato su feature statiche (basic block identificati tramite colore), ottenuto tramite l'algoritmo LSI, al variare del parametro k

T_w	IF	ID	C
0%	1	-	1
5%	1	-	1
10%	1	-	1
20%	1	-	1
30%	2	2	2
40%	2	2	3

Tabella 4.6: Analisi delle inconsistenze tra il clustering basato su feature statiche (basic block identificati tramite colore), ottenuto tramite l'algoritmo LSI, e quello di riferimento

nuovo spazio vettoriale e $T_d = 0,3$ come valore di soglia per la distanza in corrispondenza del quale verrà appiattito il dendrogramma.

Analisi dei risultati

I risultati dell'esecuzione di LSI, sono presentati in Tabella 4.5 e mostrati in Figura 4.9 per $k = 75$. La copertura è calcolata come definito in (3.3). Riportiamo solamente i cluster contenenti due o più elementi. Ispezionando manualmente il contenuto di ciascun cluster osserviamo che la maggior parte di essi sono piccoli, salvo poche eccezioni. Solamente le famiglie Ackantta, Fujacks, Msnpass e MyDoom si riescono a distinguere dalle altre. Il cluster #45, contenente 267 campioni, raccoglie la quasi totalità dei membri delle famiglie Beagle, Sinowal e ZBot assieme ad una buona percentuale di Conficker, mentre il cluster #7, contenente 121 campioni, unisce molti degli elementi di SillyFDC e Virut.

Analizziamo ora le inconsistenze tra la suddivisione in famiglie, considerata come clustering di riferimento, ed i cluster prodotti dalla categorizzazione strutturale basata su basic block identificati tramite colore, applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7810$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze, al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 4.6. La consistenza tra le famiglie e il cluster prodotto è molto bassa. Per valori di $T_w \leq 20\%$ si presenta un'unica inconsistenza forte che coinvolge quasi tutti i nodi appartenenti al grafo ed una consistenza tra la famiglia MyDoom ed il relativo cluster prodotto da LSI. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Per $T_w = 30\%$ le famiglie si iniziano a mappare su un insieme di cluster strutturali maggiormente definito, come rappresentato in Figura 4.9. Aumentando ulteriormente la soglia T_w

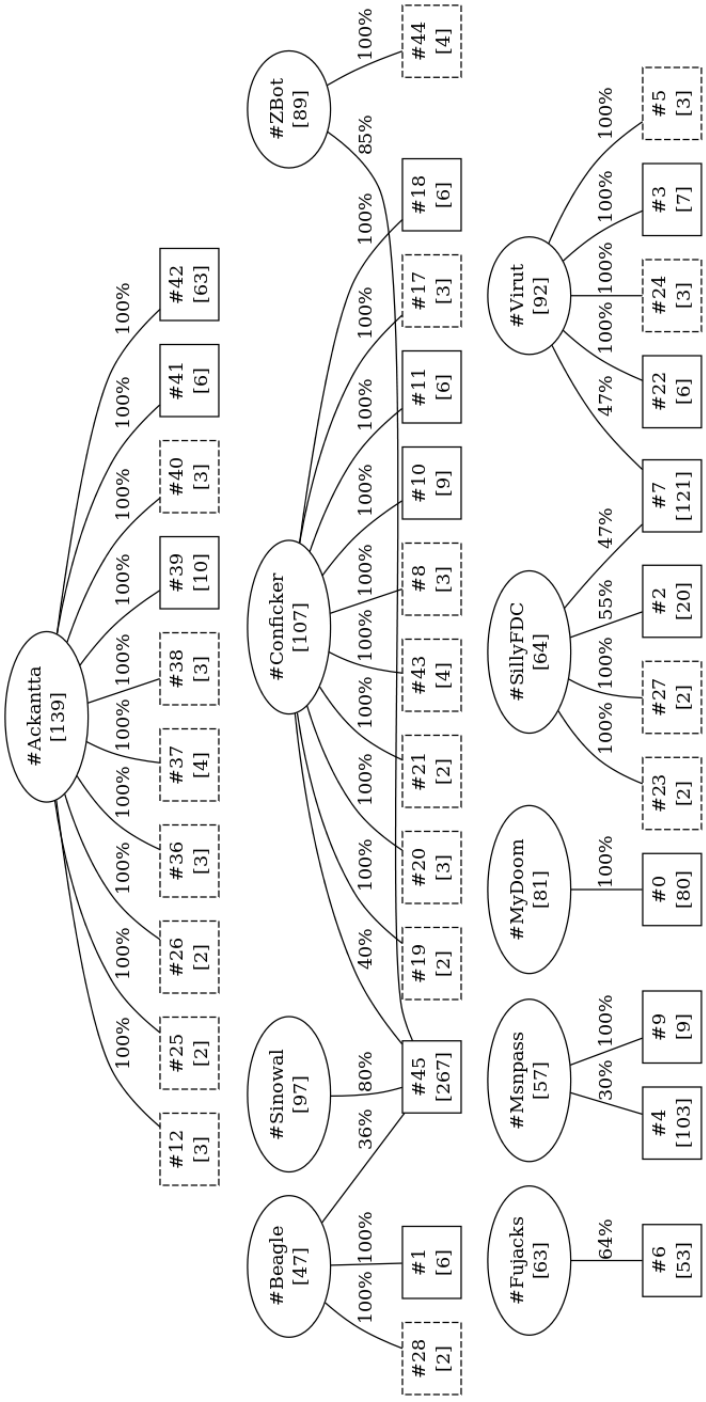

Figura 4.9: Clustering basato su feature statiche (basic block identificati tramite colore), ottenuto tramite l'algoritmo LSI con $k = 75$. Sono state rimosse alcune associazioni di scarsa entità tra cluster e famiglie allo scopo di rendere più leggibile il diagramma.

```
00ab1b7388fe569f,000cde40,000cdebc,000cdee9,000cdefd,...
0b6cb211c79b8e88,000de547,000de54d,000de56a,000de56c,...
0f967fac87c73cbe,000cdb79,000cdb8f,000cdba3,000cde3e,...
1418f4f1966cfd12,000cde3e,000cde40,000cdebc,000cdf46,...
1481404f8a209dc8,0008153b,0008153d,00081544,00081548,...
16cb4ee6adff1fc5,000cdba3,000cde3e,000cde40,000cdebc,...
198050d099316ffe,000de528,000de547,000de54d,000de56c,...
205261e0c6a193b6,000cdb79,000cdb8f,000cdba3,000cde3e,...
20867e984237daba,000cde3e,000cde40,000cdebc,000cdee9,...
217ff9646f6e950d,000cde40,000cdebc,000cdee9,000cdefd,...
...
```

Figura 4.10: Frammento di output dettagliato di DISASM. Ciascuna riga include la fingerprint in prima posizione, seguita dagli offset dei basic block del sottografo corrispondente.

rimangono due inconstenze forti, la prima formata dalle famiglie Conficker, Sinowal e ZBot, mentre la seconda, altrettanto significativa, composta dalle SillyFDC e Virut. Le inconsistenze deboli coinvolgono le famiglie Ackantta e Beagle ed i rispettivi cluster generati da LSI.

I basic block identificati tramite colore non costituiscono un modo efficace per confrontare la struttura di due campioni di malware. La verifica della presenza di blocchi contenenti determinate classi di istruzioni costituisce un criterio troppo vago per distinguere i basic block.

4.2.3 Basic block, identificati tramite colore e fingerprint

In questo metodo utilizziamo come feature statiche i basic block. A differenza del precedente sfruttiamo un modo più efficace per identificarli: aggiungiamo al colore, l'elenco delle fingerprint in cui tali blocchi compaiono. Corrediamo così l'identificativo con informazioni sia sul contenuto del blocco sia sul CFG di appartenenza. Un campione presenta la *feature statica* j, se contiene il corrispondente blocco. Per ciascun malware se ne registrano le occorrenze.

Estrazione delle feature statiche

Possiamo dettagliare ulteriormente l'output di DISASM tramite le opzioni "-w -B", ottenendo così, assieme alle fingerprint, l'elenco dei basic block che compongono il sottografo corrispondente. In prima posizione è contenuta la fingerprint, seguita dagli indirizzi in memoria dei basic block, espressi come offset. Un frammento di output dettagliato di DISASM è mostrato in Figura 4.10.

La costruzione delle nuove feature statiche, per ciascun campione, si articola nelle seguenti fasi:

Costruzione delle coppie (offset, hash-fingerprint) A partire dall'output dettagliato di DISASM estraiamo gli offset dei basic block e li correliamo alle rispettive fingerprint. Per ciascun offset raccogliamo le fingerprint in una lista ordinata e ne calcoliamo un hash. Al termine di questa fase abbiamo tutte le coppie (offset, hash-fingerprint).

Costruzione delle coppie (offset, colore) A partire dal file ausiliario `fast.dot` estraiamo tutte le coppie (offset, colore) per i blocchi ritenuti significativi, come descritto nella Sezione 4.2.2.

Costruzione delle feature statiche Presi gli insiemi ricavati ai passi precedenti, costruiamo le nuove coppie (colore, hash-fingerprint) unendo le tuple a parità di offset. Trascuriamo tutte quelle prive di corrispondenza. In questo metodo usiamo le coppie così create come feature statiche.

Costruzione delle matrici dei dati

Raccogliamo tutti i dati nella matrice *campioni-caratteristiche* $\mathbf{X_{m,n}}$ (3.1), dove m è il numero dei campioni analizzati e n è il numero delle feature statiche. L'elemento $x_{i,j}$ assume valore 1 solo se il campione i presenta la caratteristica j, 0 altrimenti. La matrice risultante contiene 836 righe, 578.037 colonne e presenta un fattore di riempimento pari allo $0,4143\%$.

Eliminiamo le feature statiche univoche riducendo così il numero delle colonne. La matrice ridotta $\mathbf{X'_{m,p}}$ (3.2) contiene lo stesso numero di righe di quella originale, 196.836 colonne e presenta un fattore di riempimento pari all'$1,1715\%$. Tuttavia quest'ultima operazione causa l'eliminazione di tutte le feature statiche corrispondenti a 133 campioni, pertanto non disponendo di informazione per tutti gli elementi del dataset originale non procediamo all'applicazione dell'algoritmo LSH.

Clustering

Come per i metodi basati sull'analisi statica descritti precedentemente, l'algoritmo LSI applicato alla matrice completa non fornisce risultati apprezzabili per il valore $k = 300$. Riduciamo quindi progressivamente il valore del parametro k analizzando di volta in volta i risultati ottenuti.

Osserviamo in Tabella 4.7 che per $k \leq 75$ si ha un'elevata copertura, mentre per $k \geq 100$ comincia a diminuire fino ad arrivare a poco più della metà del dataset. La copertura è calcolata come definito in (3.3). I diagrammi *precision-recall* in Figura 4.11 evidenziano una recall elevata per $75 \leq k \leq 150$. Per $k \geq 150$ la precision assume valori significativi, questo porta ad avere vari cluster unitari riducendo così la copertura. Per $k = 150$

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	k = 50		k = 75	
Ackantta	97,12	9	99,28	10
Beagle	89,36	9	93,62	5
Conficker	96,26	7	98,13	5
Fujacks	85,71	9	87,30	11
Msnpass	91,23	8	84,21	8
MyDoom	98,77	2	98,77	2
SillyFDC	89,06	12	84,38	5
Sinowal	95,88	13	95,88	2
Virut	82,61	15	80,43	13
ZBot	98,88	7	97,75	5
	k = 100		k = 150	
Ackantta	97,84	10	96,40	10
Beagle	85,11	7	80,85	7
Conficker	95,33	7	96,26	9
Fujacks	76,19	9	71,43	9
Msnpass	82,46	10	77,19	11
MyDoom	98,77	3	98,77	4
SillyFDC	81,25	12	75,00	12
Sinowal	92,78	14	98,97	7
Virut	72,83	12	67,39	13
ZBot	98,88	8	98,88	9
	k = 200		k = 300	
Ackantta	95,68	10	97,12	11
Beagle	80,85	9	72,34	10
Conficker	95,33	14	70,09	17
Fujacks	74,60	10	71,43	9
Msnpass	71,93	10	71,93	11
MyDoom	96,30	3	96,30	3
SillyFDC	68,75	11	65,63	10
Sinowal	96,91	10	68,04	27
Virut	60,87	13	54,35	11
ZBot	97,75	8	97,75	7

Tabella 4.7: Clustering basato su feature statiche (basic block identificati tramite colore associato alle fingerprint), ottenuto utilizzando l'algoritmo LSI con la matrice completa

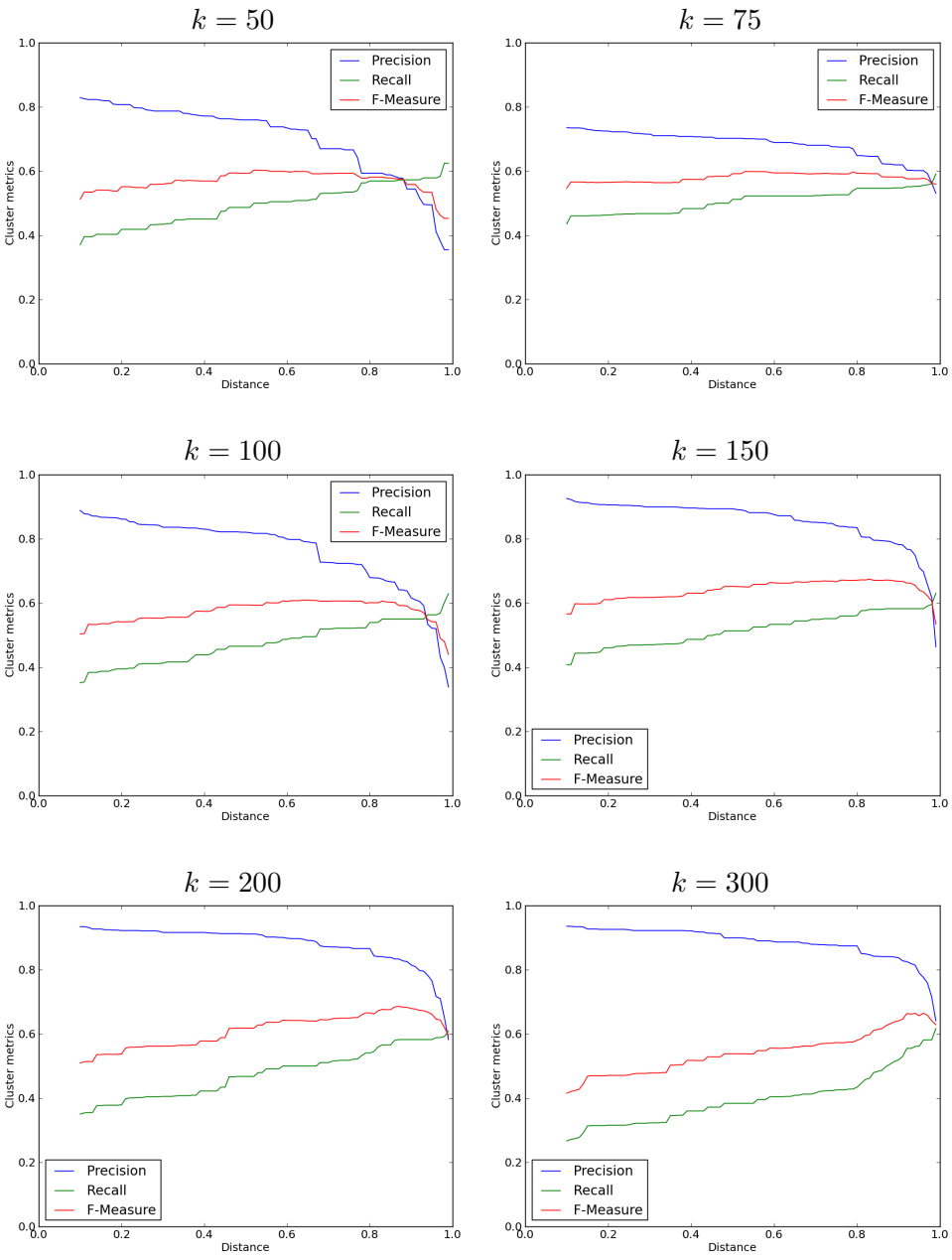

Figura 4.11: Diagrammi precision-recall relativi al clustering basato su feature statiche (basic block identificati tramite colore associato alle fingerprint), ottenuto tramite l'algoritmo LSI, al variare del parametro k

T_w	IF	ID	C
0%	1	1	-
5%	1	1	-
10%	1	1	-
20%	2	5	-
30%	1	8	-
40%	-	10	-

Tabella 4.8: Analisi delle inconsistenze tra il clustering basato su feature statiche (basic block identificati tramite colore associato alle fingerprint), ottenuto tramite l'algoritmo LSI, e quello di riferimento

abbiamo il più alto valore dell'*f-measure* per distanze ridotte. Scegliamo, pertanto, $k = 150$ come dimensioni del nuovo spazio vettoriale, anche se a discapito di una copertura non totale, e $T_d = 0,3$ come valore di soglia per la distanza in corrispondenza del quale verrà appiattito il dendrogramma.

Analisi dei risultati

I risultati dell'esecuzione di LSI, sono riportati in Tabella 4.7 e mostrati in Figura 4.12 per $k = 150$. La copertura è calcolata come definito in (3.3). Riportiamo solamente i cluster contenenti due o più elementi. Ispezionando manualmente il contenuto di ciascun cluster osserviamo che la dimensione media dei cluster prodotti è relativamente piccola. Nella maggior parte dei casi riusciamo a distinguere le varie famiglie in accordo con la clusterizzazione prodotta dalle fingerprint.

Confrontando tra loro i cluster corrispondenti ad una stessa famiglia osserviamo che si differenziano anche per una sola caratteristica comportamentale. Ad esempio, i cluster #16 e #43 della famiglia Ackantta differiscono per la creazione di nuovi processi e per il compilatore.

Come nel caso delle feature statiche basate su fingerprint non è possibile individuare un unico vero motivo che causa l'assegnamento di due campioni a cluster diversi.

Analizziamo ora le inconsistenze tra la suddivisione in famiglie, considerata come clustering di riferimento, ed i cluster prodotti dalla categorizzazione strutturale basata sui basic block identificati tramite colore associato alle fingerprint, applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7398$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze, al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 4.8. La consistenza tra le famiglie e il cluster prodotto è molto bassa. Per valori di $T_w \leq 10\%$ si presenta un'unica inconsistenza

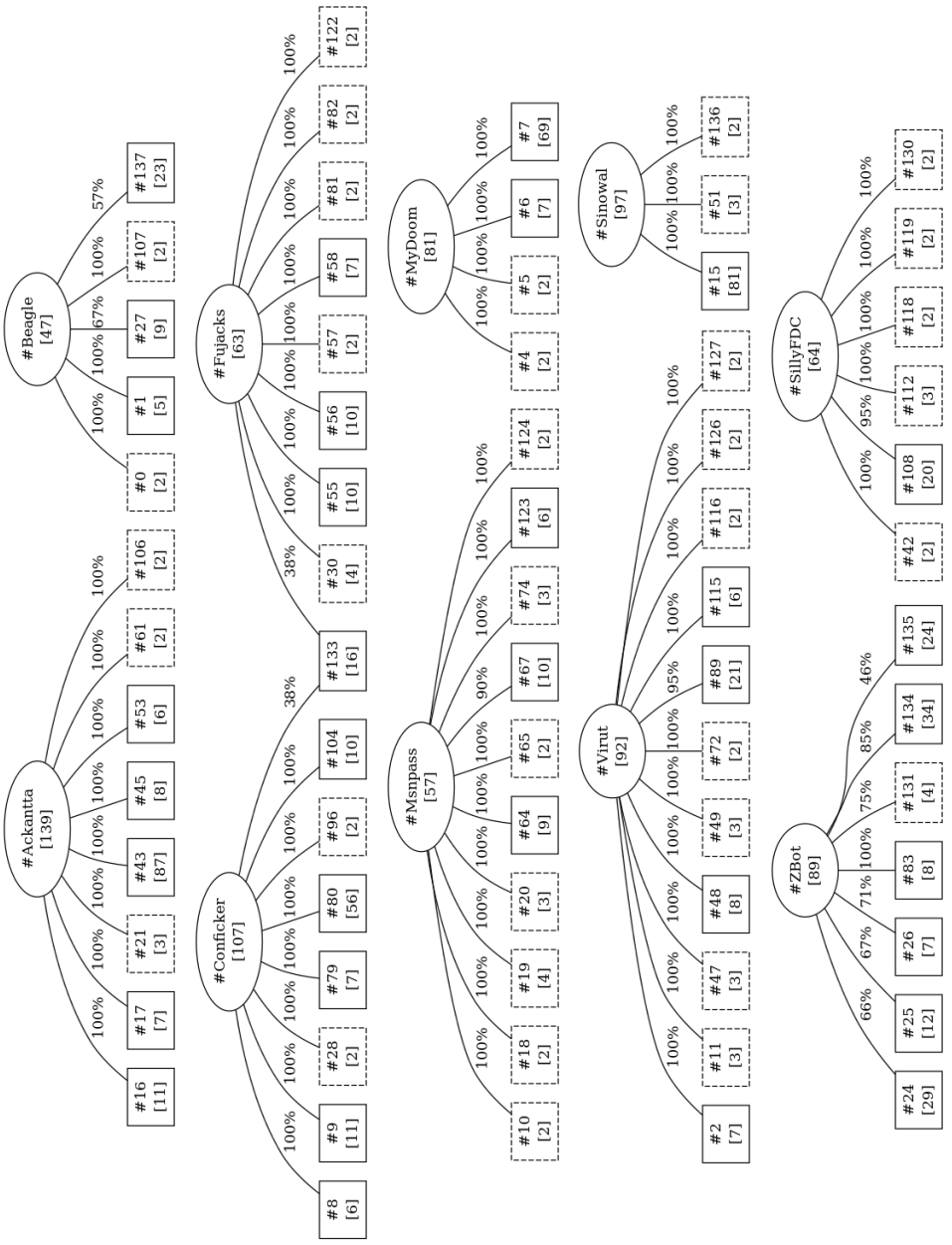

Figura 4.12: Clustering basato su feature statiche (basic block identificati tramite colore associato alle fingerprint), ottenuto tramite l'algoritmo LSI con $k = 150$. Sono state rimosse alcune associazioni di scarsa entità tra cluster e famiglie allo scopo di rendere più leggibile il diagramma.

forte che coinvolge quasi tutti i nodi appartenenti al grafo ed un'inconsistenza debole tra la famiglia MyDoom e i relativi quattro cluster prodotti da LSI. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Per $T_w = 20\%$ le famiglie si iniziano a mappare su un insieme di cluster strutturali maggiormente definito, come rappresentato in Figura 4.12. Aumentando ulteriormente la soglia T_w rimangono solo inconsistenze deboli tra le famiglie e i cluster prodotti da LSI.

L'aggiunta dell'informazione relativa alle fingerprint a quella del colore permette di discriminare maggiormente i basic block ottenendo così un clustering più coerente con le famiglie.

4.2.4 Super-fingerprint

In questo metodo utilizziamo come feature statiche le super-fingerprint, ossia hash dei sottografi di dimensione arbitraria ricostruiti a partire dall'output di DISASM del campione analizzato. Un campione presenta la *feature statica* j, se il sottografo corrispondente alla super-fingerprint fa parte del CFG del campione. Per ciascun malware se ne registrano le occorrenze.

Estrazione delle feature statiche

Partendo dall'output dettagliato di DISASM, rappresentato in Figura 4.10, ricostruiamo i sottografi di dimensione massima del CFG. DISASM scrive nel proprio output, per ciascuna riga, la fingerprint seguita dalla sequenza degli offset dei 10 basic block che ne compongono il sottografo. Alcuni sottografi risultano parzialmente sovrapposti. Inseguendo la sequenza degli offset possiamo ricostruire sottografi di dimensione superiore. La costruzione delle nuove feature statiche, per ciascun campione, si articola nelle seguenti fasi:

Estrazione degli archi Durante questa fase costruiamo l'insieme degli archi del CFG. Una singola fingerprint corrisponde ad una sequenza ordinata di 10 basic block. Partendo dall'output dettagliato di DISASM e considerando una riga alla volta, costruiamo gli archi a partire dalle coppie di blocchi consecutivi. Se una riga contiene 10 blocchi (salvo comportamenti anomali di DISASM) verranno generati 9 archi. Aggiungiamo tutti gli archi prodotti ad un insieme, scartando i duplicati.

```
edgeSet = set()
for line in fingerprintsFile:
    fingerprintLine = line.split(',')
    fingerprint = fingerprintLine[0]
    for i = 1 to fingerprintLine.length - 1:
        firstBlock = fingerprintLine[i]
        secondBlock = fingerprintLine[i + 1]
        edge = (firstBlock, secondBlock)
        edgeSet.add(edge)
```

Alla fine l'insieme degli archi contiene tutte le connessioni tra le coppie di blocchi che compongono il CFG.

Costruzione del CFG approssimato Durante questa fase aggreghiamo i blocchi estratti da DISASM tramite gli archi estratti nella fase precedente. Considerando ciascun arco contenuto nell'insieme generato al punto precedente, aggiungiamo i blocchi al grafo (se non sono già presenti) e l'arco che li collega.

```
cfg = graph()
for edge in edgeSet:
    (firstBlock, secondBlock) = edge
    if firstBlock not in cfg:
        cfg.addNode(firstBlock)
    if secondBlock not in cfg:
        cfg.addNode(secondBlock)
    cfg.addEdge(edge)
```

Alla fine il grafo contiene tutti i blocchi estratti da DISASM, legati tra loro con le relazioni estratte dall'output e indotte dalla struttura di ciascuna fingerprint.

Estrazione dei sottografi connessi Nell'ultima fase si estraggono i sottografi del CFG ottenuti al passo precedente, tramite ricerca delle componenti connesse. I sottografi così ottenuti riescono a ricostruire parte del flusso di controllo del programma analizzato.

Una volta estratti i sottografi di dimensione massima, è necessario trovare una rappresentazione che consenta di identificarli, proprio come fanno le fingerprint per i sottografi di dimensione K. Proponiamo tre rappresentazioni, a partire dalla più complessa:

(graph, fingerprintSet) una super-fingerprint è identificata da una tupla contenente il grafo raffigurante la componente connessa estratta all'ultima fase e l'insieme delle fingerprint corrispondenti ai blocchi contenuti nel grafo associato.

(blockSet, fingerprintSet) una super-fingerprint è identificata da una tupla contenente l'insieme dei blocchi che compongono il sottografo e quello delle fingerprint corrispondenti ai blocchi contenuti nell'insieme associato.

Hash una super-fingerprint è identificata da un hash calcolato sull'insieme ordinato delle fingerprint corrispondenti ai blocchi contenuti nel sottografo.

La prima rappresentazione è la più completa dal punto di vista semantico, ma è la meno adatta al confronto tra due super-fingerprint, poiché richiede la verifica dell'isomorfismo tra due grafi (problema computazionalmente difficile) o sottografi (problema NP-completo). La terza rappresentazione è la più semplice, ma consente di verificare solamente se due super-fingerprint coincidono (e quindi coincidono i grafi e le fingerprint che le hanno generate) o se sono diverse. Una minima variazione del grafo o delle fingerprint producono un hash differente, come se il grafo fosse completamente diverso. Non risulta quindi possibile verificare corrispondenze parziali. La seconda rappresentazione si colloca tra le precedenti. L'informazione semantica risente dell'eliminazione degli archi, ma è comunque più significativa di una singola hash. Questa rappresentazione consente di confrontare due super-fingerprint a patto di individuare una metrica che ne tenga in considerazione la natura insiemistica.

In questo metodo utilizziamo la terza rappresentazione per le super-fingerprint come feature statiche perché più semplice delle altre e conseguentemente più rapida da calcolare.

Costruzione delle matrici dei dati

Raccogliamo tutti i dati nella matrice *campioni-caratteristiche* $\mathbf{X_{m,n}}$ (3.1), dove m è il numero dei campioni analizzati e n è il numero delle feature statiche. L'elemento $x_{i,j}$ assume valore 1 solo se il campione i presenta la caratteristica j, 0 altrimenti. La matrice risultante contiene 836 righe, 32.397 colonne e presenta un fattore di riempimento pari allo $0,4135\%$.

Eliminiamo le feature statiche univoche riducendo così il numero delle colonne. La matrice ridotta $\mathbf{X'_{m,p}}$ (3.2) contiene lo stesso numero di righe di quella originale, 10.557 colonne e presenta un fattore di riempimento pari all'$1,2708\%$. Tuttavia quest'ultima operazione causa l'eliminazione di tutte le feature statiche corrispondenti a 164 campioni, pertanto non disponendo di informazione per tutti gli elementi del dataset originale non procediamo all'applicazione dell'algoritmo LSH.

Clustering

Come per i metodi basati sull'analisi statica descritti precedentemente, l'algoritmo LSI applicato alla matrice completa non fornisce risultati apprezzabili per il valore $k = 300$. Riduciamo quindi progressivamente il valore del parametro k analizzando di volta in volta i risultati ottenuti.

Osserviamo in Tabella 4.9 che per $k \geq 200$ si ha una riduzione della copertura e un incremento spropositato del numero di cluster. La copertura è calcolata come definito in (3.3).

I diagrammi *precision-recall* in Figura 4.13 evidenziano una precision molto elevata per $k \geq 150$, questo porta alla creazione di molti cluster unitari. Per $k = 75$ il valore della precision si riduce drasticamente non assicurando di mantenere campioni diversi in cluster distinti. In corrispondenza di $k = 100$ abbiamo il più alto valore della *f-measure* per valori bassi di distanza, pertanto scegliamo questo valore come dimensioni del nuovo spazio vettoriale, e $T_d = 0,3$ come valore di soglia per la distanza in corrispondenza del quale verrà appiattito il dendrogramma.

Analisi dei risultati

I risultati dell'esecuzione di LSI, sono riportati in Tabella 4.9 e mostrati in Figura 4.14 per $k = 100$. La copertura è calcolata come definito in (3.3). Riportiamo solamente i cluster contenenti due o più elementi. Ispezionando manualmente il contenuto di ciascun cluster osserviamo che nella maggior parte dei casi si riescono a distinguere le varie famiglie e per ognuna di esse le tipologie e le varianti diverse di malware sono raggruppate in insiemi differenti.

Confrontando tra loro i cluster corrispondenti ad una stessa famiglia osserviamo che, in accordo con gli antivirus, riusciamo a riconoscerne la tipologia o la variante. Ad esempio, i cluster #9 e #47 della famiglia Conficker contengono worm, il primo le varianti A e B, mentre il secondo B e C. Inoltre il cluster #85, sempre della famiglia Conficker, contiene trojan di tipo dropper/downloader. Un'altra ragione che porta alla distinzione tra i cluster è il compilatore: i campioni appartenenti ai cluster #13 e #77 della famiglia Msnpass sono compilati rispettivamente con Microsoft Visual Basic e Borland Delphi.

Analizziamo ora le inconsistenze tra la suddivisione in famiglie, considerata come clustering di riferimento, ed i cluster prodotti dalla categorizzazione strutturale basata su super-fingerprint, applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7487$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze, al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 4.10. La consistenza tra le famiglie e il cluster prodotto

Famiglia	Copertura [%] k = 50	Cluster	Copertura [%] k = 75	Cluster
Ackantta	99,28	9	100	10
Beagle	87,23	9	97,87	4
Conficker	97,20	8	98,13	5
Fujacks	84,13	8	87,30	10
Msnpass	91,23	10	87,72	9
MyDoom	98,77	2	98,77	2
SillyFDC	92,19	9	92,19	4
Sinowal	94,85	10	100	2
Virut	81,52	14	81,52	13
ZBot	98,88	7	98,88	3

	k = 100		k = 150	
Ackantta	97,12	10	95,68	10
Beagle	82,98	7	82,98	9
Conficker	96,26	9	96,26	9
Fujacks	80,95	10	76,19	9
Msnpass	87,72	9	75,44	10
MyDoom	98,77	3	98,77	4
SillyFDC	85,94	9	81,25	11
Sinowal	98,97	5	98,97	11
Virut	73,91	13	66,30	13
ZBot	97,75	7	97,75	9

	k = 200		k = 350	
Ackantta	94,96	11	89,93	8
Beagle	78,72	10	70,21	9
Conficker	96,26	12	81,31	17
Fujacks	76,19	9	77,78	7
Msnpass	64,91	9	57,89	7
MyDoom	96,30	3	96,30	3
SillyFDC	76,56	10	73,44	9
Sinowal	92,78	26	52,58	18
Virut	64,13	12	58,70	12
ZBot	97,75	7	97,75	7

Tabella 4.9: Clustering basato su feature statiche (super-fingerprint), ottenuto utilizzando l'algoritmo LSI

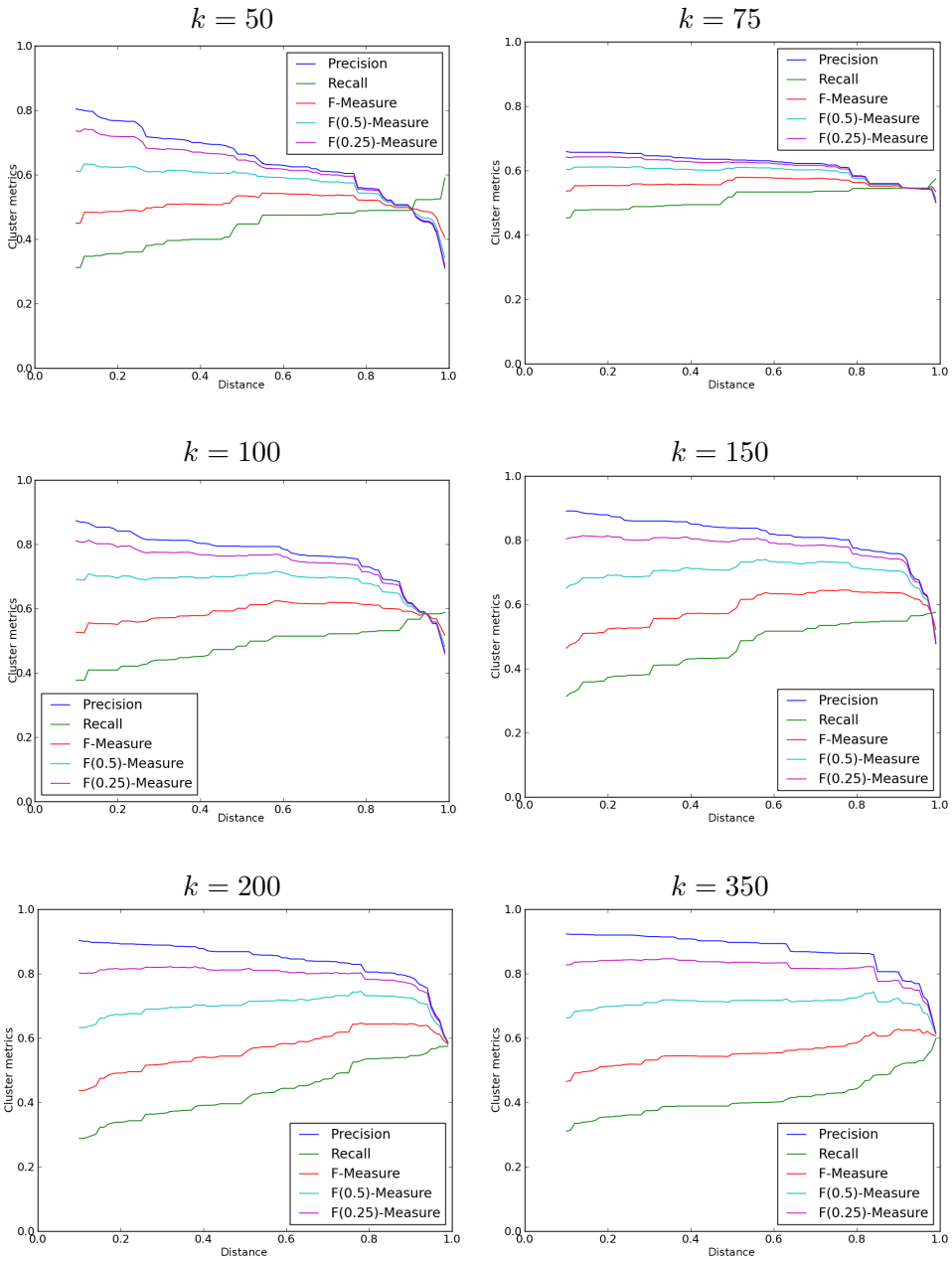

Figura 4.13: Diagrammi precision-recall relativi al clustering basato su feature statiche (super-fingerprint), ottenuto tramite l'algoritmo LSI, al variare del parametro k

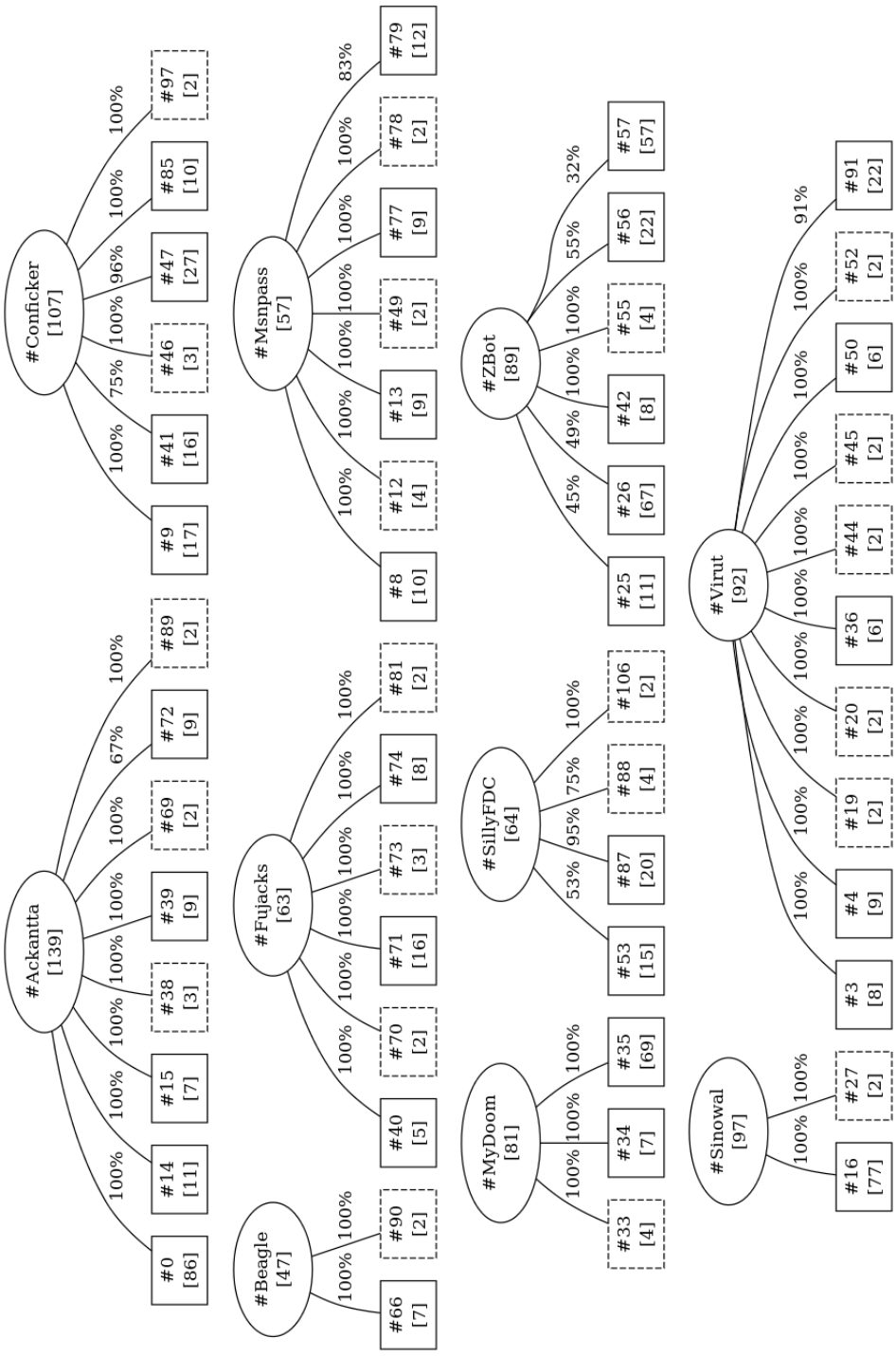

Figura 4.14: Clustering basato su feature statiche (super-fingerprint), ottenuto tramite l'algoritmo LSI con $k = 100$. Sono state rimosse alcune associazioni di scarsa entità tra cluster e famiglie allo scopo di rendere più leggibile il diagramma.

T_w	IF	ID	C
0%	1	1	-
5%	1	1	-
10%	1	2	-
20%	1	4	-
30%	-	10	-
40%	-	10	-

Tabella 4.10: Analisi delle inconsistenze tra il clustering basato su feature statiche (super-fingerprint, ottenuti), ottenuto tramite l'algoritmo LSI, e quello di riferimento

è molto bassa. Per valori di $T_w \leq 5\%$ si presenta un'unica inconsistenza forte che coinvolge quasi tutti i nodi appartenenti al grafo ed un'inconsistenza debole tra la famiglia MyDoom e i relativi tre cluster prodotti da LSI. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Per $T_w \geq 30\%$ le famiglie si mappano su un insieme di cluster strutturali definiti lasciando solo inconsistenze deboli tra le famiglie e i cluster prodotti da LSI, come rappresentato in Figura 4.14.

Questo metodo consente di raggruppare elementi molto simili dal punto di vista strutturale, ossia che condividono parti significative del CFG. Campioni di malware con stessi comportamenti ma aventi codice differente non potranno essere considerati simili poiché avranno super-fingerprint diverse. Varianti di malware che condividono solo parzialmente il CFG ricostruito saranno considerate diverse.

4.3 Confronto dei risultati

L'analisi ha evidenziato una carenza generale di consistenza per tutti i metodi presentati, dovuta principalmente alla costante presenza di poche inconsistenze forti di grosse dimensioni che coinvolgono quasi tutti i nodi del grafo. Tali inconsistenze sono dovute a relazioni poco significative causate dalla condivisione di pochi elementi. Eliminando questi archi si riducono gli effetti di bordo ottenendo un maggior numero di inconsistenze deboli o consistenze. Le prime possono essere giustificate da una maggiore specializzazione dei cluster prodotti dai nostri algoritmi rispetto alla categorizzazione basata sulle famiglie.

Le fingerprint sono utili per confrontare direttamente due campioni tra loro. Dovendo analizzare un dataset arbitrario si generano molte fingerprint, con conseguente esplosione delle dimensioni della matrice dei dati, campioni-caratteristiche. La riduzione tramite decomposizione SVD accentua la correlazione tra le fingerprint. Alcune fingerprint indipendenti potrebbero essere

mappate su un unico concetto, con conseguente aumento della similarità tra due campioni in realtà diversi. Questo problema, tra i metodi proposti, è maggiormente accentuato nel caso delle fingerprint poiché generano la matrice con dimensioni maggiori.

Il metodo basato su colore dei basic block non fornisce un clustering significativo poiché l'informazione associata è troppo generica e non consente di discriminarli. Aggiungendo le fingerprint al colore dei blocchi si riesce a specializzare l'informazione ad essi associata. La nuova matrice campioni-caratteristiche, più piccola di quella del metodo basato su fingerprint. Il clustering prodotto da questo metodo è quello più coerente con le famiglie, come confermato dalla più piccola distanza tra i sistemi di clustering ottenuta dai metodi proposti.

Le super-fingerprint sono più sensibili, rispetto a fingerprint e basic block, a modifiche del CFG. Il metodo assegna campioni che condividono rilevanti porzioni di codice allo stesso cluster, inoltre è in grado di distinguere tra le diverse tipologie e varianti di malware.

Capitolo 5

Confronto sperimentale dei cluster

In questo capitolo confrontiamo i risultati del clustering basato su feature dinamiche, descritti nel Capitolo 3, con quelli ottenuti a partire dalle feature statiche, dettagliati nel Capitolo 4. Lo scopo di questa analisi è la ricerca di un legame tra i comportamenti manifestati dai campioni e il codice che li contraddistingue.

Nella Sezione 5.2 analizziamo il clustering tra feature dinamiche e feature statiche basate su fingerprint, per poi passare alle feature statiche identificate tramite il colore dei basic block associati alle fingerprint nella Sezione 5.3 e concludere con le feature statiche basate su super-fingerprint nella Sezione 5.4. Non analizziamo le feature statiche basate sul solo colore dei basic block perché, come visto nel capitolo precedente, sono poco significative. Queste analisi sono state supportate da un'applicazione web appositamente realizzata e dettagliata in Sezione 5.1.

5.1 Un sistema web-based per il supporto all'analisi dei risultati

Abbiamo realizzato un insieme di utility di analisi, per l'elaborazione dei campioni tramite ANUBIS e DISASM, la costruzione e successive elaborazioni delle matrici campioni-caratteristiche e campioni-campioni e la creazione dei grafici. Le utility si dividono in sei categorie:

Elaborazione dei dati grezzi queste utility lavorano direttamente sui campioni che compongono il dataset e servono ad estrarre, in uno o più passi, le feature dinamiche e statiche che li caratterizzano.

- `BuildSamplesIndex.py` costruisce l'indice dei campioni;
- `ExtractUpx.py` estrae i campioni offuscati con UPX, in modo da poterli analizzare tramite DISASM;

- `SubmitToAnubis.py` invia un campione ad ANUBIS per la generazione del report contenente le informazioni da cui estrarre le feature dinamiche;
- `ExtractFeatures.py` estrae le feature dinamiche dai report XML di ANUBIS, filtrando mediante una trasformazione XSLT gli elementi di scarso interesse;
- `ExtractFingerprints.py` estrae le fingerprint dei campioni tramite DISASM;
- `DownloadVirusTotalReports.py` recupera da VirusTotal i report corrispondenti ai campioni in esame;
- `ProcessVirusTotalReports.py` elabora i report di VirusTotal estraendo le informazioni ritenute interessanti;
- `ExtractCodeSets.py` estrae le coppie di insiemi blocchi-fingerprint richieste dal primo passo dell'algoritmo di costruzione delle super-fingerprint;
- `ExtractSuperFingerprints.py` elabora gli insiemi blocchi-fingerprint di ciascun campione costruendo le super-fingerprint.

Costruzione delle matrici queste utility lavorano sulle feature dinamiche o statiche estratte dai campioni e permettono di costruire le matrici su cui andranno ad operare gli algoritmi di clustering.

- `BuildIndex.py` costruisce l'indice delle feature, necessario per definire l'insieme delle colonne della matrice dei dati;
- `BuildMatrix.py` costruisce la matrice campioni-caratteristiche, composta da tante righe quanti sono i campioni e tante colonne quante sono le feature;
- `ReduceMatrix.py` costruisce la matrice ridotta eliminando da quella completa tutte le colonne che contengono un solo elemento non nullo.

Clustering gerarchico tradizionale queste utility lavorano sulle matrici campioni-caratteristiche e costruiscono il clustering gerarchico tradizionale dei campioni.

- `ComputeJaccardDistances` calcola la matrice triangolare campioni-campioni contenente le distanze Jaccard;
- `ComputeCosineDistances` calcola la matrice triangolare campioni-campioni contenente le distanze coseno;
- `ComputeTanimotoDistances` calcola la matrice triangolare campioni-campioni contenente le distanze Tanimoto;
- `ComputeDiceDistances` calcola la matrice triangolare campioni-campioni contenente le distanze Dice;

- `ComputeClusters.py` costruisce il clustering gerarchico di tipo agglomerativo a partire dalle matrici campioni-campioni, usando, come criterio di collegamento, la media delle distanze tra i campioni dei vari cluster. Genera inoltre il dendrogramma corrispondente.

Clustering LSH queste utility lavorano sulle matrici campioni-caratteristiche e costruiscono il clustering dei campioni utilizzando l'algoritmo LSH.

- `FindPrimeNumbers.py` calcola i primi n numeri primi;
- `BuildHashMatrix.py` costruisce la matrice campioni-hash a partire dalla matrice campioni-caratteristiche;
- `LSH.py` calcola gli hash lsh a partire dalla matrice campioni-hash e li confronta, costruisce la lista delle coppie candidate di campioni simili;
- `FilterJaccard.py` calcola la similarità Jaccard effettiva tra i campioni candidati ed elimina tutte le coppie corrispondenti a falsi positivi;
- `ComputeClusters.py` costruisce il clustering gerarchico di tipo agglomerativo a partire dalla lista delle coppie di campioni simili, utilizzando, come criterio di collegamento, la distanza tra gli elementi più vicini. Genera inoltre il dendrogramma corrispondente.

Clustering LSI queste utility lavorano sulle matrici campioni-caratteristiche e costruiscono il clustering dei campioni utilizzando l'algoritmo LSI.

- `ComputeSimilarities.py` applica la decomposizione SVD troncata al valore k alla matrice campioni-caratteristiche e costruisce la matrice triangolare campioni-campioni contenente le similarità coseno;
- `ComputeClusters.py` trasforma la matrice delle similarità nella matrice delle distanze e la utilizza per costruire il clustering gerarchico di tipo agglomerativo utilizzando come criterio di collegamento la media delle distanze. Genera inoltre il dendrogramma corrispondente.

Analisi dei risultati queste utility analizzano le matrici campioni-caratteristiche, il risultato della decomposizione SVD, calcolano le metriche per la valutazione dei clustering risultanti e creano gli opportuni diagrammi.

- `AnalyzePlot_DataMatrix.py` analizza la matrice campioni-caratteristiche contando il numero di colonne con un solo elemento non nullo e riportando il risultato in un istogramma per meglio visualizzare l'impatto che avrebbe la loro rimozione sulla matrice in esame;
- `AnalyzePlot_Variance.py` costruisce il diagramma che rappresenta la distribuzione della varianza nella matrice Σ, ottenuta tramite decomposizione SVD a partire dalla matrice campioni-caratteristiche;
- `AnalyzePlot_PrecisionRecall.py` costruisce il diagramma precision-recall di un sistema di clustering;
- `AnalyzePlot_ClusteredTotal.py` costruisce il diagramma di copertura di un sistema di clustering.

Per l'implementazione delle matrici sparse e della decomposizione SVD ci si è appoggiati sulla libreria divisi2 [34], mentre per gli algoritmi di clustering gerarchico sulla libreria scipy [35]. I dati analizzati sono stati raccolti con l'ausilio di un database per facilitarne le successive consultazioni.

Inoltre, per rendere più agevole l'analisi dei risultati, abbiamo realizzato un'apposita applicazione web in PHP. Tramite questa interfaccia è possibile valutare le relazioni tra i cluster basati su analisi dinamica, analisi statica e le famiglie, il contenuto di ciascun cluster e le inconsistenze. Se ne riportano alcuni screenshot.

5.2 Feature dinamiche vs. feature statiche (basate su fingerprint)

Confrontiamo il sistema di cluster ottenuto dalle feature dinamiche con quello che usa come feature statiche le fingerprint. I risultati dell'analisi eseguita nei capitoli precedenti sono riassunti in Tabella 5.1. La copertura è calcolata come definito in (3.3).

Analizzando le inconsistenze tra il clustering prodotto dalle feature dinamiche, considerato come riferimento, e quello prodotto dalle feature statiche applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7299$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze, al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 5.2. La consistenza tra i due clustering è molto bassa, in accordo al valore della distanza prima riportato. Per bassi valori di T_w si presenta un'unica grande inconsistenza forte che coinvolge quasi tutti i nodi appartenenti al grafo e quattro consistenze che permettono di associare i

Figura 5.1: Screenshot dell'applicazione Web per l'analisi dei cluster. L'applicazione consente di scegliere il clustering, specificandone tipo e categoria, da relazionare alle famiglie e visualizza una tabella riassuntiva che ne evidenzia la composizione. Permette, inoltre, di selezionare un cluster per analizzarne il contenuto.

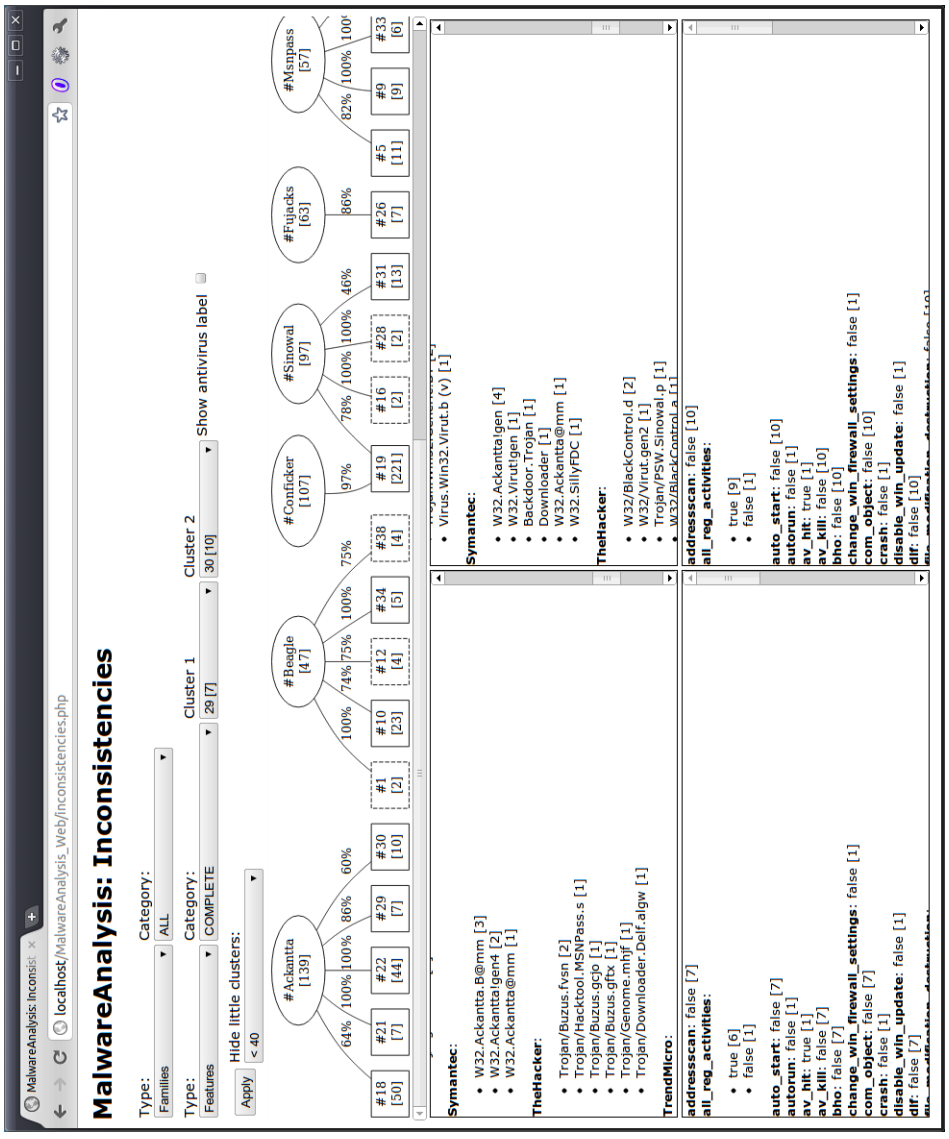

Figura 5.2: Screenshot dell'applicazione Web per l'analisi delle inconsistenze. L'applicazione consente di scegliere tra due clustering diversi specificandone tipo e categoria. Permette, inoltre, di selezionare due cluster da confrontare tra loro visualizzando i dettagli dei campioni contenuti quali le etichette assegnate dagli antivirus (sopra) e le informazioni estratte dai report di ANUBIS (sotto).

Figura 5.3: Screenshot dell'applicazione Web per l'analisi delle inconsistenze. L'applicazione consente di eliminare le relazioni poco significative, al di sotto di una soglia scelta dall'utente, per migliorare la visibilità del grafico.

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	Feature dinamiche		Feature statiche	
Ackantta	99,28	10	99,28	10
Beagle	89,36	10	85,11	8
Conficker	99,07	2	98,13	5
Fujacks	90,48	8	82,54	8
Msnpass	85,96	7	80,70	6
MyDoom	95,06	1	98,77	2
SillyFDC	89,06	9	84,38	8
Sinowal	96,91	6	95,88	9
Virut	97,83	2	82,61	15
ZBot	100	5	97,75	3

Tabella 5.1: Classificazioni indotte dai metodi basati su analisi dinamica (sinistra) e analisi statica a partire dalle fingerprint (destra)

T_w	IF	ID	C
0%	1	-	4
5%	1	-	4
10%	1	-	4
20%	2	1	6
30%	1	3	9
40%	3	4	9

Tabella 5.2: Analisi delle inconsistenze tra il clustering basato su feature dinamiche e quello su feature statiche (fingerprint)

comportamenti a parti di codice condiviso. Tuttavia queste ultime sono trascurabili rispetto ai nodi coinvolti nell'inconsistenza forte. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Tuttavia, nonostante la maggior separazione del grafo, più della metà dei campioni del dataset sono coinvolti in almeno un'inconsistenza forte.

5.3 Feature dinamiche vs. feature statiche (basate su basic block identificati tramite colore associato alle fingerprint)

Confrontiamo il sistema di cluster ottenuto dalle feature dinamiche con quello che usa come feature statiche i blocchi identificati tramite colore associato alle fingerprint. I risultati dell'analisi eseguita nei capitoli precedenti sono riassunti in Tabella 5.3. La copertura è calcolata come definito in (3.3).

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	Feature dinamiche		Feature statiche	
Ackantta	99,28	10	96,40	10
Beagle	89,36	10	80,85	7
Conficker	99,07	2	96,26	9
Fujacks	90,48	8	71,43	9
Msnpass	85,96	7	77,19	11
MyDoom	95,06	1	98,77	4
SillyFDC	89,06	9	75,00	12
Sinowal	96,91	6	98,97	7
Virut	97,83	2	67,39	13
ZBot	100	5	98,88	9

Tabella 5.3: Classificazioni indotte dai metodi basati su analisi dinamica (sinistra) e analisi statica a partire dai basic block identificati tramite colore associato alle fingerprint (destra)

T_w	IF	ID	C
0%	1	1	2
5%	1	1	2
10%	1	2	2
20%	3	3	3
30%	3	3	7
40%	4	4	9

Tabella 5.4: Analisi delle inconsistenze tra il clustering basato su feature dinamiche e quello su feature statiche (basic block identificati tramite colore associato alle fingerprint)

Analizzando le inconsistenze tra il clustering prodotto dalle feature dinamiche, considerato come riferimento, e quello prodotto dalle feature statiche applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7516$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze, al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 5.4. La consistenza tra i due clustering è molto bassa, in accordo al valore della distanza prima riportato. Per bassi valori di T_w si presenta un'unica grande inconsistenza forte che coinvolge quasi tutti i nodi appartenenti al grafo, un'inconsistenza debole e due consistenze che permettono di associare i comportamenti a parti di codice condiviso. Tuttavia queste ultime sono trascurabili rispetto ai nodi coinvolti nell'inconsistenza forte. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Tuttavia,

Famiglia	Copertura [%]	Cluster	Copertura [%]	Cluster
	Feature dinamiche		Feature statiche	
Ackantta	99,28	10	97,12	10
Beagle	89,36	10	82,98	7
Conficker	99,07	2	96,26	9
Fujacks	90,48	8	80,95	10
Msnpass	85,96	7	87,72	9
MyDoom	95,06	1	98,77	3
SillyFDC	89,06	9	85,94	9
Sinowal	96,91	6	98,97	5
Virut	97,83	2	73,91	13
ZBot	100	5	97,75	7

Tabella 5.5: Classificazioni indotte dai metodi basati su analisi dinamica (sinistra) e analisi statica a partire dalle super-fingerprint (destra)

nonostante la maggior separazione del grafo, più della metà dei campioni del dataset sono coinvolti in almeno un'inconsistenza forte. La distanza dal clustering basato sulle feature dinamiche è maggiore rispetto a quella tra feature dinamiche e feature statiche basate su fingerprint. Questo motiva le maggiori inconsistenze forti e il più alto numero di campioni coinvolti in tali relazioni.

5.4 Feature dinamiche vs. feature statiche (basate su super-fingerprint)

Confrontiamo il sistema di cluster ottenuto dalle feature dinamiche con quello che usa come feature statiche le super-fingerprint. I risultati dell'analisi eseguita nei capitoli precedenti sono riassunti in Tabella 5.5. La copertura è calcolata come definito in (3.3).

Analizzando le inconsistenze tra il clustering prodotto dalle feature dinamiche, considerato come riferimento, e quello prodotto dalle feature statiche applicando il metodo descritto nell'Appendice B. Il valore della distanza (B.1) tra questi sistemi di clustering è pari a $0,7706$, ci aspettiamo quindi un livello di inconsistenza elevato.

Approfondendo l'analisi determiniamo il numero e il tipo di inconsistenze, al variare della soglia T_w sui pesi degli archi (B.2). I risultati sono riportati in Tabella 5.6. La consistenza tra i due clustering è molto bassa, in accordo al valore della distanza precedentemente riportato. Per bassi valori di T_w si presenta un'unica grande inconsistenza forte che coinvolge quasi tutti i nodi appartenenti al grafo, un'inconsistenza debole e due consistenze che permettono di associare i comportamenti a parti di codice condiviso. Tuttavia queste ultime sono trascurabili rispetto ai nodi coinvolti nell'inconsistenza

T_w	IF	ID	C
0%	1	1	2
5%	1	1	2
10%	1	2	3
20%	1	3	4
30%	1	5	6
40%	3	5	6

Tabella 5.6: Analisi delle inconsistenze tra il clustering basato su feature dinamiche e quello su feature statiche (super-fingerprint)

forte. Aumentando il valore della soglia T_w si riduce l'impatto delle relazioni poco significative e il grafo si divide in più componenti connesse. Tuttavia, nonostante la maggior separazione del grafo, circa tre quarti dei campioni del dataset sono coinvolti in almeno un'inconsistenza forte. La distanza dal clustering basato su feature dinamiche è superiore rispetto ai metodi descritti nelle sezioni precedenti. Una giustificazione a questo risultato è data dal fatto che il sottografo associato alle super-fingerprint, avendo dimensione maggiore rispetto a quelli corrispondenti alle feature statiche impiegate nelle sezioni precedenti, potrebbe manifestare più di un comportamento e quindi mapparsi su più cluster relativi alle feature dinamiche.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi abbiamo analizzato sperimentalmente alcuni metodi di clustering basati su *feature dinamiche e statiche*, estratte, rispettivamente, tramite gli strumenti ANUBIS e DISASM, senza ulteriori interventi da parte dell'utente. Abbiamo inoltre proposto due nuove feature statiche, *colore dei blocchi associato alle rispettive fingerprint* e *super-fingerprint*, per rappresentare l'informazione strutturale ad un livello di dettaglio inferiore, ossia considerando i basic block, e superiore, considerando i sottografi del CFG di dimensione superiore a 10, rispetto alle fingerprint. Per ciascun metodo abbiamo ricavato sperimentalmente i valori dei parametri che consentono di ottenere i risultati migliori rispetto al clustering di riferimento basato sulle famiglie. Infine, abbiamo cercato una possibile correlazione tra i cluster prodotti a partire dalle feature dinamiche e quelli ottenuti dalle feature statiche, evidenziandone le inconsistenze.

Abbiamo costruito un dataset contenente campioni appartenenti alle famiglie *Ackantta*, *Beagle*, *Conficker*, *Fujacks*, *Msnpass*, *MyDoom*, *SIllyFDC*, *Sinowal*, *Virut* e *ZBot*. Su questo dataset abbiamo analizzato i metodi di clustering implementati durante il nostro lavoro.

Le matrici *campioni-feature* registrano l'occorrenza, o meno, delle caratteristiche per ciascun campione. Come era ragionevole attendersi, gli algoritmi di clustering tradizionali (quali ad esempio quelli di clustering gerarchico agglomerativo basato sulle distanze Jaccard e coseno) non sono in grado di operare con matrici di grandi dimensioni, poiché richiedono tempi di esecuzione molto elevati e forniscono risultati scarsamente significativi. Dato lo scarso fattore di riempimento di queste matrici, l'algoritmo probabilistico LSH non ottiene risultati ottimali a causa della scarsa condivisione di caratteristiche da parte dei campioni. Abbiamo constatato che l'algoritmo LSI, invece, fornisce risultati interessanti negli ambiti applicativi considerati. Lo *spazio dei concetti*, ossia lo spazio vettoriale di dimensioni minori rispetto a quello delle feature, riassume in modo efficace l'informazione associata alle feature dinamiche e statiche. I cluster ottenuti nei nostri esperimenti

rispecchiano la suddivisione in famiglie, nonostante la presenza di alcune inconsistenze, come evidenziato soprattutto dal clustering basato sulle feature dinamiche. Tuttavia, non sempre la riduzione delle dimensioni associa in modo corretto le caratteristiche, comportando talvolta l'unione di feature tra loro non correlate (es., le feature dinamiche "scrittura sul file X" e "apertura del socket Y") e conseguentemente causando la creazione di nuove relazioni tra campioni che non condividono una delle due feature, ma accomunate dallo stesso concetto.

Nelle matrici campioni-caratteristiche di grandi dimensioni l'effetto negativo dell'algoritmo LSI, causato dall'unione di feature non correlate, è maggiormente accentuato. Infatti, i metodi basati su feature statiche sono più soggetti al problema rispetto a quello basato su feature dinamiche. Questi ultimi metodi forniscono risultati meno significativi rispetto al metodo basato su feature dinamiche, come giustificato dal maggior valore della distanza tra i due sistemi di clustering. Le feature statiche basate sul colore associato alle fingerprint producono risultati analoghi alle fingerprint, ma hanno il vantaggio di essere in numero significativamente inferiore. Le feature statiche basate su super-fingerprint sono più selettive rispetto alle fingerprint, in quanto richiedono che i campioni condividano parti più grandi del CFG del programma. Per queste caratteristiche, due campioni che si differenziano per una piccola variazione nel codice potrebbero essere considerati completamente diversi.

Basandosi sui cluster prodotti dall'algoritmo LSI concludiamo che non ci è possibile individuare una relazione che consenta di legare i risultati ottenuti a partire dall'analisi dinamica con quelli dell'analisi statica. Tale problema è dovuto al fatto che le associazioni *feature dinamiche-concetti dinamici* sono profondamente diverse da quelle *feature statiche-concetti statici*, generando così alcune inconsistenze.

Un possibile miglioramento al clustering basato su feature dinamiche potrebbe risultare da un'elaborazione dei report di ANUBIS che consenta di portare le azioni rilevate ad un più alto livello di astrazione. Ad esempio, le feature dinamiche "scrittura nel file `C:\Temp\X`" e "scrittura nel file `C:\Temp\Y`" potrebbero essere trattate come una nuova feature "scrittura in un file temporaneo". Questo accorgimento dovrebbe consentire, trascurando dettagli quali il nome di un file temporaneo generato casualmente, di ridurre il numero complessivo delle caratteristiche e di conseguenza della frammentazione dei dati.

In generale, tecniche più efficienti per eliminare le informazioni ridondanti, e conseguentemente ridurre le dimensioni delle matrici dei dati, potrebbero rivelarsi utili per migliorare i risultati del clustering. Ad esempio la trasformazione dell'output di DISASM in feature basate sul colore dei blocchi associato alle fingerprint che lo attraversano, proposta e sperimentata in questo lavoro di tesi, si è rivelata efficace migliorando la qualità del

clustering prodotto e riducendo notevolmente i tempi totali di elaborazione grazie ad una rappresentazione più compatta dei dati.

Una limitazione all'applicabilità delle tecniche di analisi statica è dovuta all'incapacità di DISASM, e altri strumenti di analisi, di gestire campioni packati. L'integrazione di questo strumento con sistemi di unpacking, come quelli descritti in [27, 28, 29], consentirebbe di elaborare un maggior numero di campioni.

Appendice A

Clustering

Lo scopo di questo appendice è quello di fornire una panoramica dettagliata degli algoritmi di clustering tradizionali (Sezione A.1) e delle metriche di valutazione *precision*, *recall* ed *f-measure* (Sezione A.2) utilizzate in questa tesi. Descrive, inoltre, la *curse of dimensionality* presentando due algoritmi che permettono di ridurre le dimensioni di uno spazio vettoriale (Sezione A.3).

A.1 Metodi di clustering tradizionali

Le tecniche di clustering sono volte alla selezione e al raggruppamento di elementi omogenei in un insieme di dati e si basano sul concetto di *distanza* tra elementi. La bontà del partizionamento prodotto è fortemente influenzata dalla metrica scelta, e quindi da come è calcolata la distanza. In letteratura esistono vari tipi di algoritmi di clustering che si dividono in *partitivi* e *gerarchici*. I primi richiedono che sia prefissato il numero di gruppi della partizione risultato, mentre i secondi non hanno questo requisito e per tale motivo ci si focalizza su questi ultimi.

Esistono due tipi di strategie per il clustering gerarchico:

Agglomerativo si tratta di un approccio *bottom-up* (dal basso verso l'alto) dove si parte dall'inserimento di ciascun elemento in un cluster differente e si procede quindi all'accorpamento graduale fino ad ottenere un unico cluster;

Divisivo si tratta di un approccio *top-down* (dall'alto verso il basso) dove tutti gli elementi si trovano inizialmente in un singolo cluster che viene via via suddiviso ricorsivamente in sotto-cluster.

Il risultato di un clustering gerarchico è rappresentato con un *dendrogramma* il cui esempio è mostrato in Figura A.1.

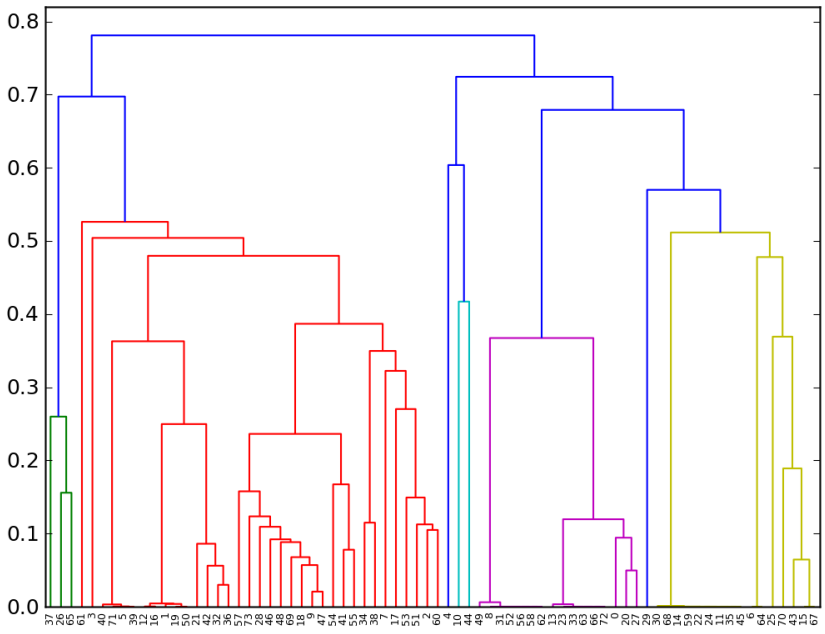

Figura A.1: Esempio di dendrogramma

Per decidere quali cluster devono essere combinati (approccio agglomerativo) o quale deve essere suddiviso (approccio divisivo) è necessario definire una misura di distanza tra coppie di elementi. La scelta di una metrica influenza la forma dei cluster, poiché alcuni elementi possono essere più "vicini" utilizzando una distanza e più "lontani" utilizzandone un'altra. Le metriche utilizzate nel seguito sono le seguenti:

Distanza Jaccard misura la distanza tra insiemi, definita come la percentuale degli elementi non comuni.

$$D_J(A,B) = 1 - J(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{A.1}$$

Distanza coseno è legata al coseno dell'angolo θ compreso tra i due vettori. Risulta utile per determinare se due vettori puntano nella stessa direzione, in tal caso il valore della distanza sarà 0.

$$D_{cos}(A,B) = 1 - cos(\theta) = 1 - \frac{A \cdot B}{\|A\| \, \|B\|} \tag{A.2}$$

Distanza Tanimoto è un'estensione della distanza coseno che migliora la precisione nel caso di attributi binari, fornendo l'equivalente della distanza Jaccard.

$$D_T(A,B) = 1 - T(A,B) = 1 - \frac{A \cdot B}{\|A\|^2 + \|B\|^2 - A \cdot B} \tag{A.3}$$

Distanza Dice è una misura della distanza correlata al coefficiente Jaccard tramite le relazioni $D = \frac{2J}{1+J}$ e $J = \frac{D}{2-D}$.

$$D_D(A,B) = 1 - D(A,B) = 1 - \frac{2|A \cap B|}{|A| + |B|} \tag{A.4}$$

Nella maggior parte dei metodi di clustering gerarchico si fa uso di un criterio di collegamento che specifica la distanza tra cluster come funzione di quella tra gli elementi nei cluster stessi. I criteri comunemente utilizzati sono:

Single linkage la similarità tra i due cluster è basata sui due punti più simili (vicini) nei differenti cluster. È determinata da una coppia di punti.

$$D_{single}(A,B) = min\{d(x,y) : x \in A, y \in B\} \tag{A.5}$$

Questo approccio è suscettibile a rumore ed outliers.

Complete linkage la similarità tra i due cluster è basata sui due punti meno simili (distanti) nei differenti cluster. È determinata da una coppia di punti nei cluster.

$$D_{complete}(A,B) = max\{d(x,y) : x \in A, y \in B\} \tag{A.6}$$

Questo approccio è meno suscettibile a rumore ed outliers, ma tende a frammentare i cluster grandi.

Average linkage la similarità tra due cluster è la media delle distanze a coppie (*pairwise*) tra tutti i singoli elementi appartenenti ai due cluster.

$$D_{average}(A,B) = \frac{1}{|A||B|} \sum_{x \in A, y \in B} d(x,y) \tag{A.7}$$

Sono disponibili varie librerie, nei più comuni linguaggi di programmazione, che implementano gli algoritmi di clustering gerarchico agglomerativo e le funzioni per il calcolo delle distanze appena descritte.

A.2 Valutazione dei metodi di clustering

Stimare la qualità dei risultati prodotti da un algoritmo di clustering è un compito difficile. È possibile quantificare il numero di cluster, il numero medio di campioni per ciascuno di essi, la copertura fornita, valutare la distanza tra coppie di cluster oppure ispezionarne manualmente il contenuto verificando che i campioni contenuti siano simili.

Il modo migliore per dimostrare la correttezza di un algoritmo di clustering è confrontare il suo output con un clustering di riferimento che per i malware, purtroppo, non esiste.

Metriche di valutazione

Un algoritmo di clustering può generare dei *falsi positivi* e dei *falsi negativi*. Per valutare la qualità dei cluster prodotti dagli algoritmi li si confrontano con un clustering scelto come riferimento utilizzando le seguenti metriche:

Precision ha lo scopo di valutare quanto un algoritmo di clustering riesce a distinguere campioni diversi. In altre parole, la precision rileva la capacità di un algoritmo di clustering di assegnare campioni diversi a cluster diversi. Intuitivamente, un algoritmo è "preciso" se ogni cluster contiene un solo elemento. Più formalmente la precision è definita dall'equazione (A.8).

$$P_i = \max\left\{|C_i \cap T_1|, |C_i \cap T_2|, \ldots, |C_i \cap T_t|\right\}, \qquad \forall C_i \in C$$
$$P = \frac{\sum_{i=1}^{c} P_i}{n} \tag{A.8}$$

dove $T = T_1, T_2, \ldots, T_t$ è il clustering di riferimento mentre $C = C_1, C_2, \ldots, C_c$ è quello prodotto dall'algoritmo in esame.

Recall ha lo scopo di valutare quanto un algoritmo di clustering riconosce campioni simili. In altre parole, la recall rileva la capacità di un algoritmo di assegnare campioni dello stesso tipo allo stesso cluster. Più formalmente la recall è definita dall'equazione (A.9).

$$R_i = \max\left\{|C_1 \cap T_i|, |C_2 \cap T_i|, \ldots, |C_c \cap T_i|\right\}, \qquad \forall T_i \in T$$
$$R = \frac{\sum_{i=1}^{t} R_i}{n} \tag{A.9}$$

dove $T = T_1, T_2, \ldots, T_t$ è il clustering di riferimento mentre $C = C_1, C_2, \ldots, C_c$ è quello prodotto dall'algoritmo in esame.

Esiste un trade-off tra le due metriche. L'algoritmo che crea un cluster per ogni campione raggiunge la precisione ottima, ma la peggior recall. Quello che combina tutti i campioni in un solo cluster, invece, raggiunge la recall ottima ma la peggior precision. Un buon algoritmo dovrebbe massimizzare sia la precision che la recall.

Precision e recall sono talvolta combinate in un unico indice per fornire una sola misura del sistema. Questo indice, chiamato *F-measure*, è la media armonica pesata di precision e recall e assume valori compresi tra 1, nel caso migliore, e 0, nel caso peggiore.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{A.10}$$

La formula generale per la *F-measure*, valida per valori di $\beta > 0$, permette di assegnare pesi diversi alle due metriche. Per $\beta > 1$ si assegna peso maggiore alla recall, mentre per $0 < \beta < 1$ si pone maggiore enfasi alla precision.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

A.3 "Curse of dimensionality"

Gli algoritmi di clustering tradizionali potrebbero dare pessimi risultati quando il numero n delle dimensioni diventa significativo. In tal caso l'elaborazione dei dati si complica, per diverse ragioni:

- incremento esponenziale del numero dei possibili valori;
- crescita del rumore che affligge i dati;
- perdita di significato del concetto di *distanza*, poiché la distinzione tra il punto più vicino e quello più lontano diventa priva di significato:

$$\lim_{n \to \infty} \frac{dist_{max} - dist_{min}}{dist_{min}} \to 0$$

- correlazione tra le dimensioni. Dato l'elevato numero delle variabili, è altamente probabile che alcune di esse risultino correlate. Potrebbero quindi esistere cluster in sottospazi affini orientati arbitrariamente, rilevanti o fuorvianti ai fini dell'analisi.

Questo problema è noto in letteratura con il nome di *curse of dimensionality* (letteralmente, "maledizione delle grandi dimensioni"). Le strategie più diffuse per affrontare questa problematica sono:

- Ricerca di un sottoinsieme di k variabili rilevanti, dove $k \ll n$. Alcune delle variabili possono essere completamente scorrelate tra loro, mentre altre ridondanti.

  Non esistono metodi esatti per la scelta delle variabili. Nella pratica ci si affida a tecniche di ricerca euristiche.

- Trasformazione delle n variabili originali in un nuovo insieme di k variabili, dove $k \ll n$. Le variabili osservate vengono sostituite con un insieme ridotto di variabili più significative, chiamate *funzioni base*, *fattori*, *variabili latenti*, *componenti principali* e così via, a seconda del metodo utilizzato per ottenerle.

  L'*analisi delle componenti principali* [30], nel seguito PCA ("Principal Component Analysis"), è una tecnica classica che consente di trasformare l'insieme originale delle n variabili in un secondo di k variabili formate da combinazioni lineari delle precedenti.

  PCA ha due importanti vantaggi:

  – le prime componenti estratte contengono la maggior parte della varianza dei dati, di conseguenza sono le più significative in termini di informazione;

– le componenti estratte sono ortogonali tra loro, facilitandone così l'interpretazione.

I dati analizzati in questo lavoro soffrono del problema appena descritto. Per poterli clusterizzare è necessario ridurne il numero delle dimensioni. A questo scopo vengono utilizzati gli algoritmi LSH e LSI, frequentemente impiegati nell'ambito dell'information retrieval.

A.3.1 LSH: Locality Sensitive Hashing

LSH [31] è un algoritmo che consente di effettuare una riduzione probabilistica delle dimensioni dei dati. L'idea alla base consiste nel calcolo di un hash degli elementi in modo tale che elementi simili, in accordo all'indice Jaccard (A.11), hanno una probabilità di collisione più elevata rispetto agli elementi diversi.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{A.11}$$

Per ottenere questo risultato si sceglie una famiglia di funzioni di hash H tali che $\mathcal{P}[h(a) = h(b)] = similarity(a,b)$ per ogni coppia di elementi a e b, e h scelta casualmente in modo uniforme in H. Una funzione di hash $h \in H$ impone un ordinamento casuale sull'insieme di tutti gli elementi.

Definendo l'hash LSH come $lsh(a) = h_1(a), h_2(a), \ldots, h_k(a)$, con k funzioni di hash scelte casualmente in modo uniforme ed indipendente da H, si ottiene $\mathcal{P}[lsh(a) = lsh(b)] = similarity(a,b)^k$.

Per matrici molto grandi la generazione di permutazioni casuali è inefficiente. Al loro posto si possono utilizzare funzioni lineari nella forma $h(x) = c_1 x + c_2 \mod P$ dove c_1 e c_2 sono costanti arbitrarie e P è il più piccolo numero primo maggiore del numero totale degli elementi nell'insieme, come descritto in [32].

Questo algoritmo richiede una soglia di similarità T_s, un numero di funzioni k per ogni hash LSH e un numero di iterazioni l. Si inizializza l'insieme delle coppie candidate S all'insieme vuoto. In seguito, per ogni iterazione, si scelgono k funzioni di hash $h_1, \ldots, h_k$ casualmente da H; si calcola la firma $lsh(a)$ per ogni elemento a dell'insieme; se due elementi a, b hanno la stessa firma lsh, $lsh(a) = lsh(b)$, nella stessa iterazione vengono inseriti nell'insieme delle coppie candidate S.

Poiché LSH fornisce solo un'approssimazione, S può contenere falsi positivi, ossia coppie di campioni non simili. Per rimuoverli si calcola la similarità $J(a,b), \forall (a,b) \in S$ e si scarta la coppia se $J(a,b) < T_s$. Un ordinamento delle coppie rimaste per similarità consente di produrre un clustering agglomerativo di tipo single linkage.

A.3.2 LSI: Latent Semantic Indexing

LSI [33] è una tecnica utilizzata nell'elaborazione del linguaggio naturale per evidenziare le relazioni tra un insieme di documenti e i termini contenuti.

A partire dai testi si costruisce la matrice *documenti-termini* $\mathbf{X_{m,n}}$ il cui elemento $x_{i,j}$ rappresenta l'occorrenza del termine j nel documento i:

$$\mathbf{X_{m,n}} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,n} \end{bmatrix}$$

La riga i-esima di questa matrice sarà il vettore $\mathbf{d_i}$ corrispondente ad un documento, che fornisce la sua relazione con ciascun termine. Allo stesso modo, la colonna j-esima della matrice sarà il vettore $\mathbf{t_j}$ corrispondente ad un termine, che fornisce la sua relazione con ciascun documento.

$$\mathbf{d_i} = \begin{bmatrix} x_{i,1} & x_{i,2} & \dots & x_{i,n} \end{bmatrix} \qquad \mathbf{t_j} = \begin{bmatrix} x_{1,j} \\ x_{2,j} \\ \vdots \\ x_{m,j} \end{bmatrix}$$

Dopo aver costruito la matrice delle occorrenze, LSI ne cerca un'approssimazione di rango inferiore. A questo scopo viene applicata una tecnica matematica, chiamata *decomposizione ai valori singolari*, nel seguito SVD ("Singular Value Decomposition"), per ridurre il numero delle colonne preservando la struttura di similarità tra le righe.

La decomposizione SVD trasforma la matrice $\mathbf{X}$ nel prodotto di tre matrici

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V^T}$$

dove $\mathbf{U}$ e $\mathbf{V}$ sono ortogonali e $\mathbf{\Sigma}$ diagonale. Gli elementi di $\mathbf{\Sigma}$ sono detti *valori singolari*, mentre le matrici $\mathbf{U}$ e $\mathbf{V}$ contengono i *vettori singolari* sinistri e destri.

Selezionando i primi k valori singolari, ed i corrispondenti vettori singolari da $\mathbf{U}$ e $\mathbf{V}$, si ottiene la miglior approssimazione di rango k della matrice $\mathbf{X}$ nel senso dei minimi quadrati (A.12). Questa approssimazione troncata della matrice documenti-termini porta alla costruzione di un nuovo spazio vettoriale chiamato *spazio dei concetti*.

$$\mathbf{X_k} = \mathbf{U_k}\mathbf{\Sigma_k}\mathbf{V_k^T} \tag{A.12}$$

La riga i-esima della matrice $\mathbf{X_k}$ sarà il vettore $\hat{\mathbf{d}}_\mathbf{i}$ contenente k elementi, ciascuno dei quali fornisce la relazione tra il documento i e ciascun concetto. Allo stesso modo, la colonna j-esima sarà il vettore $\hat{\mathbf{t}}_\mathbf{j}$ che fornisce la relazione tra il concetto j e ciascun documento.

Nella maggior parte delle applicazioni la dimensionalità k è molto minore del numero delle colonne nella matrice documenti-termini. In molti casi la dimensione ottima è intrinseca nel dominio e deve essere determinata sperimentalmente.

La conseguenza della riduzione del rango è che alcune dimensioni vengono combinate e dipendono da più di un termine. Un esempio si può vedere in (A.13).

$$(cane), (gatto), (acqua) \rightarrow (1,28 \cdot cane + 0,34 \cdot gatto), (acqua) \qquad \text{(A.13)}$$

Il nuovo spazio dei concetti può essere utilizzato per confrontare i documenti (clustering dei dati) e per individuare relazioni tra i termini. A tal fine si procede al calcolo della matrice *documenti-documenti* contenente le distanze coseno (A.2) tra ogni coppia di elementi. In seguito è possibile applicare gli algoritmi di clustering gerarchico di tipo agglomerativo.

La principale limitazione di LSI è data dalla difficoltà di interpretare i concetti. I risultati possono essere giustificati a livello matematico, ma normalmente non hanno un significato esplicito nel dominio applicativo originario.

Appendice B

Valutazione delle inconsistenze

Due sistemi di clustering possono essere confrontati in modo grafico e quantitativo. Il metodo proposto in [6] consente di valutare le (in)consistenze di tipo semantico-strutturale tra due sistemi di naming e, più in generale, tra due insiemi di cluster.

Distanza tra due sistemi di clustering

La *distanza tra due sistemi di clustering* $D(C_1, C_2)$ fornisce un'idea sul livello globale di inconsistenza, ed è definita come la distanza media tra i cluster che costituiscono i due insiemi:

$$D(C_1, C_2) = \frac{1}{2}\left(\frac{\sum_{c \in C_1} \delta(c, C_2)}{|C_1|} + \frac{\sum_{c \in C_2} \delta(c, C_1)}{|C_2|}\right) \tag{B.1}$$

dove $\delta(c, C') = \min_{c' \in C'} D_J(c, c'), \forall c \in C$ è la minima distanza Jaccard tra il cluster $c \in C$ ed ogni altro cluster appartenente a C'.

Ricerca delle (in)consistenze

La rappresentazione delle relazioni tra i cluster appartenenti ai due insiemi avviene tramite un grafo non orientato bipartito, in cui le due partizioni corrispondono agli insiemi di cluster in esame, i nodi ai cluster e gli archi indicano la percentuale dei campioni condivisi tra i due cluster.

Il problema della ricerca delle (in)consistenze può essere ricondotto all'analisi delle componenti connesse del grafo. Una volta estratte è necessario analizzarle e stabilirne la natura:

Inconsistenza forte (IF) la componente connessa contiene più di un cluster per ogni partizione. Tale inconsistenza include tutti i cluster che

fanno parte della componente connessa, poiché i cluster condividono alcuni campioni senza essere sottoinsiemi gli uni degli altri. Un esempio è visibile in Figura B.1 a sinistra.

Inconsistenza debole (ID) la componente connessa contiene solamente un cluster per una partizione e tutti gli altri cluster appartenenti alla componente connessa ne costituiscono un sottoinsieme. Un esempio è visibile in Figura B.1 al centro.

Consistenza (C) la componente connessa è costituita da due cluster che contengono esattamente gli stessi campioni. Un esempio è visibile in Figura B.1 a destra.

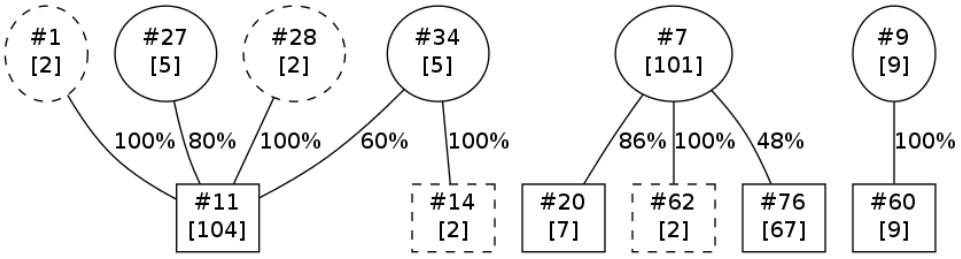

Figura B.1: Esempi di inconsistenza forte (sinistra), inconsistenza debole (centro) e consistenza (destra)

Le relazioni tra i cluster possono essere caratterizzate, in base al numero di elementi condivisi, da livelli di significatività differenti. Per tenere traccia di tale parametro, si pesano gli archi mediante la funzione definita in (B.2). In questo modo, ogni arco rappresenta il grado massimo di sovrapposizione tra i cluster c e c', ovvero l'attuale frazione condivisa, indipendentemente dalle dimensioni dei cluster stessi.

$$W(c, c') = \max \left\{ \frac{|c \cap c'|}{|c|}\%, \frac{|c \cap c'|}{|c'|}\% \right\} \tag{B.2}$$

Fissando una soglia $T_w \in [0, 100]$ e rimuovendo gli archi aventi peso $W(c, c') < T_w$, si eliminano dal grafo le relazioni scarsamente significative. In questo modo, inconsistenze forti dovute ad effetti di bordo, potrebbero diventare inconsistenze deboli o consistenze.

Bibliografia

[1] **AV-TEST: malware statistics**
http://www.av-test.org/en/statistics/malware/.

[2] **Malware History**
http://download.bitdefender.com/resources/files/Main/file/Malware_History.pdf.

[3] Mitnick Kevin D., Simon William L. - **L'arte dell'inganno**.

[4] Brett Stone-Gross, Ryan Abman, Richard Kemmerer, Christopher Kruegel, Douglas Steigerwald, and Giovanni Vigna - **The Underground Economy of Fake Antivirus Software** - 10th Workshop on the Economics of Information Security (WEIS), USA, June 2011.

[5] **PandaLabs: Quarterly Report (April-June 2011)**
http://press.pandasecurity.com/wp-content/uploads/2011/07/PandaLabs-Report-Q2-2011.pdf.

[6] F. Maggi, A. Bellini, G. Salvaneschi, S. Zanero - **Finding non-trivial malware naming inconsistencies** - Politecnico di Milano. TR-2011-19.

[7] **Virus Total**
http://www.virustotal.com.

[8] **Beagle**
http://www.commtouch.com/documents/Bagle-Worm_MOTR.pdf.

[9] **Conficker / Downadup**
http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_downadup_codex_ed2.pdf.

[10] **Analysis of Conficker**
http://web17.webbpro.de/index.php?page=analysis-of-conficker.

[11] **Fujacks**
http://www.symantec.com/avcenter/reference/ThePandaOutlaw.pdf.

[12] **ANUBIS: ANalyzing Unknown BInarieS**
http://anubis.iseclab.org.

[13] **DISASM**
http://www.cs.ucsb.edu/~seclab/projects/disasm/index.html.

[14] **UPX: The Ultimate Packer for eXecutables**
http://upx.sourceforge.net/.

[15] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda and Christopher Kruegel - **A View on Current Malware Behaviors** - LEET'09 Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more, pp. 8-8, USENIX Association Berkeley, CA, USA ©2009.

[16] Ulrich Bayer, Andreas Moser, Christopher Kruegel and Engin Kirda - **Dynamic analysis of malicious code** - J Comput Virol, 2006.

[17] A. Moser, C. Kruegel, and E. Kirda - **Exploring multiple execution paths for malware analysis** - in Security and Privacy 2007. SP '07. IEEE Symposium on, pages 231-245, 2007.

[18] **PE (Portable Executable) format**
http://msdn.microsoft.com/en-us/library/ms809762.aspx

[19] **QEMU**
http://wiki.qemu.org/Main_Page

[20] Ulrich Bayer, Christopher Kruegel, and Engin Kirda - **TTAnalyze: A Tool for Analyzing Malware** - 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference, Hamburg, Germany, April 2006.

[21] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel and Engin Kirda - **Scalable, Behavior-Based Malware Clustering** - 16th Annual Network and Distributed System Security Symposium (NDSS 2009), San Diego, February 2009.

[22] Bradford R. B. - **An empirical study of required dimensionality for large scale latent semantic indexing applications** - in Proceeding of the 17th ACM conference on Information and knowledge management, New York, NY, USA ©2008.

[23] **Polymorphic Viruses - Implementation, Detection, and Protection**
http://vx.netlux.org/lib/ayt01.html.

[24] **Understanding and Managing Polymorphic Viruses**
http://www.symantec.com/avcenter/reference/striker.pdf.

[25] C. Kruegel, E. Kirda, D. Mutz, W. Robertson and G. Vigna - **Polymorphic Worm Detection Using Structural Information of Executables** - in Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 207–226, Springer, ISBN 3-540-31778-3, Seattle, Washington, USA, September 2005.

[26] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna - **Static Disassembly of Obfuscated Binaries** - in Proceedings of the 13th USENIX Security Symposium 2004, pp. 255–270, USENIX Association, San Diego, California, August 2004.

[27] **[F]aster [U]niversal [U]npacker**
`http://code.google.com/p/fuu/`.

[28] M.G. Kang, P. Poosankam, and H. Yin - **Renovo: a hidden code extractor for packed executables** - in ACM Workshop on Recurring malcode (WORM), 2007.

[29] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee - **Polyunpack: Automating the hidden-code extraction of unpack-executing malware** - in 22nd Annual Computer Security Applications Conf. (ACSAC), 2006.

[30] Pearson K. - **On lines and planes of closest fit to systems of points in space** - Philosophical magazine 2: 559-572

[31] P. Indyk, R. Motwani - **Approximate nearest neighbor: Towards removing the curse of dimensionality** - STOC'98

[32] Taher H. Haveliwala, Aristides Gionis and Piotr Indyk - **Scalable Techniques for Clustering the Web** - WebDB'2000, Third International Workshop on the Web and Databases, In conjunction with ACM SIGMOD'2000

[33] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman - **Indexing by Latent Semantic Analysis** - Journal of the American Society for Information Science 41 (6): 391–407, 1990.

[34] **Divisi2**
`http://csc.media.mit.edu/divisi`.

[35] **SciPy**
`http://www.scipy.org/`.