ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

L'INDIVIDUAZIONE AUTOMATICA DI PLAGIO FRA CODICI SORGENTE

Relazione finale in

Programmazione

Relatore

Prof.Antonella Carbonaro

Presentata da

Mirella Menghetti

SESSIONE III
ANNO ACCADEMICO 2014/2015

Al mio babbo, come promesso

INDICE

Capitolo 1 – INTRODUZIONE

Le origini del software: codice sorgente e codice oggetto

La necessità di sviluppare un software sorge in relazione all'esigenza dell'utente di risolvere un problema. La realizzazione di un programma richiede diversi passi:

- formulazione di un modello matematico che descriva il problema (algoritmo informale);
- definizione dell'algoritmo in uno pseudo-linguaggio tramite affinamenti successivi fino a quando non vengono creati dei tipi di dato astratti (Abstract Data Type) in cui ciascuna operazione sul tipo è una funzione;
- realizzazione dell'ADT e delle funzioni.

Il risultato di queste operazioni è il testo di un algoritmo scritto in un linguaggio di programmazione, che rappresenta il cosiddetto linguaggio o *codice sorgente*.

Definizione di codice sorgente e codice oggetto

Il *codice sorgente* è la forma in cui è redatto ogni programma, in un linguaggio che spieghi ad altri umani che lo conoscono le caratteristiche e i percorsi logici seguiti dal programmatore nella realizzazione di un certo software; il *codice oggetto*, invece, è una lunga serie di impulsi elettrici negativi e positivi graficamente rappresentati da 1 e da 0, comprensibile al calcolatore.

Affinché un programma scritto in un qualsiasi linguaggio di programmazione sia comprensibile (e quindi eseguibile) da parte di un calcolatore, occorre tradurlo dal linguaggio originario al linguaggio della macchina.

L'operazione di traduzione può essere svolta da programmi *compilatori* oppure da *interpreti*.

I programmi compilatori svolgono essenzialmente tre funzioni:

1. preprocessing: esecuzione delle direttive al preprocessore (inclusione di file, definizione di macro, compilazione condizionale);
2. compilazione: creazione del file object scritto in codice macchina;
3. linking: collegamento dei vari moduli oggetto e delle librerie per generare il file eseguibile.

Gli interpreti invece traducono ogni istruzione immediatamente prima della sua esecuzione senza creare un file oggetto; in questo caso l'esecuzione è più lenta di quella dei compilatori.

Il plagio: un problema nel mondo accademico e non solo

Da quando è nata, Internet ha messo a disposizione enormi quantità di informazioni. Molti tool per il file sharing, come Napster, Gnutella, BitTorrent, hanno reso semplicissima la condivisione di musica, film e documenti. Sebbene i media elettronici abbiano favorito lo sviluppo del plagio con la diffusione dei tool per la condivisione, d'altro canto i moderni tool possono aiutare nell'individuazione del plagio e istruire gli studenti ad evitarlo. È importante far conoscere a tutti la maledizione del plagio affinché sia dato sempre il giusto credito all'autore originale di un lavoro.

Recentemente, all'Università Centrale della Florida è stata introdotta la regola per la quale gli studenti devono sottoporre il loro lavoro alla verifica da parte di un sito web preposto all'individuazione del plagio. Il plagio succhia creatività e, quando diventa una prassi abituale, tende ad atrofizzare la capacità degli studenti di ragionare e pensare in modo originale quando si trovano di fronte a un problema da risolvere. Citare la fonte alla quale si è attinto è indice di integrità accademica poiché significa riconoscere il contributo degli altri in un determinato ambito e rispettare l'intera comunità coinvolta nella ricerca nell'ambito stesso. Un atteggiamento eticamente corretto incoraggia i ricercatori onesti a proseguire nella loro attività mentre un plagio del loro lavoro senza i dovuti riconoscimenti potrebbe avvilirli a tal punto da indurli ad abbandonare il mondo della ricerca.

Un'inchiesta televisiva dell'IEEE (Institute of Electrical and Electronics Engineers) mostra che il numero di casi di plagio aumenta ogni anno. Se nel 2004 i casi rilevati furono 14, nel 2006 erano già 50 all'anno, per arrivare a 100 nel 2008.

La documentazione predisposta dalla ACM (Association for Computing Machinery) sul plagio attribuisce varie ragioni all'aumento del plagio negli ultimi anni. Una delle principali è che gli autori vogliono produrre sempre più pubblicazioni. Inoltre nelle scuole manca un'educazione etica che insegni agli studenti quali sono i comportamenti scorretti. Il plagio spesso si verifica anche quando non si cita la fonte o la si cita male.

Il plagio si può classificare in base a diversi criteri.

Se consideriamo la *forma* in cui si manifesta, possiamo definire i seguenti tipi di plagio:

- *Auto-plagio* – Riutilizzare le proprie frasi copiandole da un proprio precedente lavoro o usare la stessa idea con parole differenti;
- *Falsa paternità* – Includere il nome di qualcuno come autore senza averlo verificato. Questo tipo di plagio include anche il pagamento di qualcuno per riscrivere un altro testo. Molti laureati pensano di dover includere il nome del supervisore nei loro lavori solo perché ha erogato un contributo finanziario, invece l'inclusione di un nome nella lista degli autori deve essere sempre e comunque verificata; per indicare uno specifico riconoscimento ai finanziatori della propria ricerca si può prevedere un'apposita sezione nel proprio testo;
- *Doppia presentazione* – Presentare lo stesso articolo in due conferenze o a due giornali diversi allo stesso tempo;
- *Furto di materiale* – Copiare il materiale di qualcun altro senza chiedere il permesso;
- *Copia non autorizzata di codice sorgente* – Realizzare programmi e lavori di ricerca utilizzando il codice di qualcun altro senza chiedere il permesso né citare la fonte.

Ragionando sul *metodo* in cui il plagio si implementa, possiamo elencare diverse azioni riconosciute come plagio:

- *Copia&Incolla* – Copiare un testo da una fonte e incollarlo in un testo proprio senza permesso né citazione. È il metodo più utilizzato dagli studenti perché è quello più veloce;
- *Parafrasi inappropriata* – Riportare parole da una frase cambiando l'ordine o seguire lo stesso stile di un testo;

- *Omessa citazione* – Quando si utilizza l'informazione scritta da qualcun altro ma non si cita la fonte; è incluso il caso in cui il testo copiato sia rielaborato con altre parole;
- *Falsi riferimenti* – Citare una fonte che non è quella originale o non è quella corretta;
- *Manipolazione di dati* – Manipolare i dati di altri per nascondere il plagio. La manipolazione e la falsificazione dei risultati di ricerca è una forma di reato piuttosto grave nel campo della ricerca, che dimostra la scarsa competenza del ricercatore che copia nel proprio ambito di ricerca;
- *Furto di idee* – Presentare l'idea di qualcun altro come se fosse la propria, senza citare la fonte.

Si copia per differenti ragioni: aumentare il numero delle proprie pubblicazioni; ottenere posizioni di lavoro; conquistare una reputazione migliore nella comunità accademica. Considerando lo *scopo* per il quale si realizza, il plagio può essere:

- *Intenzionale* – Il plagio è un problema molto serio quando è intenzionale. Una ricerca di R. McCuen ha evidenziato che la decisione di copiare è la fase di un processo più ampio che prevede che chi plagia sia di fronte a un problema e non si senta in grado di risolverlo, si guardi intorno per capire se ha una soluzione già pronta a portata di mano, valuti le alternative a disposizione e decida se agire in modo illecito o no;
- *Involontario* – Anche se non è voluto, il plagio può avere sempre conseguenze serie.

Le conseguenze del plagio

Le conseguenze del plagio sono molteplici: mancanza di informazioni autentiche; credito immeritato; ombre sulla reputazione professionale di chi lo compie; limitazioni nello sviluppo della creatività dei giovani studenti. Spesso le conseguenze peggiori si manifestano a distanza di tempo.

Gli studenti onesti devono affrontare un'enorme frustrazione quando gli studenti che hanno copiato il loro lavoro ottengono ottimi voti senza fatica mentre loro invece hanno speso molto tempo ed energie per realizzare quel lavoro autonomamente. Quegli studenti che hanno ottenuto voti più alti

copiando da buone soluzioni spesso sfuggono alla punizione dal momento che non vengono scoperti.

Un'altra preoccupazione nel mondo accademico è che il plagio possa degradare la reputazione delle università. Gli studenti internazionali che arrivano in un paese hanno grandi aspettative in termini di standard accademici e sono un'importante risorsa per quel paese, sia per le tasse di iscrizione che pagano che per il contributo che apportano alla ricerca universitaria.

Il plagio è dannoso per gli studenti anche in termini di sviluppo delle capacità di comunicazione. Il linguaggio è una barriera per molti studenti, che spesso superano utilizzando materiale plagiato da altri testi.

Un altro problema creato dal plagio è il fatto che distrugge la creatività nelle menti dei giovani studenti. Quando uno studente è abituato a copiare materiale dal lavoro di qualcun altro o a scaricarlo da Internet, non si preoccupa più di ragionare sulle materie che studia. Col passare del tempo, questa abitudine di copiare si rafforza fino a far scomparire ogni abilità creativa dalla mente.

Nella cultura del copia & incolla gli individui non sviluppano l'energia che serve per creare innovazione e quindi non c'è creatività per nessuno.

Le conseguenze del plagio dipendono dalla gravità del gesto e dai regolamenti dell'istituzione. Nel caso in cui il plagio coinvolga lavori soggetti a *copyright* ci sono regole specifiche stabilite dalla legge. Anche se è considerato un reato minore, il plagio ha un costo elevato in termini di creatività, pensiero razionale, valori come l'onestà e l'integrità accademica. Può distruggere carriere, creare danni finanziari e imbarazzo nella vita sociale, accademico e professionale.

Strategie contro la maledizione del plagio

Come è chiaro, il plagio è un problema di reato accademico e disonestà, quindi prima di tutto è bene cercare di evitarlo. Esistono diverse strategie per fare ciò.

Le strategie suggerite in letteratura per evitare il plagio sono classificate in livelli individuali e di organizzazione. Un individuo singolo può adottare strategie per favorire l'abitudine ad evitare il plagio e può concentrarsi sul proprio innovativo e originale modo di pensare. Allo stesso modo, le

istituzioni possono adottare politiche per far fronte al plagio e quindi preservare la propria reputazione.

Sviluppare nelle persone la consapevolezza di quali sono le conseguenze del plagio è ancora più importante che insegnare agli studenti quali differenti metodologie vengono utilizzate per individuare il plagio. Alcuni studenti considerano un gioco evitare di essere scoperti quando copiano e l'unico modo per evitare questo atteggiamento è educarli dal punto di vista etico.

Strategie personali

In molti casi il plagio accade inavvertitamente. Le ragioni sono diverse:

- scarsa attenzione nella citazione, anche se gli studenti dovrebbero tenersi aggiornati sulle regole da seguire per comporre testi e includere citazioni (alcuni errori comuni sono: copiare il testo di qualcun altro esattamente e non includerlo tra virgolette; raccogliere le informazioni dal lavoro di un altro senza darne il giusto riconoscimento; utilizzare il materiale scaricato sul computer di qualcun altro senza ricordare la fonte del materiale);
- scarsa capacità di ricerca, non solo da parte degli studenti ma anche di ricercatori professionisti, più inclini a usare dati ipotetici per supportare i loro argomenti;
- scuse illegittime (si dovrebbe tenere a mente che "non saprei scriverlo meglio di così" non è una valida ragione per copiare);
- auto plagio (sarebbe utile avere un elenco aggiornato dei propri lavori, per evitare di riproporre per errore un lavoro già presentato);
- falso contributo al lavoro (la citazione è un atto dovuto solo se la persona che si cita ha realmente contribuito alla realizzazione del lavoro);
- aspetti culturali (in molti paesi più o meno sviluppati gli studenti e i docenti spesso ignorano le loro abitudini in relazione al plagio, mentre dovrebbero acquisire più consapevolezza delle proprie azioni e delle loro conseguenze);
- ignoranza rispetto all'attuale sviluppo di una determinata materia (un autore, prima di pubblicare un lavoro come proprio e innovativo, dovrebbe consultare diversi motori di ricerca e materiale

bibliografico per accertarsi che il lavoro che vuole presentare non esista già);

- riferimenti indiretti (quando si usa materiale che fa a sua volta riferimento ad un'altra fonte, occorre citare anche la fonte originale, in modo da dare il giusto credito al proprietario legittimo di quel testo/idea/lavoro che si condivide);
- ignoranza delle conseguenze, pensando che comunque il plagio non sia un reato grave;
- timore di non riuscire a rispettare una scadenza (la scadenza della laurea per un laureando; l'imminenza della data di una conferenza per il ricercatore che deve presentare un articolo in quell'occasione);
- difficoltà nello svolgere il ruolo di docente (spesso episodi di plagio si verificano sotto la supervisione di un docente e anche se non ne ha direttamente responsabilità, dovrebbe educare i suoi studenti ad evitare il plagio, anche imponendo loro tempi di consegna ragionevoli che non mettano troppa pressione);
- errori accidentali e dell'ultimo minuto (una scarsa pianificazione del lavoro può favorire incidenti di plagio involontario, perciò è bene darsi una buona organizzazione di lavoro e rileggere più volte il lavoro realizzato prima di consegnarlo);
- sono necessarie maggiori competenze sulle citazioni, seguendo la regola che se si è in dubbio se citare o meno è sempre meglio citare;
- lavori di gruppo (in un lavoro di gruppo deve essere chiaramente esplicitato il contributo di ognuno per evitare incidenti spiacevoli);
- prendere appunti in modo corretto è fondamentale per evitare errori o omissioni;
- conoscere a fondo le informazioni, poiché spesso il plagio si verifica per una mancata comprensione del testo originale;
- ri-verifica del materiale che abbiamo realizzato confrontandolo con l'originale per essere certi che non sia accidentalmente uguale;
- lavorare al computer richiede ancora più attenzione alle fonti utilizzate;
- uso di risorse scolastiche ma cercando sempre le fonti originali del materiale che si trova;

- imparare gli stili per i riferimenti (a seconda della conferenza o rivista o università per la quale si sta realizzando il lavoro, poiché ognuno potrebbe avere regolamenti differenti)

Strategie nelle istituzioni

Le istituzioni possono adottare diverse misure per evitare il plagio, che avrebbe un impatto negativo sulla loro reputazione:

- maggiore consapevolezza all'interno della comunità accademica (si possono organizzare seminari e dibattiti con esperti in questo campo per sviluppare consapevolezza negli studenti, nel personale universitario e nei docenti);
- educazione morale (gli studenti dovrebbero essere educati a comportamenti etici, gli stessi adottati dai loro docenti, poiché rispettare l'etica nella propria attività significa avere più rispetto anche per le persone con le quali si interagisce);
- aspettative chiare (gli insegnanti dovrebbero indicare chiaramente cosa si aspettano dagli studenti per consentire a tutti di svolgere il compito che è stato loro assegnato senza che abbiano necessità di prendere scorciatoie);
- mantenersi aggiornati con la tecnologia (i docenti devono sapere quali strumenti la tecnologia mette a disposizione degli studenti disonesti);
- azioni disciplinari (occorre punire in maniera esemplare i comportamenti scorretti, anche per preservare la qualità del lavoro di ricerca che gli studenti svolgeranno in futuro);
- tool per l'individuazione del plagio (esistono moltissimi strumenti per l'individuazione delle varie forme di plagio, anche di quello fra codici sorgente);
- leggi nazionali, anche se sarebbe auspicabile avere una piattaforma che regolamenti l'attività di scrittori e ricercatori a livello internazionale).

Il plagio è molto diffuso negli ambienti universitari, perciò è importante educare studenti e docenti offrendo loro guide e tutorial che spieghino quali sono i tipi di plagio e come evitarli.

Molti dei tool disponibili per l'individuazione automatica del plagio agiscono analizzando i lavori quando sono già stati consegnati ma il primo e più importante strumento per combattere il plagio deve essere l'educazione degli studenti.

Molte università negli Stati Uniti hanno definito chiaramente quali comportamenti sono considerati scorretti e prevedono richiami verbali e scritti, penalizzazione nei voti o assegnazione di esercizi extra per quegli studenti che vengono scoperti a copiare. Ad esempio, nel caso di plagi reiterati, la Stanford University prevede un periodo di sospensione oltre a 40 ore di servizi da prestare gratuitamente alla comunità. Anche Yale, Berkeley e MIT (Massachusetts Institute of Technology) hanno definito politiche precise e procedure severe per punire la disonestà in ambito universitario.

Anche nelle università d'Europa è in aumento l'applicazione di misure contro il plagio; molte istituzioni mettono a disposizione di studenti e ricercatori guide online e tutorial, al fine di aiutarli a comprendere l'importanza dell'integrità accademica.

Anche l'Università degli Studi di Bologna si è dotata di un Codice etico e di comportamento che definisce l'etica e la responsabilità dei comportamenti «valori fondamentali per il perseguimento delle finalità istituzionali, per favorire il merito e l'eccellenza, lo scambio con la comunità scientifica nazionale ed internazionale, la creazione di un ambiente professionale aperto al dialogo e alle corrette relazioni interpersonali, la tutela dei valori della persona in tutti i suoi aspetti».

La normativa e i tool per l'individuazione automatica

La tutela del software

La natura tecnica dei programmi per elaboratore porta con sé un'importante conseguenza: l'assoluta impossibilità di valutare la somiglianza di due software alla stregua dei criteri applicabili a tutte le altre opere dell'ingegno. Il software, infatti, pur essendo un'opera dell'ingegno umano *scritta* in un determinato linguaggio – poco importa se simbolico o convenzionale – non è destinato ad esser letto, eseguito o fruito dall'uomo ma, esclusivamente, dalla macchina e, anzi, al contrario di quanto avviene per ogni altra opera letteraria, musicale o cinematografica, il suo autore ha tutto l'interesse a che

esso rimanga segreto alla collettività. Non avrebbe pertanto alcun senso cercare di valutare l'originalità di un programma per elaboratore alla stregua di parametri estetici o in base al godimento intellettuale che esso riesce o meno a suscitare nell'utente ma si dovrà piuttosto apprezzare il livello di utilità che esso produce una volta eseguito sulla macchina.

Con il passare degli anni, il progresso tecnologico ha consentito di produrre hardware a costi sempre inferiori e il software è divenuto il motore di un settore industriale che apporta contributi sempre più significativi all'economia mondiale.

Tenuto conto del consistente valore commerciale del software, è sorta l'esigenza di individuare adeguati strumenti di tutela. Nonostante geneticamente e ontologicamente il software sia probabilmente più vicino alle invenzioni industriali che non alle opere dell'ingegno, negli anni '60 il mondo giuridico ritenne di equiparare il software alle opere dell'ingegno e, quindi, di proteggerlo dapprima negli Stati Uniti e, quindi, nel resto del mondo, attraverso il *copyright* nei paesi di Common Law e con il diritto d'autore nel vecchio continente.

In Italia, in fatto di diritto d'autore, il testo legislativo di riferimento resta tutt'oggi la Legge 633/1941 (la cosiddetta Legge sul Diritto d'Autore). Essa ha subito nel corso degli anni cospicui interventi di riforma e di integrazione, soprattutto dietro la spinta della normativa europea, come la Direttiva 91/250/CEE, relativa alla tutela giuridica dei programmi per elaboratore, attuata nel nostro paese con il Decreto Lgs. n. 518/1992.

L'individuazione automatica del plagio

La ricerca del plagio fra documenti può consistere in un confronto manuale dei testi, operazione molto lenta e imprecisa, oppure può affidarsi all'utilizzo di numerosi tool che negli anni sono stati sviluppati per consentire l'individuazione automatica, ottenendo in tempi più brevi risultati molto più soddisfacenti.

Numerosi tool sono dedicati al rilevamento di plagio nei testi: **Turnitin**, uno dei tool più famosi, disponibile come servizio web al quale gli utenti registrati possono sottoporre una serie di documenti da confrontare; **SafeAssignment**: disponibile anch'esso come servizio web che confronta i documenti caricati dall'utente con milioni di altri documenti presenti in rete;

Docoloc: servizio web che sfrutta le potenzialità delle API di Google per effettuare i confronti.

Negli ultimi anni **Turnitin** è stato adottato come software anti plagio anche in diverse università italiane, tra cui l'Università di Bologna, presso la quale viene utilizzato per verificare l'effettiva originalità delle tesi di laurea. Nella figura 1 è rappresentata l'interfaccia a disposizione del docente, in particolare l'area di lavoro *Quick submit* che permette di analizzare un singolo elaborato, così come compare sul manuale docenti disponibile nell'apposita sezione del portale www.unibo.it.

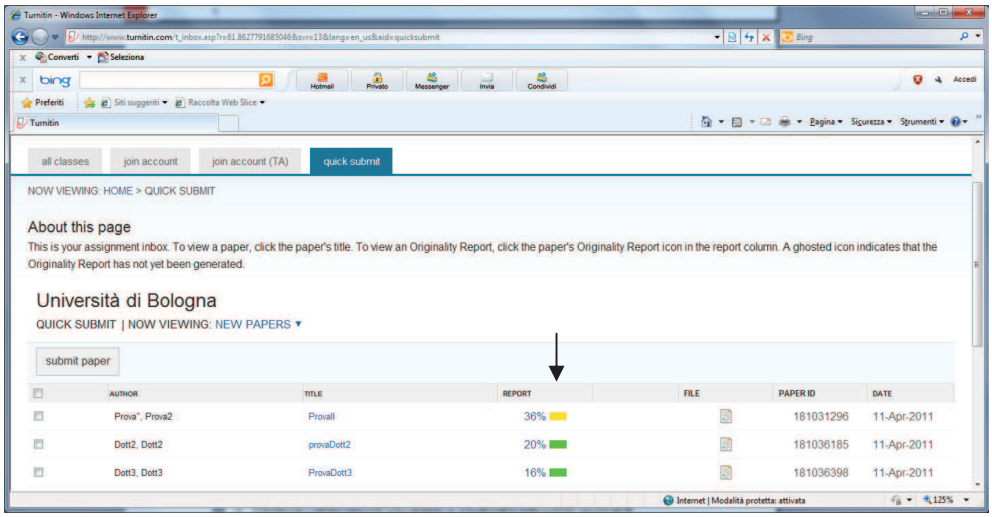

Figura 1 – Interfaccia del tool Turnitin. Al termine dell'analisi la colonna *REPORT* restituisce la percentuale di similarità, evidenziando i valori con colori diversi: blu = 0%; verde < 24%; giallo 25%-49%; arancione 50%-74%; rosso >75 %

Quando si usano questi web service per valutare le esercitazioni svolte dagli studenti è corretto informarli che il procedimento assegnerà ad ogni documento una sorta di impronta digitale (detta *fingerprint*) che verrà a sua volta utilizzata per ulteriori verifiche future; non sono rari casi di studenti che, non correttamente informati dalla loro istituzione, hanno minacciato di citarla in giudizio per violazione della proprietà intellettuale.

Il problema del plagio non è confinato al mondo accademico ma interessa anche il mondo commerciale. Anche in ambito commerciale quindi sono stati creati diversi tool, come **iThenticate, CopyGuard** o **Copyscape**, applicazioni web che sono in grado di analizzare e confrontare un documento con milioni di altri presenti in rete. Questi tool sono stati realizzati con lo scopo di proteggere le aziende sia dalla possibilità di plagio che dalla perdita di dati.

I linguaggi di programmazione impongono dei vincoli sintattici che determinano alti gradi di similarità anche fra codici sorgente scritti autonomamente, perciò analizzare il software è più complesso che analizzare dei testi scritti.

Per tale ragione, negli ultimi anni sono stati sviluppati numerosi tool per la rilevazione automatica di plagio fra codici sorgente, come ***MOSS*** o ***JPlag***, anch'essi disponibili come servizi web che ricevono in input una serie di documenti, li confrontano e restituiscono i risultati dei confronti sotto forma di pagine HTML.

Obiettivo di questa tesi è raccogliere e ordinare le informazioni disponibili sul plagio fra codici sorgente, con particolare attenzione alla classificazione dei tipi di plagio fra codici e all'analisi dei principali tool esistenti per l'individuazione automatica.

Dall'ampia letteratura disponibile sull'argomento, emerge che la rivoluzione informatica ha suscitato nel mondo giuridico un profondo dibattito in merito alla definizione del software e agli strumenti di tutela ad esso connessi, perciò ho deciso di dedicare un capitolo anche alla riflessione sul contesto giuridico e normativo, in Italia e nel mondo.

Poiché non esiste una bacchetta magica per combattere il plagio, è realistico pensare che utilizzare diverse tecniche fra loro complementari possa dare risultati migliori; alcuni approcci innovativi in tal senso, derivanti principalmente da ricerche effettuate nel mondo accademico, sono descritti nel capitolo conclusivo della tesi.

Un altro aspetto del problema che mi ha colpito è la questione etica connessa al plagio. Si tratta di una problematica che già avevo vissuto da studente e che ha trovato conferma nelle numerose pubblicazioni che ho consultato: molti studiosi universitari hanno realizzato sondaggi all'interno della comunità accademica per testare il grado di sensibilità al problema del plagio e proposto diverse soluzioni volte non solo a combattere il plagio ma anche a prevenirlo, puntando sullo sviluppo di una maggiore consapevolezza del problema negli accademici (sia studenti che docenti). Anche in ambito commerciale non mancano tentativi di sfuggire alla piaga del plagio, considerato a tutti gli effetti una forma di pirateria informatica.

Alcuni di questi studi, i cui risultati offrono interessanti spunti di riflessione per il futuro, sono riportati nel capitolo dedicato alle conclusioni.

Capitolo 2 – IL PLAGIO FRA CODICI SORGENTE: DEFINIZIONE, TIPI, RILEVAMENTO

Il plagio fra codici sorgente

Per distribuire un programma interpretato si deve necessariamente distribuire il codice sorgente, rendendo possibili operazioni di plagio.
Il plagio è un problema anche nell'insegnamento della programmazione. Il codice sorgente può essere ottenuto in vari modi, inclusi Internet, le banche dati di codici, i libri di testo. Esistono risorse online alle quali gli studenti possono rivolgersi per assumere esperti programmatori che realizzino per loro i progetti assegnati dai docenti. Queste opportunità rendono il plagio molto semplice, tanto che i casi di plagio negli ultimi anni sono in costante aumento.
I contenuti digitali sono molto facili da copiare e manipolare poiché è tale la quantità di dati messa a disposizione dai motori di ricerca su web che il fenomeno del plagio di codice sorgente è molto difficile da tenere sotto controllo.
Quando parliamo di "plagio fra codici sorgente" intendiamo riferirci al riutilizzo non autorizzato della struttura e della sintassi di un programma già esistente; in realtà, considerata la natura così variegata dei linguaggi di programmazione e la semplicità con la quale un codice può essere modificato, non solo il plagio è un fenomeno sempre più diffuso, ma non è nemmeno semplice darne una definizione condivisa universalmente.

Verso una definizione: un'indagine nel mondo accademico

G. Cosma e M. Joy hanno realizzato un'interessante indagine per provare a trovare una definizione condivisa di plagio del codice sorgente nel mondo

universitario. La ricerca, realizzata sotto forma di questionario online, ha coinvolto circa 110 istituti segnalati dall'HEA-ICS, Centro che nel Regno Unito raggruppa gli istituti che si occupano di insegnamento universitario di scienze dell'informazione e informatica.

I questionari sono stati compilati anonimamente ma era inclusa una sezione nella quale facoltativamente era possibile indicare qualche informazione personale in più; dei 59 che hanno risposto al questionario, 43 hanno indicato presso quale istituzione lavoravano. Il questionario comprendeva per lo più domande a risposta chiusa; la maggioranza dei quesiti presentava diversi scenari, per ognuno dei quali veniva descritto il modo in cui gli studenti avevano ottenuto e utilizzato del materiale e si chiedeva di indicare se era presente in ogni scenario un reato accademico e, se sì, di quale tipo.

Cosa si copia

Il plagio nelle esercitazioni di programmazione può andare oltre alla copia del codice sorgente; può includere commenti, dati di input del programma, interfacce. I commenti all'interno del codice possono essere copiati e contribuire a identificare i casi di plagio del codice. I dati di input e l'interfaccia utente possono essere soggetti a plagio se sono parte delle richieste specificate dall'assegnazione. La maggioranza degli intervistati (40 su 59) afferma che i dati di input possono essere plagiati ma da soli non sono sufficienti per identificare il plagio. Secondo alcuni l'aver copiato i dati di input è un problema se gli studenti sono valutati anche per le strategie di testing del loro programma; in questo caso per stabilire la presenza o meno di plagio occorre valutare anche i dati di input usati per testare il programma, la documentazione tecnica e i manuali utente. Anche l'interfaccia deve essere analizzata alla ricerca di eventuali segnali di plagio solo se l'assegnazione richiede agli studenti di sviluppare da soli un'interfaccia per il loro programma.

Come si copia: adattare, convertire e riutilizzare il codice sorgente

Gli scenari proposti agli intervistati prevedevano casi di copia, adattamento, conversione di codice sorgente da un linguaggio di programmazione a un altro nonché utilizzo di software per generare codice automaticamente. Il generatore di codice è un'applicazione che, ricevuti in input dei metadati (ad esempio lo schema di un database), crea codice sorgente compatibile con il

modello di progettazione. Un esempio di generatore di codice shareware è JSPMarker che, sulla base di un database ricevuto in input, è in grado di generare velocemente e facilmente un codice sorgente completo e il set di JavaServer pages necessarie per rappresentare la connettività del database.

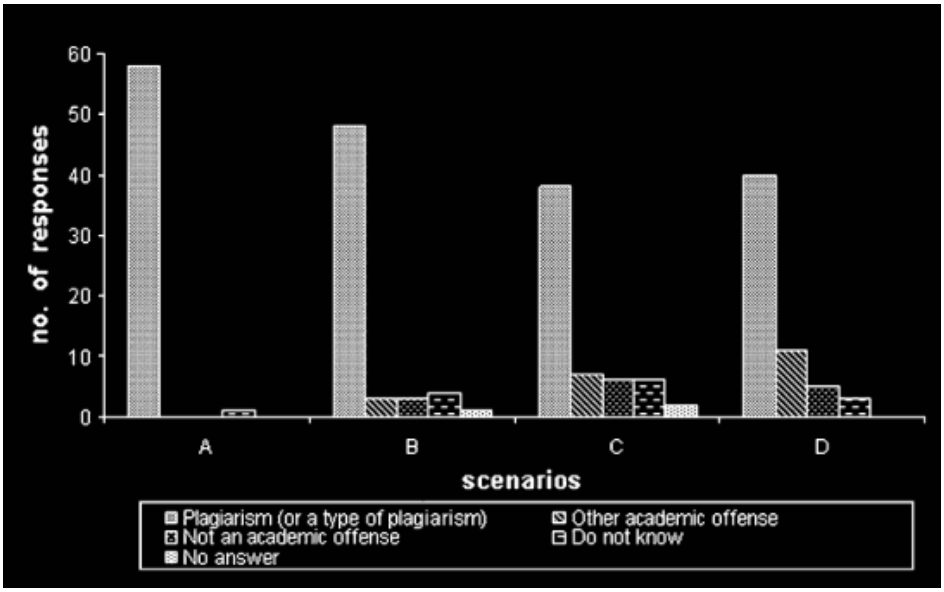

Figura 2 – Scenari e risposte. *A*: lo studente riproduce/copia il codice sorgente di qualcun altro senza effettuare alcuna alterazione e lo propone come proprio senza citare alcun riconoscimento all'autore; *B*: lo studente riproduce/copia il codice sorgente di qualcun altro, adatta il codice al proprio lavoro e lo propone come proprio senza citare alcun riconoscimento all'autore; *C*: lo studente converte tutto o parte del codice sorgente di qualcun altro in un differente linguaggio di programmazione e lo propone come proprio senza citare alcun riconoscimento all'autore; *D*: lo studente usa un generatore di codice (software che può essere usato per creare automaticamente codice passando attraverso ad una procedura guidata), rimuove i commenti inseriti automaticamente dal software all'interno del codice, e propone il codice come proprio senza citare alcun riconoscimento all'autore

Gli scenari e le relative risposte sono rappresentati in figura 2. Le risposte relative agli scenari *A* e *B* concordano ampiamente che *riprodurre/copiare il codice sorgente di qualcun altro modificandolo oppure no e presentandolo come proprio senza citare alcun riconoscimento all'autore* costituisce plagio di codice sorgente. Un accademico ha commentato:

«...negli ambienti *object-oriented* il riutilizzo è incoraggiato, ovviamente riutilizzare alcuni elementi non è plagio. Ritengo debbano essere chiari i confini e i limiti in ogni circostanza e cerco di comunicarlo chiaramente anche ai miei studenti ma ovviamente potrebbero esserci dei problemi».

Riguardo allo scenario *B*, alcuni accademici hanno commentato che per stabilire se l'adattamento del codice rappresenta un plagio occorre considerare il grado di adattamento, per esempio quanto del codice è copia del lavoro di qualcun altro e quanto del codice è stato adattato senza citare l'autore. Per esempio un programma può non essere considerato plagiato se

è stato costruito partendo da un codice esistente e adattandolo talmente che non è rimasto nulla del codice originale per il quale sarebbe dovuto un riconoscimento all'autore.

Un altro accademico ha ribadito:

«...copiare codice da un sito web che supporta gli studenti in uno specifico problema è potenzialmente una buona pratica. Un codice copiato al 100% è una questione differente. Occorre considerare anche il contesto in cui un codice viene copiato. Se l'unico risultato finale dovessero essere il codice e la documentazione la violazione è palese. In questo senso suppongo che il problema sia quanto delle esercitazioni sia attualmente copia di altri lavori (senza riconoscimenti)».

In relazione al *convertire tutto o parte del codice sorgente di qualcun altro in un differente linguaggio di programmazione presentandolo come proprio senza citare alcun riconoscimento all'autore* (scenario *C*), diversi degli accademici intervistati ribadiscono che se il codice è convertito automaticamente senza sforzo da parte dello studente, allora questa procedura costituisce plagio. Se lo studente invece prende spunto da un codice scritto in un altro linguaggio di programmazione e crea un codice sorgente interamente da zero allora questa procedura probabilmente non è plagio.

Un accademico ha commentato: «in ogni caso ci deve essere qualche presunto vantaggio per lo studente nel fare ciò (perché lo farebbe altrimenti?) e una rottura con il sistema di valutazione. Anche se il vantaggio fosse minimo, un bravo studente saprebbe quasi certamente riconoscere la situazione e usarla per discutere delle differenze».

Agli intervistati è stato chiesto se considerassero plagio il caso in cui *uno studente usasse un software generatore di codice, rimuovesse i commenti inseriti automaticamente dal software all'interno del codice e proponesse il codice come proprio senza citare alcun riconoscimento all'autore*. La maggioranza degli intervistati considera l'uso di codice generato automaticamente senza riconoscimenti come plagio, a meno che il permesso di usare questo tipo di software non sia previsto dalle specifiche di un'assegnazione.

L'indagine suggerisce che agli studenti dovrebbe essere richiesto di citare la fonte di ogni materiale che utilizzano se non è un loro lavoro originale,

anche quando il riutilizzo di codice sorgente è consentito. Tutto il materiale dovrebbe essere riconosciuto indipendentemente dai permessi di licenza (per esempio: *open source*, *free-use*, *fair-use*).

L'auto-plagio

Nelle esercitazioni non di programmazione, l'auto-plagio si verifica quando uno studente riutilizza parti di esercitazioni precedentemente presentate e le include in un'altra assegnazione senza provvedere a un'adeguata dichiarazione di questo fatto. Nella programmazione, che prevede nei suoi moduli il riutilizzo di codice sorgente, l'auto-plagio può non essere considerato una violazione.

Di fronte allo scenario in cui agli studenti *non è consentito di riproporre materiale già realizzato da loro e presentato per un'assegnazione precedente; per un assegnazione valutata, uno studente ha copiato parti di codice sorgente che aveva realizzato per un'altra assegnazione senza dichiararlo*, le risposte degli intervistati sono state controverse.

La maggioranza (48 su 59) ha considerato questa situazione come una violazione, definendola come «auto-plagio», «violazione delle regole delle esercitazioni, se il riutilizzo non era consentito», «frode se il riutilizzo non è riconosciuto esplicitamente». Alcuni accademici hanno ritenuto che questo caso non fosse un plagio, poiché è corretto riutilizzare il codice sorgente a disposizione dove è possibile ed è molto difficile imporre a uno studente di non riutilizzare del materiale che ha già.

Le esercitazioni e la collaborazione tra gli studenti

Quale dovrebbe essere il peso minimo di un'assegnazione all'interno di un modulo affinché si proceda con l'investigazione per individuare il plagio? A questa domanda la maggioranza degli intervistati ha risposto suggerendo una politica di tolleranza zero, anche se l'offesa viene considerata tanto maggiore quanto maggiore è il contributo dell'assegnazione. Il plagio si può verificare indipendentemente dal fatto che un'assegnazione sia valutata o meno; molto intervistati hanno commentato che quando si scopre il plagio in un lavoro non valutato, gli studenti dovrebbero essere avvisati e informati o allertati circa le implicazioni del loro gesto sull'attribuzione del voto.

L'inchiesta ha sollevato anche la questione dell'appropriatezza / non appropriatezza della collaborazione tra gli studenti. C'è massima concordia

nel definire pedagogicamente preziosa la condivisione delle idee e del lavoro fra gli studenti, anche in previsione di un futuro lavoro in team. Dall'inchiesta emerge che l'occorrenza del plagio varia, sia in tipo che in frequenza, a seconda dell'attività intrapresa; ad esempio il plagio tende ad essere più frequente durante i test che richiedono l'accesso al codice piuttosto che durante quelli condotti a livello di interfaccia software.

Definizione di plagio fra codici

Il plagio di codice sorgente nel mondo accademico si verifica quando uno studente riutilizza un codice sorgente realizzato da qualcun altro e, intenzionalmente o accidentalmente, dimentica di citare adeguatamente l'autore, presentando il lavoro come proprio. Questo comporta ottenere codice sorgente, con o senza permesso dell'autore originale, e riutilizzarlo come parte di un'altra valutazione. Un'azione di plagio include quindi tre aspetti: *acquisizion*e (con o senza il permesso dell'autore originale), *riutilizzo*, *riconoscimento inadeguato*.

Il termine *acquisizione*, può riferirsi a diverse azioni:

- pagare qualcuno perché crei una parte o tutto il codice;
- rubare il codice sorgente di un altro studente;
- collaborare con uno o più studenti per creare un'assegnazione di programmazione anche nel caso in cui era stato chiesto agli studenti di lavorare individualmente;
- scambiare parti di codice sorgente tra gruppi differenti creando la stessa assegnazione

Il *riutilizzo* può verificarsi nel caso di:

- riproduzione/copia di codice senza modifiche;
- riproduzione/copia di codice con alcuni leggeri adattamenti;
- conversione di una parte di codice o di tutto in un differente linguaggio di programmazione;
- generazione codice con un generatore automatico

Si parla di *riconoscimento inadeguato* nel caso in cui lo studente:

- dimentichi di citare la fonte e l'autore;
- fornisca referenze errate;
- fornisca referenze false;

- modifichi l'output del programma affinché sembri funzionare anche se non è così.

Le differenze tra le politiche universitarie, i requisiti di ogni singola assegnazione e le preferenze accademiche personali possono suscitare varie riflessioni tra docenti e studenti sulla definizione di plagio. Gli ambienti *object-oriented* incoraggiano il riutilizzo di codice; quando il riutilizzo è permesso, gli studenti dovrebbero adeguatamente dichiarare le parti di codice sorgente scritte da altri autori. I docenti devono informare chiaramente i loro studenti sulle loro preferenze, specialmente su riconoscimento e riutilizzo del codice sorgente.

Classificazione dei tipi di plagio

Il plagio del codice sorgente si differenzia dagli altri tipi di plagio; facile da eseguire, è difficile da individuare anche perché spesso è il risultato della combinazione di tante porzioni di codice plagiato perciò riconoscere le fonti può essere estremamente complesso. Le modifiche possibili sul codice sorgente sono di tipo *lessicale* oppure *strutturale*. Una modifica *lessicale* è un semplice cambiamento nel testo effettuato con un qualunque text editor (aggiunta/rimozione di commenti, modifica del nome di una variabile, ecc.) ed è molto facile da rilevare. Per effettuare modifiche *strutturali* invece è necessaria una certa conoscenza della programmazione; sono modifiche strutturali quelle che interessano le iterazioni, i cicli, le istruzioni interne alle funzioni, le dichiarazioni.

Potremmo riassumere come segue i diversi modelli di plagio descritti in letteratura:

1. copiare parola per parola, cambiando solo i commenti;
2. modificare gli spazi e la formattazione, rinominare gli identificatori;
3. riordinare i blocchi di codice e le dichiarazioni al loro interno;
4. cambiare l'ordine di operandi e operatori nelle espressioni;
5. cambiare i tipi di dato e aggiungere dichiarazioni o variabili ridondanti;
6. rimpiazzare strutture di controllo con altre equivalenti

Faidhi e Robinson (1987) hanno definito sei differenti livelli di modifica, che vanno dalla più semplice alla più complicata: se il programma originale

è il livello 0, il livello 1 consiste nella sola modifica di commenti e indentazioni; parliamo di livello 2 se la modifica riguarda i nomi degli identificativi; livello 3 per modifiche nella dichiarazione di costanti, variabili e procedure; livello 4 quando vengono modificati i moduli del programma; livello 5 se il plagiatore interviene sulle dichiarazioni; livello 6 se vengono modificate le espressioni logiche.

Copia&Incolla

Consideriamo un codice sorgente scritto in Java, costituito da due classi, `Example1` e `IntArray`.

```
class Example1{
public static void main(String[] args){
IntArray test = new IntArray(12,8);
System.out.println("Randomly generated integer array");
System.out.println(test);
test.sort();
System.out.println("Array after it is sorted");
System.out.println(test);
}// end main
}// endclass Example1


class IntArray {
int size, max;
int numswaps, numcomparisons;
int [] values;

IntArray (int size, int maxvalue){
reset(size, maxvalue);}
// end IntArray constructor

public void reset(int sizeIn, int maxIn){
size = sizeIn; max = maxIn;
values = new int [size]; randomize();
}// end reset

public void randomize(){
for (int i =0; i<size; i++)
   values[i] =(int)(Math.random()*max)+1;
numswaps=0; numcomparisons=0;
```

```
}// end randomize

public boolean compare(int i, int j){
if (i<size && i>=0 && j<size && j>=0){
   numcomparisons++;
   return values[i]> values[j];
}else return false;
}// end compare

public void swap(int i, int j){
if (i<size && i>=0 && j<size && j>=0){
   numswaps++;
   int temp = values[i];
   values[i] = values[j];
   values[j] = temp;}
}// end swap

public void sort(){
      for (int i =0; i<size; i++)
            for (int j = i+1; j<size; j++)
                  if (compare(i,j)) swap(i,j);
}// end sort

public String toString(){
String str ="size: "+size+
            "max: "+max+" values: ";
for (int i=0; i<size;i++) str = str+" "+values[i];
str = str+"\n number of swaps = "+numswaps+
            "number of comparisons = "+numcomparisons;
return str;
}// end toString
}// endclass IntArray
```

Immaginiamo ora che uno studente debba realizzare un codice che esegua l'ordinamento degli interi in ordine decrescente; immaginiamo poi che, cercando sul web, lo studente rintracci il codice `IntArray`; ecco come potrebbe rielaborarlo per realizzare il proprio:

```
class MyArray {
int size;int max;int [] values;
MyArray (int size, int maxvalue){
reset(size, maxvalue);}
```

```
public void reset(int sizeIn, int maxIn){
size = sizeIn;
max = maxIn; values = new int [size];randomize();}

public void randomize(){
for (int i =0; i<size; i++)
      values[i] =(int)(Math.random()*max)+1;}

public boolean compare(int i, int j){
if (i<size && i>=0 && j<size && j>=0)
{return values[i]< values[j];}else return false;}

public void swap(int i, int j){
if (i<size && i>=0 && j<size && j>=0)
{int temp = values[i];
values[i] = values[j];values[j] = temp;}
}

public void sort(){
for (int i =0; i<size; i++)
for (int j = i+1; j<size; j++)
      if (compare(i,j)) swap(i,j);}

public static void main(String[] args){
MyArray test = new MyArray(10,20);
System.out.println("Randomly generated array");
System.out.println("Values: ");
for (int i=0; i<test.size;i++)
      System.out.print(" "+test.values[i]);
test.sort();
System.out.println("\nArray sorted (descending order)");
System.out.println("Values: ");
for (int i=0; i<test.size;i++)
      System.out.print( " "+test.values[i]);
}}
```

Appare chiaro a chiunque abbia un po' di familiarità con la programmazione che il codice presentato dallo studente è copiato. Questo tipo di *copia&incolla* è la forma più comune di riutilizzo di codice che si trova nei lavori presentati dagli studenti.

Collusione

Se lo studente non riesce ad eseguire l'esercitazione che gli è stata affidata, è normale che chieda aiuto. Può accadere che lo studente riceva aiuto da parte di qualcuno che è d'accordo a fornirgliene ma, anche se non è un caso di furto del lavoro di altri, anche la *collusione* è una forma di plagio poiché l'esercitazione assegnata dal docente prevedeva che il lavoro venisse svolto individualmente e la collaborazione di una terza persona non viene dichiarata in alcun modo.

Modifica di codice non riconosciuta

Modificare il lavoro di altri spostando porzioni di codice e presentare il risultato come un lavoro proprio equivale a riutilizzare il codice originale senza modificarlo. Parliamo di *furto delle idee* di qualcun altro e, se non si riconosce esplicitamente la fonte alla quale si è attinto, si incorre nel reato di plagio.

Traduzione non riconosciuta

Si tratta di plagio anche quando uno studente partendo da un codice Java lo *rielabora* per generarne un altro in C++. Tale prassi può essere accettata solo se lo studente cita esplicitamente l'autore del codice originale, evitando di ingannare deliberatamente il docente.

Generazione di codice non riconosciuta

Esistono tool per la *generazione automatica di codice*, molto utili ad esempio per implementare codice in un linguaggio partendo da codice scritto in un altro. Niente di male nell'utilizzo di questo software, purché il ruolo del tool sia esplicitamente riconosciuto.

Riutilizzo del codice senza testarlo

Alcune regole da tenere a mente: se non sai come testare un codice, non riutilizzarlo; se non sai che cosa puoi testare con quel codice, non usarlo; se sai come e su che cosa testare un codice, riutilizzalo solo dopo averlo testato completamente e con successo.

Clonazione

Rientra nel mondo del plagio anche la *clonazione*, una pratica comune nello sviluppo di software e la cui rilevazione è un'operazione essenziale per la

manutenzione e l'evoluzione del software stesso. In generale il codice clonato può essere classificato in 4 modi diversi:

- Tipo 1: una copia esatta dell'originale senza modifiche;
- Tipo 2: copia sintatticamente identica ma con variabili, tipi di dato o identificatori di funzioni differenti;
- coppia di cloni (CP-Clone Pair): coppia di porzioni di codice che sono simili o identiche ad altre;
- gruppo di cloni (CC-Clone Cluster): l'unione di coppie di cloni che hanno porzioni di codice in comune.

Metodi per il rilevamento automatico del plagio

Plagiare un software significa duplicarlo o modificarlo per produrne in breve tempo un altro funzionalmente equivalente ma che sembri apparentemente diverso. La ricerca del plagio richiede tempo poiché richiede di confrontare fra loro coppie di documenti che contengono centinaia o anche migliaia di righe.

Esistono diversi strumenti atti a ricercare segnali di plagio ai differenti livelli, in base al tipo di algoritmo usato. Potremmo raggrupparli in 3 grandi classi: quelli basati sulla *metrica* del software; quelli basati su *token*; quelli basati sulla *semantica* o *strutturali*.

I primi ad essere usati sono stati quelli basati sulla *metrica*, che confrontano i codici analizzando dimensione del programma, complessità, numero di parole, sequenze di istruzioni, operatori e operandi. Questi algoritmi ignorano completamente gli aspetti sintattici del codice perciò non lavorano molto bene nel caso in cui il plagio si manifesti come modifica o ridenominazione di variabili, funzioni e altre porzioni di codice.

I tool basati su *token* confrontano due codici per cercare la presenza di stringhe consecutive uguali; sono i più appropriati per confrontare i codici degli studenti e sono molto usati anche per individuare codice clonato. Il punto di forza di questi metodi è la velocità di computazione, dovuta all'utilizzo di valori hash nei vettori, tuttavia spesso non localizzano le parti veramente simili.

Nel campo delle tecnologie token-based uno dei tool più famosi è **CCFinder**, che suddivide il codice in token specifici e quindi li confronta in base a diverse regole; benché molto diffuso, CCFinder non è in grado di

trattare i casi in cui il codice è stato plagiato cambiando l'ordine delle dichiarazioni poiché si basa sul principio di cercare le sottostringhe comuni più lunghe.

Gli strumenti più sofisticati sono quelli basati sulla *semantica* o *strutturali*, in base ai quali per ogni pezzo di codice si costruisce una struttura corrispondente (di solito un grafo). Tali metodi convertono i programmi in strutture differenti ma preservano ciascun aspetto strutturale del programma originale fino nel dettaglio. Implementarli è più complicato ma i metodi strutturali sono più robusti rispetto ad alcuni tipi di plagio come l'inserimento di codice non rilevante (o *dead code*, cioè quella parte di codice sorgente che viene eseguita ma i cui risultati non sono utilizzati in nessun'altra computazione).

Greedy-String-Tiling (GST) e Running-Karp-Rabin Greedy-String-Tiling (RKR-GST)

Greedy-String-Tiling è un algoritmo che cerca di trovare la sottostringa più lunga comune a due stringhe; per farlo marca le sottostringhe comuni trovate ad ogni iterazione come *tile* per fare in modo che vengano escluse dalle iterazioni successive. Il risultato dell'applicazione dell'algoritmo è un set di *tile* che rappresentano l'impronta digitale del documento.

L'algoritmo Greedy-String-Tiling (GST) riceve in input due coppie di codici (rappresentati come stringhe di token) e li confronta ricorsivamente, come descritto dallo pseudo codice di seguito riportato:

```
SEARCH-PIECE P := FIRST-SEARCH-PIECE  //si confronta ogni
//stringa di un codice con le stringhe (P) dell'altro
TERMINATE := FALSE
DO AGAIN //loop
      P_MAX := SCANPATTERN(P)
      IF P_MAX > 2 × P
            THEN P := P_MAX
            ELSE MARKPIECES(P) //creazione di un tile, cioè
//dell'associazione uno-a-uno tra una sottostringa di P e una
//sottostringa dell'altro codice
      IF P > 2 × MIN_MATCH_PIECE
            THEN P := P DIV 2
            ELSE IF P > MIN_MATCH_PIECE
                  THEN P := MIN_MATCH_PIECE
```

```
                    ELSE TERMINATE := TRUE
UNTIL TERMINATE
```

Si noti che:

- P_{MAX} è il valore massimo di match fra le stringhe considerate ad ogni iterazione;
- `MARKPIECES(P)` è il processo di tiling, nel quale si confrontano i token delle due stringhe elemento per elemento per trovare il match poi i token coinvolti vengono marcati e in questo modo esclusi da confronti futuri

Il metodo proposto è pienamente efficace nell'individuazione del plagio fra codici in cui siano stati modificati i nomi di variabili e funzioni o in cui istruzioni o parti di codice siano state riordinate. La fase di tokenizzazione risulta piuttosto veloce, inoltre tale metodo offre la possibilità di visualizzare i falsi positivi nella fase di confronto fra codici e fornisce ulteriori informazioni sui codici analizzati: l'*inclusione*, cioè la quantità di informazioni di un codice incluse anche nell'altro; la *copertura*, ovvero il valore di quanto un lavoro più ampio risulta coperto da un lavoro più piccolo.

L'algoritmo di Running-Karp-Rabin (RKR) è stato implementato per velocizzare la ricerca delle sottostringhe e ricerca tutte le occorrenze di una piccola stringa (un pattern, *P*) all'interno di una stringa più lunga (un testo, *T*) applicando una funzione hash su tutte le sottostringhe di lunghezza |*P*| e quindi confronta tutti i valori hash di *T* con il valore hash di *P*.

RKR e GST lavorano insieme: RKR viene eseguito su entrambi i documenti e su ogni sottostringa identificata come match da RKR, viene invocato GST per estendere il match oltre i valori hash.

L'algoritmo di winnowing

L'algoritmo di winnowing seleziona un valore hash minimo in una finestra di valori hash; la finestra ha dimensione pari a $w = t - n + 1$, dove t è la soglia garantita (se una stringa è almeno di lunghezza t allora verrà inclusa nel matching) e n è la soglia di rumore (stringhe di lunghezza n o inferiore non saranno incluse nel matching).

Gli alberi sintattici (AST)

Ci sono già diverse tecnologie basate sulla semantica, come **DECKARD** che usa i vettori per memorizzare l'albero sintattico, o anche sistemi online come **JPlag** e **MOSS**, per i quali tuttavia il fatto di dover caricare il codice usando un sistema online può mettere in pericolo la sicurezza del codice stesso. Comunque con le tecnologie basate sulla semantica l'efficienza migliora. Vediamo come.

La creazione dell'albero sintattico comprende 4 step:

1. Preprocessing;
2. Analisi lessicale (attraverso un primo compilatore che legge il codice carattere per carattere, lo confronta con specifiche regole e poi restituisce il token corrispondente);
3. Analisi sintattica (attraverso un secondo compilatore, generatore del parser sintattico, che riceve in input i token creati dal compilatore precedente);
4. Creazione dell'albero sintattico (quando una specifica sequenza di token rispetta una determinata regola, viene allocata memoria, viene generato il nodo dell'albero e poi viene memorizzato, a seconda della sua posizione nel codice).

La struttura della definizione di funzione nell'albero sintattico è un sottoalbero in cui il nodo radice è la *definizione della funzione* e i nodi figli sono i *parametri* e il *body* della funzione. L'albero sintattico generato non è un albero binario. Se la funzione ha *n* parametri, il nodo *parametri* avrà *n* figli. Consideriamo ad esempio il codice sorgente 1:

```
void fun1()
{
	int a = 2;
	fun2(a);
}

int fun2(int x)
{
	printf("%d", x);
}
```

La struttura dell'albero sintattico relativa a questo codice è rappresentata nella figura 3.

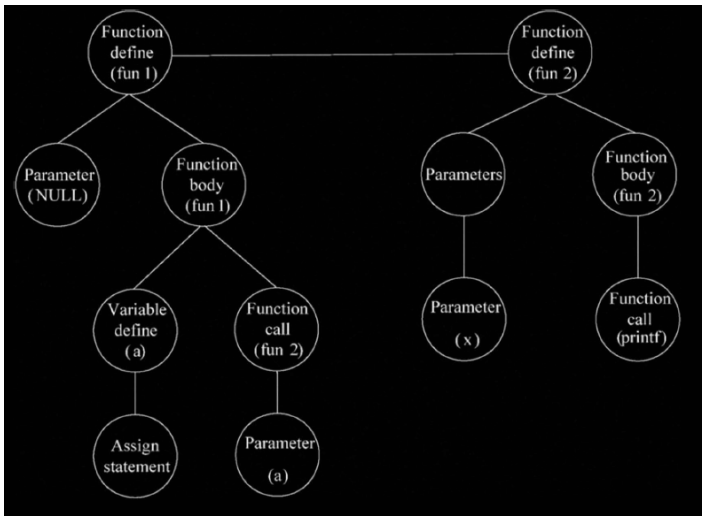

Figura 3 – Struttura dell'albero sintattico generato dal codice 1

A questo punto lo step successivo è il confronto degli alberi sintattici. Una comparazione diretta dei nodi è complicata e poco efficiente perciò si considera il valore hash dell'albero e si calcola il valore in base al tipo di ogni nodo e ai propri sottoalberi. In questo modo si riduce la difficoltà della comparazione senza perdere alcuna informazione relativa all'albero.
Prima di tutto si attraversa l'albero sintattico e si ricavano il valore hash e ogni altra informazione collegata sia per il codice sospetto che per quello originale. Poi si classifica ogni sottoalbero in base al numero dei nodi figli e si memorizza l'informazione in un array di liste.
Successivamente si confrontano i sottoalberi con lo stesso numero di figli nell'albero generato dal codice sospetto e in quello relativo al codice originale. Infine si registrano i sottoalberi con gli stessi valori hash e la loro corrispondente posizione nel codice sorgente.

Un esempio di algoritmo di rilevamento dei cloni basato su AST è quello implementato nel 2014 all'Università di Timisoara. L'algoritmo segue diversi step:

- parsing del codice sorgente e creazione dell'AST corrispondente;
- applicazione della funzione hash su ogni sottoalbero di AST e raggruppamento delle sottosequenze così ottenute in gruppi basati sul loro valore hash;

- applicazione dell'algoritmo di rilevamento dei cloni che include tre sotto algoritmi: algoritmo *di base* (usato per rilevare i cloni nei sotto alberi); algoritmo *di rilevamento nelle sequenze* (rileva i cloni nelle sequenze delle dichiarazioni); algoritmo *di generalizzazione* (ricerca cloni più complessi dei precedenti);
- restituzione di risultati

L'algoritmo *di base* cerca i cloni nei sotto alberi dell'AST, ovvero confronta ogni sottoalbero con ogni altro sottoalbero. Tale confronto richiede molto tempo nel caso di sistemi software complessi; una soluzione a questo problema è usare una funzione hash per codificare i sottoalberi e in base ai valori hash così ottenuti piazzare ogni subtree in un bucket.

L'algoritmo *di rilevamento nelle sequenze* è in grado di rilevare i cloni nelle sequenze delle dichiarazioni, all'interno dell'AST usando l'algoritmo di base. Tutte le sequenze delle dichiarazioni rilevate nel codice sorgente vengono memorizzate in una lista.

L'algoritmo *di generalizzazione* ricerca ulteriori cloni più nascosti, ipotizzando di trovarli vicino a quelli che ha già rilevato. Seguendo questa logica, l'algoritmo visita il nodo padre di ogni clone rilevato e verifica se è anche lui un clone oppure no.

L'analisi del codice clonato usando gli AST offre un'accuratezza elevata ma poiché prevede operazioni complesse ha lo svantaggio di essere più lento di altri algoritmi altrettanto accurati.

Il diagramma in figura 4 illustra l'intero procedimento.

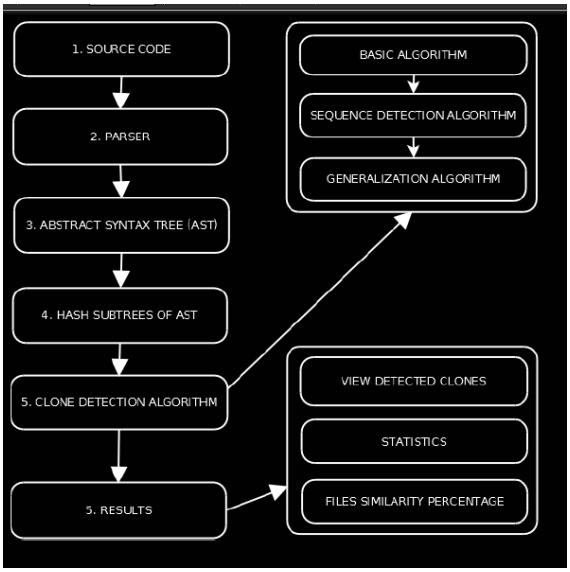

Figura 4 – Architettura del sistema

Algoritmo di base

Si tratta del primo dei 3 algoritmi che costituiscono l'algoritmo di rilevamento dei cloni. L'input è rappresentato dai sottoalberi dell'AST che sono stati sottoposti ad hashing e poi raggruppati in bucket durante il processo di parsing.

Nella figura 5 sono rappresentate le relazioni fra le classi che compongono il modulo dell'algoritmo di base. La classe *ComparisonManager* è il punto di partenza dell'algoritmo; la *LoadBalancer* riceve in input tutti i bucket e assegna ad ogni bucket un thread per la comparazione. Ogni istanza della classe *ComparisonThread* è gestita dalla *ComparisonManager* e viene eseguita quando le viene assegnato un bucket. La classe *Clones* è usata da tutto l'algoritmo di rilevamento dei cloni per gestire i cloni individuati. Memorizza tutti i cloni in una mappa in cui la chiave è il sottoalbero e il valore è rappresentato da una lista di sottoalberi.

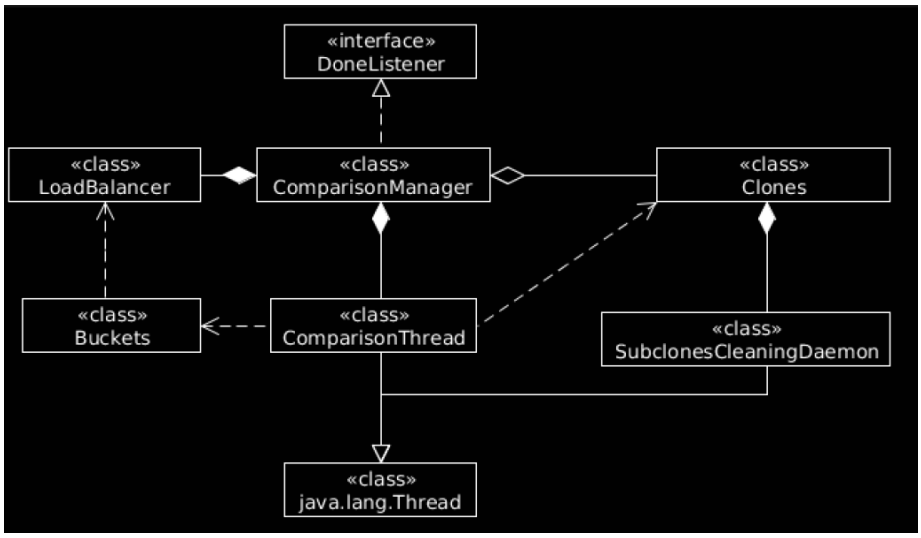

Figura 5 – Algoritmo di base

Algoritmo di rilevamento nelle sequenze

È la parte dell'algoritmo principale che ricerca i cloni nelle sequenze delle dichiarazioni. Con il parsing del codice sorgente, tutte le sequenze vengono memorizzate nella classe *SequenceList.* L'algoritmo si svolge in due fasi: nella prima si cercano i cloni in ogni sequenza; nella seconda si gestiscono le sequenze riconosciute come cloni. La classe *SequenceManager* è responsabile dell'avvio dell'interruzione dell'algoritmo. Una volta terminata l'esecuzione, la lista dei cloni viene fusa con la lista principale e la classe

SubcloneCleaningDaemon provvede a cancellare tutti i nuovi sottocloni. L'algoritmo è rappresentato nella figura 6.

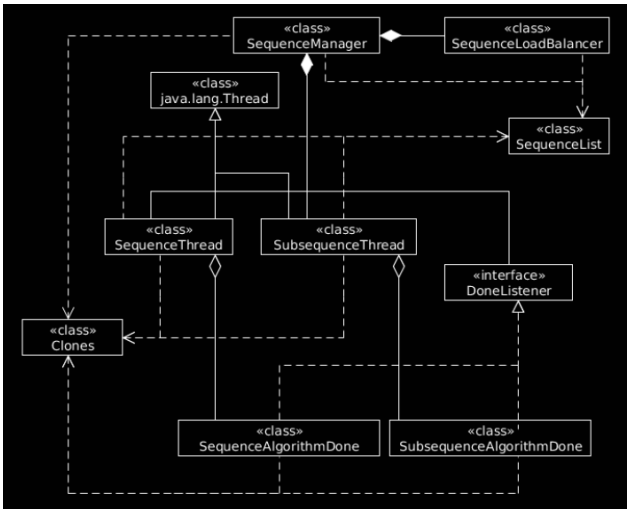

Figura 6 – Algoritmo di rilevamento nelle sequenze.

Algoritmo di generalizzazione

Si occupa di rilevare gli ulteriori cloni ottenuti con l'inserimento o la rimozione di dichiarazioni. Verifica se i nodi padre dei cloni già rilevati sono anch'essi dei cloni. La classe *GeneralizationManager* è la classe principale dell'algoritmo; la *GeneralizationLoadBalancer* calcola il numero di nodi dell'AST e in base a questo valore assegna una chiave al sotto albero della lista principale dei cloni attribuendogli un *GeneralizationThread.*

L'algoritmo di rilevamento dei cloni è stato testato su un gruppo di codici sorgente scritti in linguaggio C, su una macchina Linux. Il tempo di esecuzione generalmente è aumentato all'aumentare della dimensione dei codici da analizzare ma i risultati di rilevamento di cloni sono molto buoni.

L'allineamento delle sequenze

L'allineamento delle sequenze è un metodo per calcolare la relazione esistente tra stringhe che sono state modificate con l'aggiunta di spazi o con lo spostamento di caratteri. Nel metodo di seguito descritto prima di tutto si ottengono dei token facendo un'analisi lessicale del codice, poi si calcola l'allineamento delle sequenze e infine si confrontano i risultati ottenuti per stabilire il valore di similarità. Solitamente l'allineamento delle sequenze include un numero minimo di simboli gap, come in figura 7.

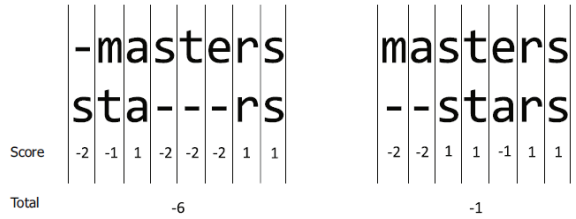

Figura 7 – Esempio di allineamento di sequenze

Diamo una definizione della funzione che calcola il punteggio valutando i simboli *gap* consistenti o inconsistenti per ogni carattere nelle stringhe.

$$w(x, y) = \begin{cases} m(if\ x = y) \\ d(if\ x \neq y\ and\ y \neq -) \\ g(if\ x \neq y\ and\ y = -) \end{cases}$$

dove $x \in \sum (un\ dato\ set\ di\ caratteri),\ y \in \sum \cup \{ - \}$

La matrice $D = |s| \times |t|$, memorizza i risultati intermedi del punteggio di allineamento che possono essere calcolati con la seguente formula ricorsiva:

$$D(i,0) = i \times g$$
$$D(0,j) = j \times g$$

$$D(i,j) = max \begin{cases} D(i-1, j-1) + w(s[i],t[j]) \\ D(i-1, j) + g \\ D(i, j-1) + g \end{cases}$$

Uno dei diversi metodi proposti per calcolare la similarità, è l'indice *sim*, che usa l'allineamento della sequenza per rilevare il plagio nel codice sorgente di un programma. *Sim* calcola la similarità tra due codici sorgente usando un valore normalizzato compreso fra 0.0 e 1.0, seguendo questi step:

1. suddivisione in token (cioè blocchi di testo raggruppati in base alla loro funzione) dei due codici, *S* e *T;*
2. i token di *T* vengono divisi in tante unità, una per ogni funzione;
3. si calcola l'allineamento confrontando ogni token di *S* con ogni unità-funzione di *T* ottenuta nel passaggio precedente;
4. per ogni funzione, si ripete la somma del punteggio di allineamento;

5. normalizzazione.

Il confronto di cui al punto 3 restituisce il valore 2 se i due token confrontati sono gli stessi e i tipi di dato di entrambi sono identificatori; 1 se sono gli stessi ma i tipi non sono identificatori; 0 se i due token non sono gli stessi e i tipi di dato di entrambi sono identificatori; – 2 se non sono identici e uno dei due è un simbolo gap (oppure se non sono identici e i tipi di dato di entrambi non sono identificatori).

Esiste anche l'indice *ilar*, che migliora la capacità di individuazione del plagio. *Ilar* restituisce in output il valore di similarità di due codici, calcolati come segue:

1. suddivisione in token dei due codici, S_1 e S_2. Ottenuti i token di ciascun codice, ogni token viene poi diviso in tante unità, una per ogni funzione;
2. selezionata da S_1 una sequenza di token divisa in unità-funzione, si calcola l'allineamento tra la sequenza selezionata e ogni sequenza di S_2. Il punteggio di allineamento così ottenuto, si usa poi per la normalizzazione;
3. per ogni punteggio di allineamento tra funzioni S_1 e S_2, si cerca quello massimo e poi si calcola il punteggio medio.

I due indici presentati mostrano alcune differenze: mentre *sim* usa elementi lessicali per generare i token, *ilar* si basa su elementi sintattici; *sim* include nei token il valore letterale o un identificatore mentre *ilar* non include tali informazioni. I numerosi esperimenti effettuati confrontando codici plagiati e non dimostrano che entrambi gli indici citati possono rivelarsi molto efficaci nell'individuazione del plagio.

La misura della similarità

Con *similarità* si intende un valore numerico che conta la separazione logica fra due oggetti rappresentati da un set di attributi misurabili. Esistono 36 diversi metodi per misurare la similarità; Julie Baby, Kannan T, Vinod P e Viji Gopal del Department of Computer Science & Engineering della SCMS (School of Engineering and Technology, Ernakulam, Kerala, India) hanno studiato un metodo che sfrutta le diverse tecniche di misurazione della similarità per classificare i documenti copiati. Tale metodo prevede sei moduli distinti:

1. Preparazione dei dati (ovvero separazione dei dati in dati informativi e dati da testare);
2. Conversione (ovvero creazione di una lista di token costituita da funzioni e operatori del linguaggio C in ordine alfabetico);
3. Estrazione degli attributi predominanti (cioè gli attributi che compaiono in almeno il 40% dei file considerati in una stessa famiglia);
4. Calcolo della distanza (ovvero calcolo della similarità tra i file della stessa famiglia utilizzando tutti i 36 metodi di calcolo disponibili. Per ogni diverso metodo, il file con il valore minimo viene scelto come file base per le fasi successive);
5. Definizione della soglia di computazione (ovvero calcolo della distanza tra la distanza di ogni file base e ognuno degli altri file della famiglia);
6. Classificazione dei file testati (ovvero calcolo della distanza tra il file testato e i file base della famiglia per verificare se il file testato appartiene alla famiglia).

Per valutare questo metodo sono state prese in esame quattro famiglie di file, ciascuna contenente 26 codici sorgente; il 97,22% dei file testati è risultato classificato correttamente, il che dimostra che l'utilizzo dei 36 metodi di calcolo della similarità disponibili, consente di individuare efficacemente il plagio fra codici sorgente.

Modelli di uso dinamico dello stack

I software realizzati di recente contengono una quantità rilevante di codice clonato e questo è un problema per diverse ragioni: i cloni gonfiano i costi di manutenzione del software; consumano molte più risorse computazionali; i cambiamenti inconsistenti contenuti nei cloni generano errori e guasti. Park, Son, Kang, Choi e Jeon, studiosi universitari della Korea, hanno proposto un nuovo schema di analisi della similarità fra software basato su un modello di utilizzo dinamico dello stack.

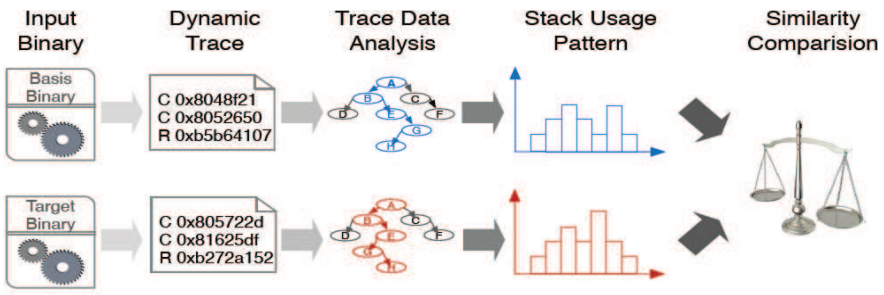

Figura 8 – Rappresentazione della struttura di analisi della similarità fra software basato su un modello di utilizzo dinamico dello stack

Come mostrato in figura 8, uno dei concetti chiave della struttura proposta è il modello di utilizzo dello stack come strumento per il calcolo della similarità tra codici binari. Lo spazio di memoria dedicato al processo è diviso in quattro regioni: testo, dati, stack e heap. Lo stack è utilizzato per ogni chiamata di funzione, gestione di argomenti, indirizzi di ritorno e variabili locali. Utilizzare lo stack per l'analisi della similarità presenta un'ulteriore interessante proprietà. Quando uno sviluppatore copia parti di codice da altri prodotti, ha bisogno di richiamare i codici attraverso le loro interfacce, quindi il monitoraggio dello stack può essere usato per identificare ogni azione di plagio.

Il modello di utilizzo dello stack include quattro componenti:

- raccolta di tracce dallo stack;
- generazione di un DCSG (Dynamic Call Sequence Graph);
- raffinamento del modello di utilizzo dello stack;
- calcolo della similarità.

La raccolta di tracce è finalizzata a collezionare informazioni durante l'esecuzione (ad esempio tipi di istruzioni, indirizzo corrente, indirizzo della funzione invocata, indirizzo di ritorno) per misurare la dimensione dello stack e calcolare quanto aumenta prima e dopo le chiamate a funzione.

Dalla collezione di tracce viene poi creato un grafo, chiamato DCSG, in cui ogni nodo rappresenta una funzione mentre ogni arco corrisponde ad una chiamata a quella funzione; il numero associato ad ogni arco è la dimensione dello stack incrementata dalla chiamata. Il grafo permette di eliminare le funzioni che sono irrilevanti per l'analisi della similarità.

Partendo dal grafo, è possibile costruire il modello di utilizzo dello stack, come quelli rappresentati nella figura 9. Nella figura 9(a) sono rappresentate diverse funzioni: prima di tutto viene invocata la F1 che utilizza 4 byte dello

stack; successivamente la F1 chiama la F2, che a sua volta ha bisogno di 6 byte, perciò la dimensione dello stack aumenta a 10. La figura 9(b) mostra invece un modello alternativo in cui l'asse delle y rappresenta la dimensione dello stack incrementata solo dalle invocazioni di ogni funzione e non da tutte le chiamate.

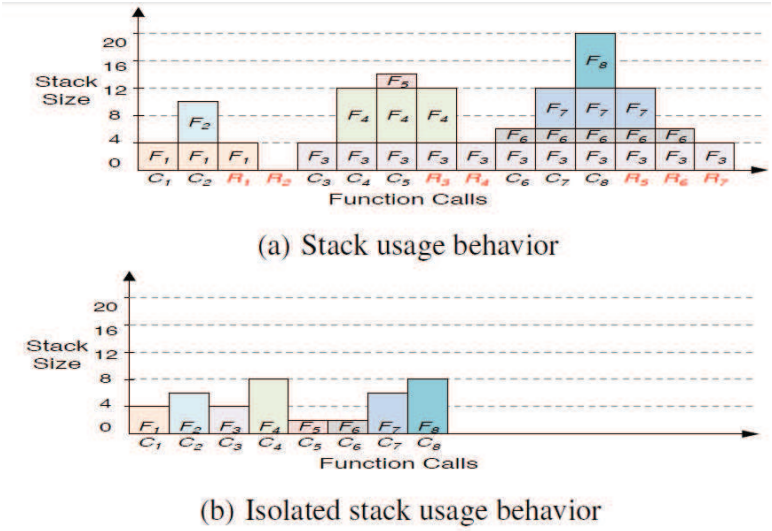

Figura 9 – Esempi di modelli di utilizzo dello stack

La fase finale del procedimento prevede il calcolo della similarità, sulla base del modello creato. Il modello può essere espresso come una stringa in cui ogni carattere corrisponde alla dimensione che occupa nello stack, perciò possiamo confrontare due stringhe usando qualsiasi metodo di calcolo della similarità. In questo studio è stato utilizzato il metodo LCS (Longest Common Subsequence) in cui la similarità è calcolata con la seguente equazione:

$$Sim(SUP_a, SUP_b) = \frac{|\, LCS(SUP_a, SUP_b) \,|}{|\, LCS(SUP_a) \,|}$$

in cui SUP_a e SUP_b sono i modelli di utilizzo dello stack dei codici binari *a* e *b* rispettivamente e $|x|$ è la lunghezza di *x*. Nel nostro esempio il codice sospetto è il codice *b*.

Sulla base degli esperimenti effettuati, il modello proposto consente di calcolare la similarità con maggiore accuratezza rispetto a **MOSS**, uno dei

principali tool di rilevamento del plagio (descritto dettagliatamente nel capitolo 4 di questa tesi); inoltre risulta essere molto efficace nel caso in cui i codici siano stati modificati con porzioni di altro codice.

Le funzioni Okapi BM25

Okapi BM25 è una famiglia di funzioni usate per il recupero di informazioni relative alla similarità all'interno di testi. Queste funzioni misurano la similarità fra una data query Q e un documento D all'interno di una collezione di documenti. La similarità è la somma dei termini comuni a entrambi i documenti.

$$Okapi(Q,D) = \overbrace{w_t}^{\text{IDF}} \times \sum_{t \in Q \cap D} \frac{\overbrace{(k_1 + 1) f_{d,t}}^{\text{TF}}}{\underbrace{K + f_{d,t}}_{\text{IDL}}} \times \frac{\overbrace{(k_3 + 1) f_{q,t}}^{\text{QTF}}}{k_3 + f_{q,t}}$$

in cui:

$$w_t = log_e\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right)$$

$$K = k_1 \times \left((1 - b) + b \times \frac{D_{terms}}{avgD_{terms}}\right)$$

$f_{d,t}$ = *numero di occorrenze del termine t nel documento*

$f_{q,t}$ = *numero di occorrenze del termine t nella query*

D_{terms} = *numero di termini nel documento*

Q_{terms} = *numero di termini nella query*

f_t = *numero di documenti nella collezione in cui occorre il termine*

N = *numero documenti nella collezione*

$avgD_{terms}$ = *numero medio di termini per documento nella collezione*

IDF = Frequenza Inversa del Documento

IDL = Lunghezza Inversa del Documento

TF = Frequenza del Termine

QTF = Frequenza del Termine nella Query

La formula contiene 3 valori costanti sintonizzabili, k_1, k_3 e b, che controllano rispettivamente la frequenza del termine nel documento, quella del termine nella query e la frequenza inversa del documento.

Perché un rilevatore di plagio dia buoni risultati è importante mantenere basso il livello di falsi positivi, con un valore di recupero basso e un valore di precisione alto. Con *recupero* e *precisione* intendiamo le misure della completezza e della correttezza del rilevamento di codice plagiato, secondo le seguenti formule:

$$Recupero\ (R) = \frac{Numero\ di\ positivi\ classificati\ correttamente}{Numero\ di\ positivi\ attuali}$$

$$Precisione\ (P) = \frac{Numero\ di\ positivi\ classificati\ correttamente}{Numero\ di\ positivi\ classificati}$$

L'output ideale di un sistema di rilevamento di plagio è una lista di coppie classificate in base al punteggio di similarità. Un documento plagiato dovrebbe essere in cima alla lista e avere un punteggio di similarità molto alto. Documenti non plagiati invece dovrebbero essere in fondo alla lista, con punteggi bassi.

L'ottimizzazione *particle swarm* può essere usata per migliorare l'efficienza della funzione di similarità *Okapi BM25*, modificando i parametri sintonizzabili della funzione. Dei valori bassi di k_1 e k_3 indicano che versioni lineari di frequenza del termine nel documento e frequenza del termine nella query non possono essere usati per rilevare efficacemente il plagio. I risultati migliori si ottengono aumentando il valore k_1 e decrementando k_3. Ciò dimostra che ai termini nella query dovrebbe essere assegnato un peso maggiore rispetto a quello dei termini nel documento.

La LSA (Latent Semantic Analysis)

La LSA (Latent Semantic Analysis) è una tecnica di raccolta di informazioni che include algoritmi matematici applicati a collezioni di testi. Inizialmente una collezione di testi viene preprocessata e rappresentata come una matrice termine-per-file che contiene i termini e il conteggio delle loro occorrenze nei file. Un algoritmo chiamato SVD (Singular Value Decomposition) decompone questa matrice in matrici separate che catturano la similarità tra i termini e tra i file attraverso varie dimensioni. LSA mira a

cercare le relazioni nascoste (latenti) tra termini differenti. LSA classifica termini e file in una struttura semantica che dipende dalla similarità semantica (da cui Latent Semantic). LSA tipicamente è utilizzato per indicizzare collezioni di testi molto grandi. Una delle caratteristiche più importanti di LSA è che scopre importanti relazioni tra i termini riducendo il rumore nei dati.

LSA è molto abile nel rilevare documenti che contengono cambiamenti semantici. Tratta ogni documento come borsa di parole, quindi, se due documenti sono veramente simili ma contengono cambiamenti lessicali e strutturali, come tentativo di nascondere plagio, questi saranno rilevati da LSA. Un altro vantaggio dell'utilizzo di LSA è che è indipendente dal linguaggio e dunque non ha bisogno di sviluppare parser diversi per i vari linguaggi di programmazione per riuscire a rilevare codici simili.

Gli alberi filogenetici

L'analisi della paternità di un software è molto simile alla procedura di individuazione del plagio attraverso l'albero filogenetico. Studiare il processo di evoluzione di un singolo codice è fondamentale per individuare correttamente eventuale codice plagiato. Consideriamo la filogenesi di un codice omogeneo, cioè le relazioni che intercorrono all'interno di un set di codici sorgenti realizzati da un singolo programmatore. Vogliamo ottenere l'albero filogenetico per questi codici omogenei con una radice comune.

Per tracciare l'evoluzione di un software è necessario definire due criteri: 1) un criterio per stabilire la relazione tra due differenti codici sorgente; 2) un criterio per stabilire la direzione dell'influenza di un codice su altri codici ad esso collegato.

Alcuni ricercatori hanno sviluppato una distanza asimmetrica di similarità partendo dal tradizionale punteggio di similarità basato su allineamento locale. La funzione allineamento trova sub sequenze simili in base al punteggio di similarità fornito dalla matrice di similarità.

La distanza dell'evoluzione è definita in base al matching fra sequenze allineate. Solitamente la similarità normalizzata fra due programmi A e B è definita come rapporto fra il punteggio di matching della regione allineata e il punteggio di matching massimo possibile. Possiamo definire la distanza dell'evoluzione come $dist(A,B) = 1 - S(A,B)$, dove S è la similarità normalizzata. La direzione dell'evoluzione può essere ottenuta direttamente

dalla distanza dell'evoluzione. Per due programmi *A* e *B* esistono due distanze dirette definite come *dist(A,B)* e *dist(B,A)*. La distanza dell'evoluzione è pari a 0 quando i due programmi sono identici.

Una volta calcolate distanze e direzioni dell'evoluzione per ogni coppia di programmi, è possibile costruire l'albero filogenetico. L'albero è calcolato come l'albero di copertura minimo del sottografo senza la rimozione delle direzioni (si ricorda che l'albero di copertura, o *spanning tree*, contiene tutti i vertici di un grafo ma solo un sottoinsieme degli archi, cioè solo quelli necessari per connettere tra loro tutti i vertici con uno e un solo cammino). Successivamente si rimuovono gli archi di ingressi multipli, ovvero quelli che arrivano in entrata ad vertice dell'albero che riceve già due archi in entrata e quindi non può riceverne altri, dal momento che l'albero filogenetico prevede che non vi siano più di due vertici padre. Il risultato finale è una sorta di foresta di copertura più che un singolo albero. L'algoritmo seguente descrive in pseudo codice gli step principali che caratterizzano la costruzione della foresta filogenetica:

```
procedure PHYLOTREE(V)
    F ← FrequencyVector(V);
    M ← SimilarityMatrix(F);
    E ← {(v_i, v_j) | v_i, v_j ∈ V, i ≠ j, D(v_i, v_j) < D(v_j, v_i)};
    T ← MinSpanningTree(Undergraph(G(V,E)));
    while T contains multiple-entry vertex do
        {e_i} ← incoming edges in multiple entry;
        e* ← maximal weight edge in {e_i};
        T ← T − e*;
    end while
    return T;
end procedure
function D(v1,v2)
    (x,y) ← align(v1, v2);
    return 1 − 2M[x,y]/(M[v1, v1] + M[v2, v2]);
end function
```

Algoritmo per la costruzione della foresta filogenetica

La figura 10 mostra un esempio di albero filogenetico costruito usando l'algoritmo di cui sopra; i nodi indicano i codici sorgente e gli archi la direzione dell'evoluzione calcolata.

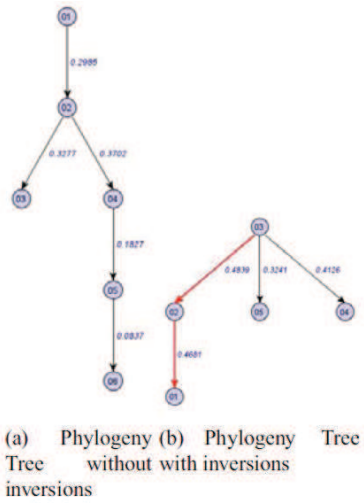

Figura 10 – Albero filogenetico. Ogni nodo rappresenta un codice sorgente e gli archi denotano le distanza della direzione e dell'evoluzione

La figura 10 (b) mostra degli archi evidenziati in rosso: rappresentano una inversione della direzione, che si ha quando uno studente ha sottoposto il codice P_i rimuovendo qualche dichiarazione e ottenendo il codice P_{i-1} così la cancellazione ha determinato un'alta penalizzazione nel calcolo dell'allineamento adattivo locale.

La misura asimmetrica della distanza serve per misurare il grado di evoluzione tra due codici omogenei; tra codici omogenei, la distanza è molto breve. Costruire l'albero filogenetico permette di individuare il plagio in una coppia di programmi, anche nel caso di programmi brevi come i codici sorgenti assegnati come esercitazioni agli studenti universitari.

L'approccio Fuzzy

Il rilevamento del plagio nella programmazione consiste nell'identificazione di file di codice sorgente che contengono frammenti simili o identici. Un approccio di tipo *fuzzy clustering* è una soluzione indipendente dal linguaggio di programmazione e quindi non c'è bisogno di sviluppare nessun parser o compilatore a supporto di questo approccio.

Il rilevamento basato su algoritmi di matching delle stringhe considera le caratteristiche strutturali del codice e per questa ragione tali algoritmi falliscono nell'individuazione di file simili che contengono una parte consistente di codice offuscato. Con un approccio di tipo *fuzzy* (che sta per *sfocato*) usiamo l'approccio statistico della SVD (Single Value Decomposition) per rimuovere il rumore dai codici sorgente. Il processo di confronto comincia con il clustering dei codici, ovvero la suddivisione in

cluster sulla base della similiarità, usando l'approccio clustering *Fuzzy C-Means*; successivamente si ottimizzano i risultati usando il sistema ANFIS (Adaptive Neuro Fuzzy Inference System).

Negli algoritmi basati su matching delle stringhe, prima di tutto si divide il codice in token e quindi ogni codice viene rappresentato come una serie di token di stringhe. L'algoritmo più noto per la tokenizzazione è il Running Karp-Rabin Greedy-String-Tiling (RKR-GST), sviluppato inizialmente per il tool di rilevamento di plagio **YAP3** ma usato anche da altri tool, come **JPlag**. L'ampia letteratura esistente rivela che l'approccio *Fuzzy* non è stato applicato al rilevamento di plagio tra codici sorgente tanto quanto gli approcci basati su matching delle stringhe o sul confronto della struttura.

G. Acampora e G. Cosma della Nottingham Trent University hanno ideato una struttura computazionale innovativa per analizzare i codici sorgente in sospetto di plagio; la figura 11 ne illustra le principali caratteristiche.

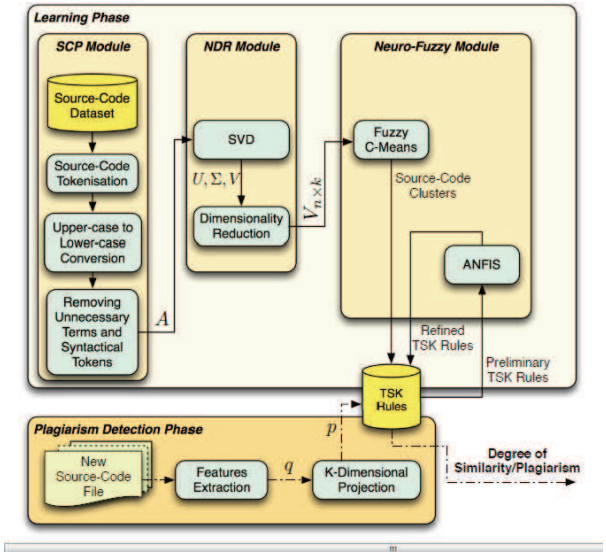

Figura 11 – L'architettura del sistema di rilevamento di plagio fra codici sorgente basato su approccio Fuzzy

Lo scopo del modulo SCP (Source Code Preprocessing) è di preprocessare tutti i codici sorgente in modo da rimuovere i termini e i caratteri non necessari ai fini della computazione, per ridurre la dimensione dei dati e catturare più efficientemente la rappresentazione semantica di ogni codice sorgente. Con l'applicazione del preprocessing, il modulo SCP genera il VSP (Vector Space Model) un modello dello spazio dei vettori che rappresenta con una matrice la frequenza in cui alcuni termini, individuati nel preprocessing, compaiono in ogni codice sorgente; ad ogni riga

corrisponde la frequenza del termine e ad ogni colonna corrisponde un codice sorgente.

Il modulo NDR (Noise Dimensionality Reduction) mira a ridurre la dimensione dei dati e quindi la complessità dello spazio, rimuovendo rumore e dati irrilevanti. Questo compito si realizza con la SVD (Singular Value Decomposition) che identifica e ordina i dati per dimensione sulla base della loro rilevanza nel rappresentare il codice dato in uno spazio dimensionale ridotto. Dopo la SVD, viene creata una rappresentazione semantica dello spazio dedicato ai codici sorgente.

Nel Neuro Fuzzy Module viene applicato il clustering *Fuzzy C-Means* per generare una collezione di cluster in cui ogni cluster contiene un codice sorgente caratterizzato da una collezione simile di identificatori, poi si ottimizzano i dati applicando l'algoritmo ANFIS.

Ridurre la dimensione dei dati permette di rimuovere il rumore e i file che contengono codice simile ma distinto, possono essere sottoposti insieme al procedimento di clustering.

Gli esperimenti eseguiti dimostrano che l'approccio *fuzzy-clustering* è in grado di catturare la similarità qualitativa e semantica degli elementi e permette di superare diversi problemi che si incontrano nel rilevamento automatico di plagio con gli approcci tradizionali come la dipendenza dal linguaggio e il non rilevamento dovuto all'offuscamento del codice.

Caratterizzazione dinamica dei programmi

I core values

Con il rapido sviluppo dell'industria software e l'esplosione dei progetti *open source*, il furto di software è diventata una seria preoccupazione per le aziende di software e per la comunità *open source*.

Gli strumenti finora proposti per il rilevamento di plagio fra codici sorgente presentano diversi limiti: resilienza agli strumenti di offuscamento del codice; abilità nel lavorare direttamente su eseguibili binari di programmi sospetti poiché spesso il codice sorgente dei software sospetti è difficile da reperire. I metodi di rilevamento possono suddividersi in 4 classi, in base a come effettuano il confronto: 1) confronto statico di codici sorgente; 2) confronto statico di eseguibili; 3) diagrammi di controllo dinamici; 4) API dinamiche.

I metodi inclusi nelle prime tre classi sono vulnerabili agli attacchi di offuscamento del codice mentre quelli della classe 2 non sono in grado di lavorare sugli eseguibili ma solo sui sorgenti. L'interessante approccio proposto recentemente da alcuni membri dell'IEEE prevede una caratterizzazione dinamica dei programmi eseguibili, attraverso l'individuazione dei *core values*. Alcuni valori di runtime vengono calcolati al momento dell'esecuzione del programma perciò è complicato modificarli. Tali valori, definiti *core values,* sono calcolati dinamicamente, perciò più difficili da offuscare.

Le tecniche di analisi statica dei codici per il rilevamento di plagio possono essere classificate in cinque categorie:

- quelle basate su stringhe: ogni linea del codice sorgente è considerata una stringa e un codice si considera copiato se la corrispondente sequenza di stringhe fa il match con una parte di codice del programma originale;
- quelle basate su AST: ogni codice viene tradotto in un AST. Se due AST hanno sottoalberi in comune, potrebbe esserci plagio;
- quelle basate su token: un programma viene sottoposto a parsing e poi a tokenizzazione, cioè suddivisione in sequenze di token, che verranno confrontate a coppie per valutare la similarità;
- quelle basate su PDG (Program Dependency Graph): il grafo rappresenta le relazioni tra flusso di istruzioni di controllo e dati e, per stabilire la similarità, si cercano sottografi simili;
- quelle basate su birthmark: con birthmark si intende una caratteristica unica che viene individuata per caratterizzare il programma. Per confrontare i programmi si confrontano i relativi birthmark.

Nessuna di queste tecniche è resiliente all'offuscamento del codice. La tecnica proposta dall'IEEE invece risulta resiliente all'offuscamento del codice tanto che potrebbe essere utilizzata anche nei casi di *app repackaging* ovvero plagio di software nel mondo mobile che interessa i principali mercati di app, Google Play e iTunes.

I valori runtime di un programma sono definiti come i valori generati dall'esecuzione delle istruzioni macchina; i *core values* sono costruiti a partire dai valori runtime. Si considerino v_P il valore runtime del programma

P che riceve *I* in input e *f* una funzione di trasformazione, i valori *non core* si possono definire come come quelli che godono delle seguenti proprietà:

1. se v_P non deriva dall'input *I,* v_P non è un *core value* di *P;*
2. se v_P non è incluso nel set di valori runtime della funzione *f(P),* v_P non è un *core value* di *P.*

In questa ricerca come metodo di rilevamento del plagio è stato utilizzato VaPD, che non richiede l'accesso ai codici sorgente.

Poiché non tutti i valori associati all'esecuzione di un programma sono *core values* è importante selezionare i tipi di valori inclusi in una sequenza di valori: meglio includere solo i valori che derivano dall'esecuzione di un'istruzione di aggiornamento (come *add*) e che rispettano rigidamente la semantica del programma.

Una sequenza di valori costruita con un'analisi dinamica potrebbe contenere diversi valori *non core* prodotti da computazioni intermedie; per eliminare i valori *non core* gli autori della ricerca propongono un procedimento di raffinamento sequenziale, basato sulla seguente regola: se $i_{k,v}$ è la k-esima istruzione di aggiornamento della variabile v, possiamo evitare di considerare l'output di $i_{k,v}$ se la variabile v viene cancellata nell'istruzione immediatamente successiva $i_{k+1,v}$. Questo procedimento sarà ripetuto finché non sarà eseguita la prima istruzione che legge n e aggiorna il valore di una variabile $\neq n$.

Gli indirizzi di memoria o i valori dei puntatori memorizzati nei registri sono transitori, quindi non devono essere considerati; per rilevare questi valori si è utilizzato un sistema che monitora i cambiamenti della memoria allocata durante l'analisi del programma e annota in una lista le pagine di memoria coinvolte. Se un valore runtime viene trovato all'interno della lista, il sistema VaPD lo scarta.

Il metodo scelto per la misura della similarità è LCS, basato sull'individuazione della sottosequenza comune più lunga e definito come segue:

$$Sim(v_P, v_S) = \frac{| LCS(v_P, v_S) |}{| v_P |}$$

dove v_P e v_S sono rispettivamente la sequenza di valori ottimizzata del programma principale e la sequenza di valori del programma sospetto e

$|LCS(v_P, v_S)|$ rappresenta la lunghezza della sottosequenza comune più lunga.

Diversi test hanno dimostrato che i metodi basati su *core values* sono resilienti rispetto a numerose tecniche di offuscamento del codice ma sensibili al riordinamento del codice. Per superare questo limite si propone l'utilizzo di un grafo VDG (Value Dependence Graph), costruito a partire dalla traccia di esecuzione di un programma. Data una sequenza di valori runtime $T = \{v_0, v_1, ..., v_{n-1}\}$, per ogni coppia di valori runtime v_i e v_j , con $i < j$, se $v_i \in \Phi(v_j)$ cioè appartiene all'insieme degli antenati diretti di v_j, allora aggiungiamo un arco (v_i, v_j) al nostro grafo. Per migliorare l'efficienza del sistema, si possono anche scegliere i percorsi più significativi all'interno del grafo raggruppando in uno solo tutti i percorsi che trasformano un input iniziale in un output finale. Inoltre possiamo rimuovere dal grafo i nodi e gli archi non utilizzati.

La tecnica proposta presenta anche diversi limiti: VaPD può estrarre la sequenza di valori da una piccola parte del programma e non è detto che riesca a estrarre i valori valutando l'intero scope del programma principale; VaPD non può essere applicato se l'algoritmo del programma è troppo semplice; VaPD può confondere falsi positivi e falsi negativi; il sistema non è stato ancora sufficientemente testato su problemi reali per poterne affermare l'efficacia.

DKISB

I software *open source* invitano gli utenti ad utilizzare, modificare e distribuire il software sotto certi tipi di licenza. Tuttavia molte aziende incorporano software di terze parti senza rispettare i termini di queste licenze favorendo i propri interessi commerciali.

Per prevenire il furto di software, alcuni programmatori usano tecniche di offuscamento per rendere oscuro il loro codice; in questo modo però non ostacolano la copia diretta del codice. Alcuni studiosi del Dipartimento di Scienza e Tecnologia dell'Università di Xi'an Jiaotong in Cina, propongono un nuovo metodo, detto *birthmarking*, che estrae un set di caratteristiche che possono essere usate per identificare univocamente un programma; due programmi con lo stesso *birthmark* indicano la presenza di plagio. Il problema di questa tecnica è l'estrazione di caratteristiche che siano

veramente rappresentative e che rimangano le stesse anche dopo l'offuscamento del codice.

La presente ricerca propone un *birthmark* basato su una sottosequenza di istruzioni eseguite. Le istruzioni chiave considerate sono quelle che generano un aggiornamento di valori e che riflettono il modo in cui gli input vengono processati. Si utilizzano esecuzioni multiple e si applica l'algoritmo *k-gram* per calcolare la similarità fra i *birthmark*.

Un *birthmark* è un set di caratteristiche estratte da un programma che riflettono le proprietà intrinseche del programma. Nella definizione classica p_B è detto *birthmark* nel programma p se e solo se soddisfa le seguenti condizioni:

- p_B è ottenuto solo dallo stesso p;
- se il programma q è una copia di $p \Rightarrow p_B = q_B$

Se parliamo di *birthmark* dinamico allora la definizione cambia: se p è un programma e I il suo input, allora p_B^I è il *birthmark* dinamico di p se e solo se sono soddisfatte le seguenti condizioni:

- p_B^I è ottenuto solo dallo stesso p quando viene eseguito p con I come input;
- se il programma q è una copia di $p \Rightarrow p_B^I = q_B^I$

In pratica il rilevamento del plagio dipende dalla soglia ε e dalla funzione *sim* che calcola la similarità tra p_B e q_B secondo la seguente equazione:

$$sim(p_B,q_B) = \begin{cases} \geq 1-\varepsilon & positivo: p \text{ è una copia di } q \\ \leq \varepsilon & negativo: p \text{ non è una copia di } q \\ altrimenti & inconcludente \end{cases}$$

Un *birthmark* di alta qualità deve rappresentare la semantica del programma, come la sequenza di istruzioni generate durante l'esecuzione del programma ma l'intera sequenza genererebbe un *birthmark* troppo lungo per le future analisi; meglio che le istruzioni chiave siano costituite da una piccola porzione della sequenza, secondo la seguente definizione.

Considerata *trace(p,I)* come la sequenza delle istruzioni generate durante l'esecuzione del programma p con l'input I, allora ogni istruzione c in *trace(p,I)* è considerata istruzione chiave se e solo se soddisfa le seguenti condizioni:

- *c* è un'istruzione di aggiornamento di valori;
- *c* è un'istruzione collegata all'input.

Usiamo *key(p,I)* per indicare una sotto sequenza di *trace(p,I)* costituita esclusivamente da istruzioni chiave.

Nonostante il fatto che le istruzione chiave riducano il numero di istruzioni da analizzare, la dimensione della sequenza di istruzioni chiave può essere imprevedibile. Per vincolare la sequenza chiave ad una lunghezza k, utilizziamo l'algoritmo *k-gram*.

Data la sequenza di operazioni eseguite $opcode = \langle e_1, e_2, ..., e_n \rangle$, considerata la lunghezza predefinita *k*, la sottosequenza $opcode_j(k) = \langle e_j, e_{j+1}, ..., e_{j+k-1} \rangle$, con $(1 \leq j \leq n - k + 1)$, può essere generata facendo scorrere la finestra sopra a *opcode* di un passo alla volta.

Infine definiamo il *birthmark* esaminato nella presente ricerca DYKIS:

Sia $key(p, I) = \langle ins_1, ins_2, \ldots, ins_n \rangle$ una sequenza di istruzioni chiave registrata durante l'esecuzione di un programma *p* con in input il vettore *I*; sia $opcode = \langle e_1, e_2, ..., e_n \rangle$ la corrispondente sequenza di operazioni eseguite; e_i è l'operazione corrispondente a ins_i. Sia $gram(p, I, k) = \langle g_j | g_j = \langle ej, e_{j+1}, ..., e_{j+k-1} \rangle \rangle$, con $(1 \leq j \leq n - k + 1)$ la sequenza *k-gram*. Si definisce il set di coppie di valori chiave $p_B^I(k) = \{\langle g_m, freq(g_m) \rangle \mid g_m \in gram(p, I, k) \wedge \forall_{m1 \neq m2} \cdot g_{m1} \neq g_{m2}\}$, dove $freq(g_m)$ rappresenta la frequenza di g_m che compare in *gram (p, I, k),* come il *birthmark* di *p* con input *I* basato su DYnamic Key Instruction Sequence.

Se *p* e *q* sono il codice principale e quello sospettato, data una serie di input $\{I_1, ..., I_n\}$, quando eseguiamo i programmi otterremo *n* coppie di *birtkmark* DYKIS $\{(p_{B1}, q_{B1}), ..., (p_{Bn}, q_{Bn})\}$. L'esistenza del plagio fra i codici *p* e *q* viene decisa in base al punteggio di similarità calcolato fra le corrispondenti coppie di *birthmark* DYKIS.

Sono stati condotti diversi esperimenti per valutare i *birthmark* DYKIS.

Il valore del parametro *k* nell'algoritmo *k-gram* gioca un ruolo importante; in base alle indagini effettuate la cosa migliore è fissare un valore di k compreso fra 3 e 10, per diverse ragioni:

- un algoritmo con valori piccoli di *k* è più efficiente;
- un valore di *k* inferiore a 3 rischia di generare falsi positivi;
- un valore maggiore di 10 genera molti falsi negativi.

Negli ultimi anni il plagio di interi programmi per mobile è diventato un problema serio; è opportuno pensare di ottimizzare DYKIS per poterlo utilizzare efficacemente anche in questo campo.

Rilevamento di plagio negli algoritmi

Il plagio degli algoritmi è un problema poco affrontato. La ricerca di Zhang, Jhi, Wu, Liu e Zhu propone due approcci dinamici basati su valori per rilevare il plagio di algoritmi: *N-version* e *annotazione*.

Negli ultimi anni il plagio ha sollevato grande preoccupazione per la tutela della proprietà intellettuale. Mentre esiste molta letteratura sul plagio del codice, poco è stato scritto sul plagio degli algoritmi.

Qual è la differenza? Se due software sono sviluppati in maniera indipendentemente da due aziende diverse che usano lo stesso algoritmo non esiste plagio di software perché le aziende stanno agendo in maniera indipendente. Rilevare il plagio di un algoritmo è un'operazione molto più impegnativa rispetto al rilevamento di plagio nel codice. Un algoritmo può essere scritto in linguaggi diversi e in modi diversi, a seconda della creatività dell'autore, tuttavia gli autori ritengono che un algoritmo determini delle caratteristiche distintive a livello di codice che non possono essere occultate. Queste caratteristiche rappresentano la firma di riconoscimento di un algoritmo perciò è importante valutare qual è una buona firma per un algoritmo e come estrarre la firma da un algoritmo. I due approcci *N-version* e *annotazione* consentono proprio di estrarre queste caratteristiche partendo dal codice.

Dato un codice sorgente con almeno un'implementazione di un algoritmo principale, si cerca di capire a fondo il funzionamento dell'algoritmo per poi verificare se è coinvolto o meno in un caso di plagio.

Il primo step è identificare e rappresentare la firma di un algoritmo. Esistono molte proprietà candidate a caratterizzare un algoritmo:

- la sequenza / il grafo delle chiamate di sistema, anche se spesso sono uguali per algoritmi molto diversi;
- la sequenza / il grafo delle chiamate a funzioni;
- il grafo del flusso del controllo (CFG), che rappresenta il flusso del controllo tra i principali blocchi del programma;

- il grafo del flusso di dati, usato per rappresentare il flusso dei dati fra i principali blocchi del programma. Questa proprietà può essere manipolata molto facilmente, dividendo i blocchi o inserendo blocchi inutili;
- dipendenza delle istruzioni dal livello dei dati, ovvero la relazione tra i valori di runtime.

I valori di runtime sono i valori che compaiono nell'output delle istruzioni macchina eseguite durante il runtime. Dato un input, il valore fondamentale di un algoritmo (c.d. *core value*) è il subset dei valori runtime della sua implementazione.

Come estrarre questi valori? Esistono due differenti approcci: il primo consiste nell'identificazione dei valori nel caso in cui sappiamo già quali sono; il secondo caso è quello in cui sappiamo quali sono i valori da cercare ma non sappiamo quali non lo sono, perciò possiamo scartare i valori che non appaiono fondamentali e sperare che quelli che lasciamo siano effettivamente valori *core*.

Approccio N-version

Usiamo questo approccio per filtrare le diversità mentre analizziamo implementazioni diverse dello stesso algoritmo conservando i valori di runtime persistenti. Individuiamo i valori *non core* per poi ricercarli e rimuoverli dai valori di runtime. Se P_A è un'implementazione dell'algoritmo *A*, v_{PA} è il valore di runtime di P_A che riceve *I* in input e Q_A è ogni altra implementazione di *A*, allora i valori *non core* soddisfano almeno una di queste proprietà:

- se v_{PA} non deriva da *I*, v_{PA} è il valore *non core* di *A;*
- se v_{PA} non è incluso nel set di valori runtime di Q_A che riceve *I* in input, v_{PA} è il valore *non core* di *A*.

L'architettura dell'approccio *N-version* è rappresentata in figura 12.

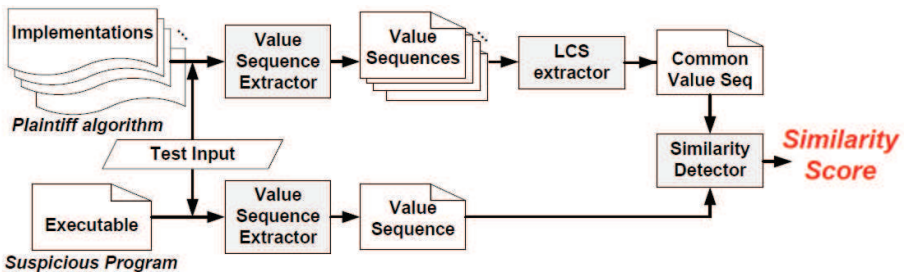

Figura 12 – Rappresentazione dell'approccio *N-version*

Per ogni implementazione l'estrattore di sequenze di valori estrae una sequenza di valori ridefinita che contiene solo i valori di runtime derivati dall'input. Per raffinare l'estrazione si possono considerare ulteriori schemi:

- istruzioni macchina aggiornate (cioè quelle istruzioni che modificano il valore in input);
- riduzione sequenziale (cioè esclusione di quei valori che sono stati aggiornati durante l'esecuzione);
- raffinamento basato su ottimizzazione (applicato solo agli algoritmi principali);
- rimozione dell'indirizzo (gli indirizzi di memoria non sono valori *core* perché possono variare in base alle trasformazioni tecniche binarie).

L'estrattore LCS genera la LCS (Longest Common Subsequence), la sequenza di tutti i valori raffinati delle *N* implementazioni dell'algoritmo. Questa sub sequenza comune alle implementazioni è considerata il *core value* dell'algoritmo.

Il rilevatore di similarità confronta la LCS delle N implementazioni con la sequenza raffinata di valori del programma sospetto.

Annotazione

La *N-version* richiede molte implementazioni indipendenti dell'algoritmo e questa esigenza può rappresentare un limite. L'idea base dell'approccio *annotazione* è di utilizzare un'informazione ausiliare per identificare le dichiarazioni critiche che generano i valori *core*. Lo schema è mostrato nella figura 13.

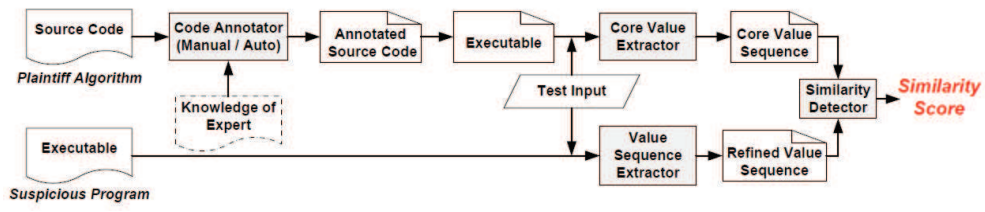

Figura 13 – Rappresentazione dell'approccio annotazioni

L'annotatore di codice aggiunge annotazioni al codice sorgente o automaticamente o in base alle conoscenze di esperti del settore. Queste

annotazioni identificano quali variabili in quali dichiarazioni conterranno dei valori *core* durante l'esecuzione.
Scegliamo di annotare nel codice sorgente anziché nel codice binario per diverse ragioni. Prima di tutto ogni valore *core* è il valore runtime di una variabile nel codice sorgente, quindi un'adeguata annotazione a livello di sorgente è sufficiente per estrarre tutti i valori *core*. Inoltre il codice sorgente può evidenziare numerosi valori *non core* generati durante l'esecuzione. Infine il sorgente viene scritto dai programmatori in base alla descrizione dell'algoritmo mentre il codice binario è generato dai compilatori a partire dal codice sorgente.
Una volta estratta la firma dell'algoritmo, occorre misurare la similarità in termini di proporzione della presenza di valori comuni nelle sequenze di entrambi i programmi. Data $|s|$ la lunghezza della sequenza s, il punteggio di similarità si calcola con la seguente formula:

$$\textit{punteggio di similarità} = \frac{|\textit{sequenza di valori comuni}|}{|\textit{sequenza di firme}|}$$

Una tecnica di confronto alternativa è la VDG (Value Dependence Graph), nella quale vengono registrati i valori di runtime con la loro dipendenza. Ogni valore rappresenta un nodo del grafo e la dipendenza tra i valori rappresenta un arco.
Numerosi esperimenti mostrano che gli approcci basati su valori non sono applicabili a tutti gli algoritmi poiché fanno affidamento sull'estrazione di valori runtime che si nascondono in mezzo a tante istruzioni *non core*.
I valori *core* possono essere un utile ausilio anche nell'individuazione di plagio del codice inoltre l'estrazione della firma può contribuire alla tutela della proprietà intellettuale.

Rilevamento di plagio tramite bytecode: il sistema $PINT_B$

La maggior parte dei programmi per l'individuazione del plagio confronta i codici sorgente e rileva le coppie di programmi plagiati. Usare il codice sorgente per individuare il plagio può compromettere la sicurezza del codice stesso. In un articolo scritto dai ricercatori coreani Ji, Woo e Cho si propone

una nuova tecnica di individuazione del plagio nei programmi Java, che usa il *bytecode* senza fare riferimento al codice sorgente. La procedura di individuazione del plagio usando il *bytecode* prevede due fasi principali. Nella prima si dividono in token i file class di Java analizzando i metodi presenti nel codice. In seguito si confrontano i token usando l'allineamento locale adattivo. In base ai risultati dell'esperimento, possiamo concludere che le distribuzioni della similarità dei codici sorgente e di quella del bytecode sono molto simili.

Il sistema di individuazione del plagio tramite *bytecode* può essere usato come strumento di verifica preliminare prima dell'individuazione del plagio tramite confronto dei codici.

Il sistema di confronto dei *bytecode* non richiede i sorgenti di Java e confronta direttamente i file class. La procedura è simile a quella di comparazione di codici sorgente: 1° step) linearizzazione e 2° step) comparazione. Nella linearizzazione si estraggono solo le istruzioni binarie ricavabili dal main e si registrano in una sequenza. Durante la comparazione si confrontano le due sequenze di *bytecode* usando un tipico algoritmo di comparazione, cioè l'allineamento locale adattivo.

L'individuazione di plagio tramite *bytecode* può essere parzialmente robusta di fronte agli attacchi di tipo 1 e 2 come sopra descritti, perché gli aspetti testuali inclusi i commenti vengono cancellati nel codice binario. Tuttavia nei confronti degli altri tipi di plagio l'accuratezza di questo metodo è equivalente a quella dell'individuazione basata su confronto di codici sorgente.

Panoramica del sistema

Parliamo del sistema $PINT_B$ (Plagiarism INvestigating Tool using Bytecode), che opera in due fasi. Nella prima, dato un set di file class, vengono generate delle sequenze di token. Nella seconda, si valuta la similarità tra i due programmi Java presi in esame attraverso l'algoritmo di allineamento locale. La figura 14 illustra il funzionamento del sistema $PINT_B$.

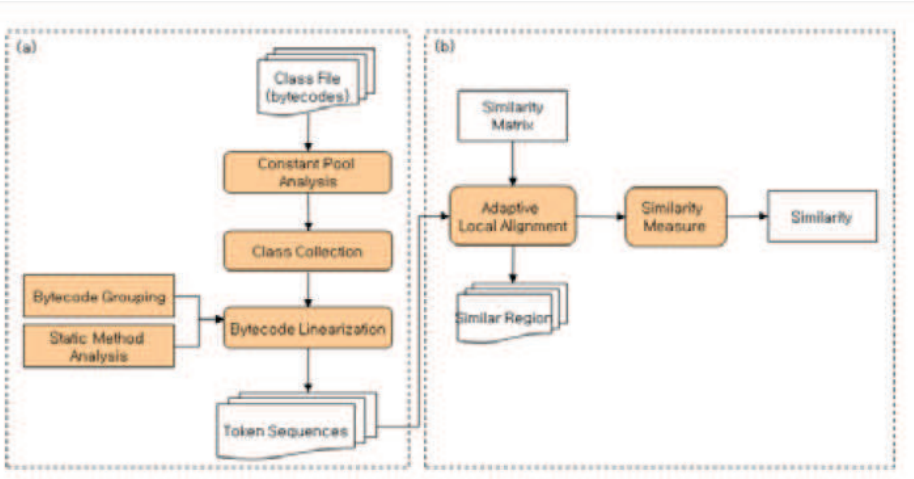

Figura 14 – Architettura del sistema $PINT_B$. Il sistema prevede due step: a) generazione di sequenze di token (fase 1) e b) valutazione della similarità per rilevare eventuale plagio (fase 2)

Nella fase 1 $PINT_B$, ricevuti in input due programmi java, raggruppa tutti i file class legati ad ogni programma java che contengono informazioni necessarie all'esecuzione del programma stesso. Successivamente $PINT_B$ genera una sequenza di token a partire dal *bytecode* usando un metodo statico di analisi che esegue il *bytecode* a livello di metodi ed estrae i token precedentemente definiti da ogni metodo, seguendo l'ordine di esecuzione dei metodi.

Nella fase 2, il sistema riceve in input un file che contiene la sequenza di token generati nella fase 1. A questo punto $PINT_B$ valuta la similarità fra tutte le coppie di sequenze di token usando un algoritmo di allineamento adattativo locale.

La struttura del file class

Il *bytecode* è un set di istruzioni binarie. Generalmente il *bytecode* consiste in un codice operativo con i suoi operandi che viene eseguito sulla macchina virtuale. Il *bytecode* Java è costituito da 203 istruzioni, generate dall'esecuzione del file class sulla macchina virtuale attraverso il compilatore java.

Un programma java è un set di file class. Ogni file è costituito da un header, un gruppo di costanti, dei campi, una tabella dei metodi e una tabella degli attributi. La sezione header comincia sempre con il numero magico, che identifica il formato del file class e il cui valore è 0xCAFEBABE. Le informazioni sulla versione e le costanti sono indicate di seguito al numero

magico. Poi si elencano le informazioni sui campi e sulla tabella dei metodi e infine, in fondo al file class, si trova la tabella degli attributi. Tutte queste informazioni sono utilizzate per il debugging dalla java virtual machine.
La figura 15 mostra la struttura del file class di java.

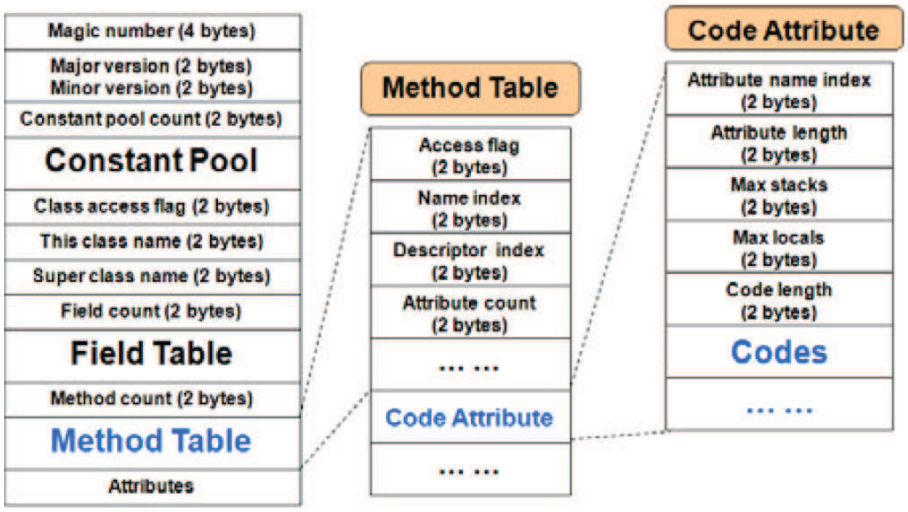

Figura 15 – La struttura di un file class di java

Il *bytecode* risiede nell'attributo codice contenuto nella tabella dei metodi. Per esempio se nel file Pa.java sono dichiarati quattro metodi main() e tre metodi definiti dall'utente, nel file Pa.class ci saranno cinque tabelle di metodi, incluso il costruttore di default. La struttura dei campi e della tabella dei metodi è la stessa; questa tabella include cinque campi che hanno accesso a flag, nome e descrittore dell'indice, contatore degli attributi e tabella degli attributi. Java usa sette tipi di tabella degli attributi. La tabella dei metodi può contenere anche tabelle degli attributi che possono essere a loro volta eseguite. L'attributo codice è la più importante tabella di attributi perché contiene i *bytecode* per l'esecuzione dei metodi. collegate esse stesse all'esecuzione. $PINT_B$ usa questi *bytecode* per rilevare il plagio.

Linearizzazione del bytecode

La linearizzazione del *bytecode* è lo step che genera una sequenza di token a partire da un file class analizzandone i *bytecode*. $PINT_B$ usa il raggruppamento dei *bytecode* e il grafo delle chiamate per rilevare i metodi.
Il raggruppamento dei *bytecode* è una procedura che raggruppa i *bytecode* simili in base alla funzione. Per migliorare le prestazioni della JVM, in Java sono stati definiti diversi *bytecode* che eseguono la stessa operazione. Per

esempio in Java esistono i *bytecode* duplicati *iload_1, iload_2, iload_3* che sono simili al codice *iload* che carica il valore di un gruppo di variabili locali in cima allo stack. La differenza fra questi codici sta solo nel fatto di avere o meno operandi. Queste istruzioni sono usate opzionalmente dal compilatore in base alla posizione delle variabili locali nel codice sorgente. La figura 16 mostra la tabella di raggruppamento dei *bytecode* di $PINT_B$.

Group	Bytecode	attribute
bc_add	iadd, fadd, dadd	add operation
bc_push	bipush, sipush, …	push a constant to stack
bc_load	iload, iload_1, iload_2, fload, …	move data to stack
bc_store	istore, istore_1, istore_2, fstore, …	store data to variable
bc_invoke	invokevirtual, invoke_static, …	method invocation
… …	… …	… …

Figura 16 – Tabella di raggruppamento dei bytecode. $PINT_B$ usa 59 gruppi di *bytecode* per rilevare il plagio

Il raggruppamento dei *bytecode* migliora l'accuratezza della rilevazione riducendo la diminuzione della similarità che si verifica quando c'è una piccola differenza tra i *bytecode*. Cambiare la posizione delle variabili locali è un metodo di plagio ampiamente usato perciò $PINT_B$ linearizza questi *bytecode* nella stessa sequenza di token. La figura 17 mostra un semplice esempio della differenza fra i *bytecode* dovuta allo spostamento di variabili locali.

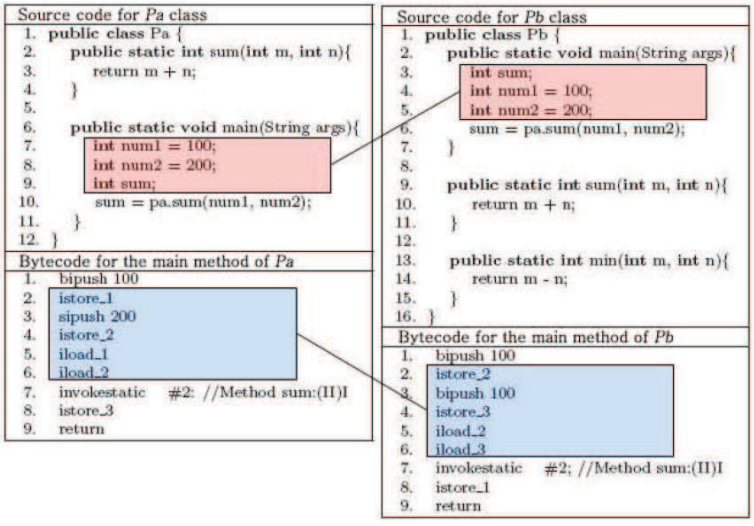

Figura 17 – Un semplice esempio della differenza di bytecode dovuta alla posizione della variabili locali

Come si nota in figura 17, anche se i due programmi differiscono solo per la posizione delle variabili, i rispettivi *bytecode* sono differenti. $PINT_B$ linearizza questi bytecode come mostrato in figura 18.

Figura 18 – I risultati della linearizzazione di due programmi P_a e P_b

$PINT_B$ tiene conto anche delle caratteristiche del flusso del programma utilizzando il grafo delle chiamate nella linearizzazione. Nella figura 17 si nota che l'ordine dei metodi è differente; dividendo il codice in sequenze di token il *bytecode* di P_a sarebbe stato linearizzato seguendo l'ordine *sum()* → *main()* mentre P_b sarebbe stato linearizzato in *main()* → *sum()* → *min()*, inserendo anche il metodo *min()* che è inutilizzato perciò potrebbe essere un segnale di plagio.

$PINT_B$ può individuare efficacemente il plagio nei metodi collegati.

Valutazione della similarità dei programmi

$PINT_B$ valuta la similarità di tutte le coppie di sequenze di token e cerca le regioni simili in una coppia di programmi. La similarità indica in generale in cosa i due programmi sono simili e le regioni simili indicano le regioni in sospetto di plagio.

La similarità dei *bytecode* è calcolata con un algoritmo di allineamento locale adattivo, attualmente utilizzato per individuare programmi plagiati all'ICPC (International Collegiate Programming Contest) organizzato dalla ACM. L'algoritmo originale di allineamento locale sul quale il nostro algoritmo si basa è quello introdotto da Smith e Waterman, che è stato utilizzato per allineare due sequenze di DNA. È basato su una matrice di punteggi. La figura 19 mostra un esempio del funzionamento base dell'algoritmo di allineamento locale.

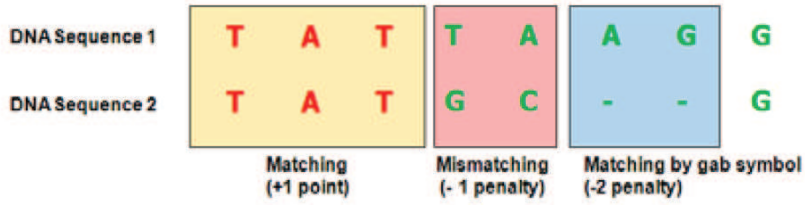

Figura 19 – Un semplice esempio dell'algoritmo di allineamento locale di Smith e Waterman

Consideriamo che $D=\{d_1, d_2, ..., d_m\}$ sia una sequenza di DNA costituita da m DNA. Se i DNA sono due, la matrice di punteggi derivante dall'algoritmo di allineamento locale avrà +1 punto se le due sequenze sono uguali, – 1 punto se sono diverse e – 2 punti se il matching si ottiene inserendo un valore ulteriore detto gap, nel caso in cui caratteri identici o simili siano allineati in colonne successive (d_i o d_j = '_'), dove i e j sono gli indici dei DNA. Sebbene i due DNA presentino regioni differenti, l'allineamento locale continua a calcolare la similarità inserendo dei valori con alcune penalità.

La strategia di base dell'allineamento locale adattivo è che il punteggio di matching delle parole chiave debba registrare la frequenza delle parole chiave. Più precisamente, si attribuisce un punteggio alto se le parole sono meno frequenti e un punteggio più basso per le parole che compaiono più frequentemente.

La stessa regola è applicata per le penalizzazioni nel caso di mismatching. Dal momento che è raro vedere parole con frequenza bassa che vengono usate da due programmi allo stesso tempo, due programmi che usano parole con frequenza bassa dovrebbero sempre essere considerati simili.

La parte più importante dell'allineamento locale adattivo è la matrice di similarità. Date due parole chiave k_i e k_j, in cui una di queste potrebbe essere un valore gap, la matrice di similarità degli elementi $M(k_i, k_j)$ rappresenta il punteggio positivo o negativo in presenza o meno di matching.

Per determinare la matrice adattiva di similarità, la frequenza della parole dovrebbe essere calcolata in anticipo. Consideriamo che $P = \{ p_1, p_2, ..., p_n \}$ sia un gruppo di programmi costituito da n programmi e assumiamo che $occur(p,k)$ conti in numero di occorrenze della parola chiave k nel programma p. Il numero totale di occorrenze di k nel gruppo di programmi P è definito come $occur(P,k) = \Sigma_{p \in P} occur(p,k)$. In base a questa definizione

la frequenza f_i^P della parola chiave k_i nel gruppo di programmi P è definita come $f_i^P = occur(P,k_i) / \Sigma^r_{j=1} occur(P,k_j)$.

Poiché il denominatore $\Sigma^r_{j=1} occur(P,k_j)$ rappresenta la somma del numero di occorrenze di tutte le parole chiave, la frequenza f_i^P della parola chiave k_i oscilla fra 0 e 1 ($0 \leq f_i^P \leq 1$).

Usando la frequenza delle parole chiave come definita sopra, la matrice adattiva di similarità M^P può essere definita come segue:

$$M^P(k_i,k_j) = \begin{cases} -\alpha \cdot log2\,(f_i^P \cdot f_j^P) & se\ k_i = k_j \\ \beta \cdot log2\,(f_i^P \cdot f_j^P) & se\ k_i \neq k_j \\ 4\beta \cdot log2\, f_i^P & se\ k_j\ è\ un\ gap \\ 4\beta \cdot log2\, f_j^P & se\ k_i\ è\ un\ gap \\ -\infty & se\ k_i\ e\ k_j\ sono\ gap \end{cases}$$

α e β sono parametri di regolazione e la somma di questi parametri è settata a 1 ($\alpha + \beta = 1$). Possiamo aggiustare i pesi relativi per il punteggio di matching e quello di mismatching usando questi parametri.

La similarità di due programmi è normalizzata tra 0% e 100%. Il punteggio di similarità è definito dalla funzione seguente:

$$SIM_{abs}(A,B) = \Sigma_{(a,b)\in\, align(A,B)}\, M^P(a,b)$$

in cui *align* è la funzione che allinea i due programmi A e B che calcolano le aree simili. Ci sono anche altri metodi usati per calcolare la similarità normalizzata. Ad esempio la funzione di similarità $SIM_{sum}(A,B)$ definita in questo modo:

$$SIM_{sum}(A,B) = \frac{2\ SIM_{abs}(A,B)}{SIM_{abs}(A,A) + SIM_{abs}(B,B)}$$

In un altro metodo di normalizzazione il punteggio di similarità è calcolato dividendo $SIM_{abs}(A,B)$ per il punteggio di similarità più basso fra $SIM_{abs}(A,A)$ e $SIM_{abs}(B,B)$. Consideriamo che $SIM_{min}(A,B)$ sia normalizzata dal punteggio di similarità più basso. Possiamo calcolare che la similarità

non sia influenzata dalla differenza di dimensione dei due programmi, come segue:

$$SIM_{min}(A,B) = \frac{SIM_{abs}(A,B)}{min\{SIM_{abs}(A,A),\ SIM_{abs}(B,B)\}}$$

Esperimenti

Sono stati condotti vari esperimenti per individuare il plagio in java utilizzando i *bytecode*. Sono stati predisposti tre set di dati di cui due sono stati sottoposti agli studenti di un corso di programmazione in Java e il terzo è un semplice codice usato dal sistema **JPlag**. Prima di tutto abbiamo confrontato la distribuzione della similarità dei *bytecode* con i corrispondenti codici sorgente. Poi abbiamo testato il coefficiente di correlazione tra la distribuzione della similarità dei *bytecode* e quella dei codici sorgente. La tabella 1 mostra le statistiche dei dati sperimentali.

		lines of souce code			
Group	Files	Max	Min	Average	Stdevp
G01	35	146	46	79.05	24.84
G02	32	180	50	87.41	33.23
G03	30	942	199	355.00	131.30

Tabella 1 – Statistiche dei dati sperimentali

I parametri di controllo α e β sono settati a 0,5, il che implica che il punteggio per il matching e le penalità per il mismatching siano uguali per ogni token.

La figura 20 mostra le distribuzioni della similarità. Le linee rosse mostrano la distribuzione dei *bytecode* e quelle blu quella dei codici sorgente.

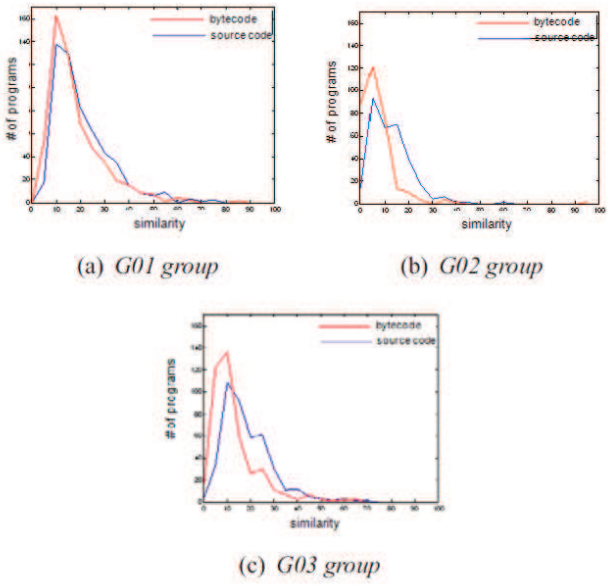

Figura 20 – La distribuzione della similarità dei bytecode e quella dei codici sorgente

La fig. 20 (a) mostra la distribuzione della similarità del gruppo di programmi G01. Per G01 la distribuzione della similarità dei *bytecode* e quella dei codici sorgente è strettamente simile. La fig. 20 (b) e la 20 (c) mostrano la distribuzione dei gruppi G02 e G03 rispettivamente ed evidenziano che generalmente la distribuzione dei *bytecode* è più bassa di quella dei codici sorgente.

Per tutti i gruppi testati, la media della similarità dei *bytecode* è più bassa di quella dei codici sorgente; il gruppo G03 contiene coppie di programmi plagiati per i quali la similarità è pari al 100% e $PINT_B$ riporta anch'esso la stessa similarità del 100%.

Come esperimento finale si è studiato il coefficiente di correlazione tra la distribuzione delle similarità dei *bytecode* e quella delle similarità dei codici sorgente. Se la correlazione è abbastanza alta, allora $PINT_B$ può essere utilizzato per individuare casi di plagio del codice sorgente. La figura 21 mostra il diagramma della distribuzione dei valori di similarità nelle coppie di codici sorgente (asse delle x) e in quelle di bytecode (asse delle y).

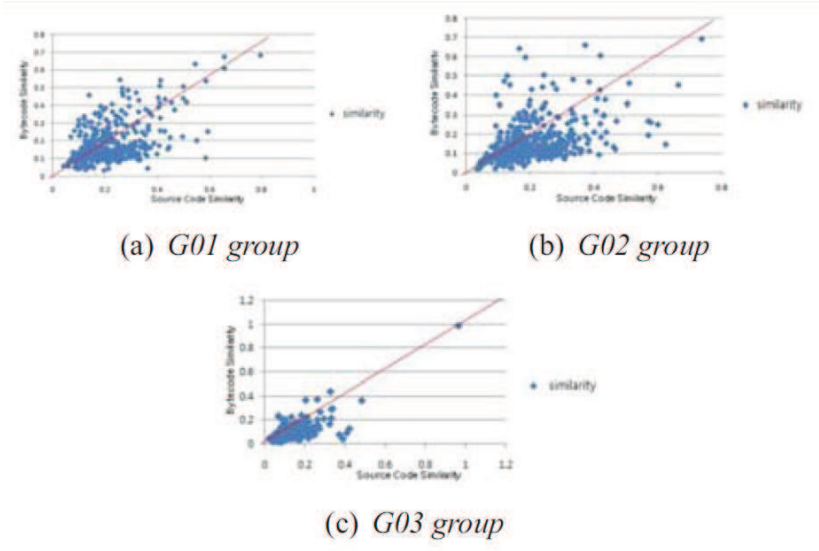

Figura 21 – Diagramma della distribuzione dei valori di similarità nelle coppie di codici sorgente e in quelle di *bytecode*

Per ciascun gruppo i valori di similarità più bassi del 50% mostrano molte differenze. Tuttavia per le coppie con alta similarità il coefficiente di correlazione è abbastanza alto. Le coppie meno simili non sono considerate poiché non c'è sospetto di plagio.

$PINT_B$ mostra di essere meno robusto di fronte al plagio che si manifesta con alterazione di dichiarazioni. In conclusione la rilevazione di plagio tramite *bytecode* può essere realizzata come primo step per poi rilevare il plagio nel codice sorgente. $PINT_B$ può essere molto efficace per individuare il plagio nei codici Java commerciali.

Capitolo 3 – LA TUTELA DEL SOFTWARE

La tutela giuridica del software: le radici storiche

La disciplina delle opere dell'ingegno è indissolubilmente legata all'evoluzione tecnologica. L'invenzione della stampa a caratteri mobili rappresentò una svolta epocale per lo sviluppo di una tutela giuridica della proprietà intellettuale, dal momento che prima dell'invenzione della stampa i costi e i tempi della copia a mano erano tali per cui la diffusione dei libri era molto limitata.

Con lo sviluppo dell'editoria, che applicava procedimenti industriali alla produzione di opere librarie, si venne a creare un nuovo diritto (o più propriamente un fascio di diritti) che poteva essere oggetto di negoziazione contrattuale. Questo diritto, che sorgeva in capo all'autore in via esclusiva, poteva essere ceduto allo stampatore dietro compenso.

Il clima culturale in cui si iniziò a pensare ad una tutela giuridica delle opere creative (all'epoca nel senso di opere letterarie) era quello della rivoluzione industriale, della Rivoluzione francese, dell'illuminismo. Si cominciò a parlare di diritto d'autore come strumento per la tutela del lavoro intellettuale e, per la prima volta nella storia, un riferimento esplicito alla tutela della proprietà intellettuale fu inserito in una carta costituzionale, quella degli Stati Uniti d'America del 1787.

Nel corso dei secoli la tutela del diritto d'autore è stata definita in maniera sempre più precisa nel tentativo di tutelare sia l'interesse pubblico ad accedere alle opere creative che il diritto dell'autore di esclusiva sullo sfruttamento economico della propria opera.

Nonostante geneticamente e ontologicamente il software sia probabilmente più vicino alle invenzioni industriali che non alle opere dell'ingegno negli anni '60 il mondo giuridico, per una serie di ragioni, invero non sempre di carattere strettamente giuridico, ritenne di equiparare il software alle opere

dell'ingegno e, quindi, di proteggerlo dapprima negli Stati Uniti e, quindi, nel resto del mondo, attraverso il *copyright* nei paesi di Common Law e con il diritto d'autore nel vecchio continente.

L'evoluzione del *copyright* è stata profondamente condizionata dalle innovazioni tecnologiche e soprattutto dalla rivoluzione informatica e telematica, che, al pari dell'invenzione della stampa a caratteri mobili, ha avuto un impatto profondo sul mondo giuridico connesso alle opere di ingegno.

Sul finire degli anni '50 le industrie del settore informatico, che sino a quel momento avevano investito ingenti capitali e risorse nello sviluppo dell'hardware, compresero l'importanza e l'utilità del software e, di conseguenza, cominciarono a sviluppare programmi applicativi preconfezionati capaci di risolvere i più diversi problemi degli utenti.

Il software rimase, di fatto, un bene accessorio all'hardware ed era, per lo più, venduto con questo in un unico pacchetto, sino al 1969, anno in cui l'Autorità Antitrust americana impose all'IBM di cessare tale pratica commerciale ritenendola gravemente lesiva della libertà di concorrenza dando così vita al mercato del software.

A quel punto i programmi per elaboratore acquistarono la dignità di bene economico giuridico autonomo e, ovviamente, si cominciarono a delineare molti e delicati problemi di tutela.

Con il passare degli anni, il progresso tecnologico ha consentito di produrre hardware a costi sempre inferiori annullando, di fatto, le differenze qualitative tra le macchine realizzate da produttori diversi. Il software è divenuto il motore di un settore industriale che apporta contributi sempre più significativi all'economia mondiale generando occupazione e gettiti fiscali e aumentando la produttività, la capacità e la competitività dei più diversi settori.

Tenuto conto del consistente valore commerciale del software, degli ingenti investimenti necessari alla sua realizzazione e della sua vulnerabilità nonché del fenomeno, dilagante sin dall'inizio, della pirateria, il mondo industriale comprese subito l'importanza di individuare adeguati strumenti di tutela.

Piano piano l'industria del settore iniziò a preoccuparsi di realizzare sistemi tecnici di tutela, come la predisposizione di bombe logiche (che bloccavano automaticamente le funzionalità del software all'espiare del periodo per il

quale la singola copia era stata concessa in licenza), o il ricorso a diversi metodi di crittografia che sarebbero dovuti servire a limitare il fenomeno della riproduzione abusiva.

Nell'ordinamento esistevano anche alcune tecniche giuridiche che potevano in qualche modo tutelare il software, come la disciplina degli obblighi di fedeltà del prestatore di lavoro nonché quella del segreto industriale e della concorrenza, ma tali tecniche imponevano obblighi di astensione e segretezza solo a determinati soggetti in particolari rapporti con l'impresa che sviluppava il software.

Si è ritenuto allora più opportuno che la tutela giuridica del software potesse essere affidata o alle norme sul diritto d'autore o alle norme in materia di brevetto per invenzioni industriali; le prime furono concepite originariamente per le opere dell'ingegno (nel senso di creazioni di tipo artistico-letterario) e le seconde invece per le invenzioni (nel senso di soluzioni tecniche destinate all'applicazione industriale).

Il software (assieme alle banche dati elettroniche e alle opere di design industriale) è proprio una di quelle tipologie di creazioni intellettuali che pongono problemi in merito a questa classificazione. A creare dubbi è proprio una caratteristica peculiare del software: la sua funzionalità, ovvero la sua vocazione di opera destinata alla soluzione di problemi tecnici; caratteristica questa che lo avvicina ineluttabilmente alla categoria delle invenzioni dotate d'industrialità.

D'altro canto, però, il software appare carente del requisito della materialità considerato da alcuni giuristi come *condicio sine qua non* per la brevettabilità.

Storicamente, inoltre, la tutela brevettuale venne vista con diffidenza dalle aziende produttrici di hardware: esse temevano che tale prospettiva avrebbe attribuito un eccessivo potere alle aziende di software e reso il commercio dell'hardware schiavo delle loro scelte di mercato. E dal canto loro le aziende produttrici di software accettarono di buon grado la proposta di optare per il *copyright* per una ragione banalmente economica: infatti se la tutela di diritto d'autore è automatica e gratuita, quella brevettuale richiede una procedura di formalizzazione e registrazione spesso molto costosa.

Normative nazionali e internazionali

La prima legge sul diritto d'autore (chiamata legge Le Chapelier) risale al 1791; dalla Francia il modello è stato esportato in gran parte dell'Europa continentale e principalmente in Italia e Germania.

Nel 1886 è stata stipulata la «Convenzione di Berna per la protezione delle opere letterarie e artistiche» con la quale sono stati posti alcuni principi essenziali e comuni ai vari ordinamenti e alla quale ha via via aderito gran parte dei paesi del mondo industrializzato.

Tuttavia, se dal punto di vista puramente di principio la Convenzione di Berna ha cercato di realizzare una convivenza tra gli elementi caratterizzanti dei sistemi di *copyright* e quelli di *droit d'auteur*, dal punto di vista dei modelli contrattuali è stato il sistema di *copyright* ad avere la meglio e a diffondersi su scala quasi globale.

In ambito internazionale, il paese che per primo ha risolto la questione della tutela dei programmi per elaboratore sono stati gli USA che, con il «Computer Software Copyright Act» del 12 dicembre 1980, prevedendo la *registrabilità* dei programmi, hanno considerato gli stessi opere *d'ingegno e non invenzioni.*

Un'altra importante convenzione internazionale è il TRIPs, ovvero The Agreement on Trade Related Aspects of Intellectual Property Rights (sancito a Marrakech nel 1994 e promosso dall'Organizzazione Mondiale del Commercio). Come la Convenzione di Berna anche il TRIPs è stato via via sottoscritto da quasi tutti i paesi industrializzati e tutti i sottoscrittori hanno dovuto adeguare le loro legislazioni ai principi ivi sanciti, al di là che si trattasse di ordinamenti di diritto d'autore o di *copyright*.

Esiste anche un'agenzia specializzata delle Nazioni Unite chiamato OMPI Organizzazione Mondiale per la Proprietà Intellettuale (oppure WIPO World Intellectual Property Organization) e preposta a gestire a livello internazionale le questioni inerenti alla proprietà intellettuale in senso lato (comprendente quindi anche l'ambito dei brevetti, dei marchi, del design industriale e delle denominazioni geografiche).

In Italia, fino alla emanazione del D.Lgs. 518/1992, il problema della tutela del software è stato affrontato esclusivamente dalla giurisprudenza, che sin dai primi anni '80 riconosceva ai programmi per elaboratore la tutela d'autore.

La Direttiva CE del 14 maggio 1991, n. 250, recepita e attuata in Italia con il D.Lgs. 29 dicembre 1992, n. 518 proprio partendo dal presupposto che la strada brevettuale non era percorribile e che era necessario assicurare al software un trattamento giuridico uniforme che ne favorisse la circolazione nel mercato globale, sancì la tutelabilità del programma per elaboratore alla stregua delle opere letterarie e artistiche.

Il d.lgs. 518/1992 ha stabilito che i programmi per elaboratore sono protetti «come opere letterarie ai sensi della Convenzione di Berna sulla tutela delle opere letterarie e artistiche».

Tale tutela si estende ai programmi «in qualsiasi forma espressi purché originali» mentre ne restano esclusi «le idee ed i principi che stanno alla base di qualsiasi elemento di un programma compresi quelli alla base delle sue interfacce».

In Italia, in fatto di diritto d'autore, il testo legislativo di riferimento resta tutt'oggi la Legge 633/1941, la cosiddetta Legge sul Diritto d'Autore. Essa ovviamente non è rimasta immutata dal 1941 ma ha subito nel corso degli anni cospicui interventi di riforma e di integrazione, soprattutto dietro la spinta della normativa europea; oltre al già citato Decreto Lgs. n. 518/1992 (Attuazione della direttiva 91/250/CEE relativa alla tutela giuridica per i programmi per elaboratore), è il caso di ricordare anche: Decreto Lgs. n. 154/1997 (*Attuazione della Direttiva 93/98/CEE concernente l'armonizzazione della durata di protezione del diritto d'autore e di alcuni diritti connessi*), Decreto Lgs. n. 169/1999 (*Attuazione della Direttiva 96/9/CEE relativa alla tutela giuridica delle banche dati*), Legge 248/2000 (*Nuove norme di tutela del diritto d'autore*), Decreto Lgs. n. 95/2001 (*Attuazione della direttiva 98/71/CE relativa alla protezione giuridica dei disegni e dei modelli*), Decreto Lgs. n. 68/2003 (*Attuazione della direttiva 2001/29/CE sull'armonizzazione di taluni aspetti del diritto d'autore e dei diritti connessi nella società dell'informazione*), Decreto Lgs. n. 118/2006 (*Attuazione della direttiva 2001/84/CE relativa al diritto dell'autore di un'opera d'arte sulle successive vendite dell'originale),* Decreto Lgs. n. 140/2006 (*Attuazione della direttiva 2004/48/CE sul rispetto dei diritti di proprietà intellettuale).*

Il software e la proprietà intellettuale

Il software è tutelato dal diritto d'autore come opera letteraria, considerando che è costituito da una serie di istruzioni redatte in un linguaggio di programmazione, ma in realtà tale impostazione vale più per il codice sorgente che non per il codice oggetto, dal momento che quest'ultimo non ha alcuna finalità di comunicazione e non è rivolto all'uomo ma solo alla macchina.

L'opera dell'ingegno per essere protetta ai sensi della legge sul diritto d'autore deve avere «carattere creativo» o, meglio deve costituire il risultato di un'attività intellettuale dell'autore; ai fini dell'applicazione del diritto d'autore al software, si considera meritevole di tutela ogni programma per elaboratore che risulti frutto di un'attività intellettuale dell'autore e non di una mera azione di copiatura.

Rispetto alle altre opere dell'ingegno un SW è caratterizzato dalle seguenti proprietà:

- *Scarsa materialità*. Letteralmente il termine SW indica le componenti di un computer che sono *soft* (cioè *leggere*) cioè i programmi, in contrapposizione a quelle *hard* (cioè *pesanti*), che sono le componenti fisiche, ben visibili e tangibili, come il processore, la memoria (RAM), i dischi, le schede di rete, etc.
- *Natura intrinsecamente tecnica*. In tale contesto, la finalità dell'opera è la soluzione di problemi. Un programma, quando viene attivato, esegue delle azioni (istruzioni specificate nel programma) e produce dei risultati. Nel gergo informatico viene detto che un programma implementa un algoritmo, cioè esegue una sequenza di passi che compiono una determinata azione, compito o funzione. L'aspetto interessante è che, come nella matematica, un certo risultato può essere ottenuto con procedimenti diversi, cioè con algoritmi diversi. Questo vuol dire che un certo problema può essere risolto con più metodi o algoritmi ottenendo lo stesso risultato. Di conseguenza programmi diversi (in termini di algoritmi che implementano) possono risolvere lo stesso problema. Questa caratteristica è molto importante nella discussione in atto dato che gli algoritmi astratti non sono brevettabili.

- *Assume forme diverse*. Per poter essere eseguito su un computer la forma del SW cambia. Il programma di solito viene scritto in un linguaggio di programmazione detto di *alto livello* cioè facilmente comprensibile dagli esseri umani (detto sorgente) e poi viene tradotto in un formato intermedio (codice oggetto, *bytecode*, ...) e/o in forma direttamente interpretabile dal computer (eseguibile).
- *Dualità del SW*. Questa caratteristica è veramente importante dal punto di vista normativo. Difatti un programma di per sé è un file di testo e perciò da un lato sarebbe assimilabile ad un'opera letteraria soggetta dunque alla normativa sul diritto d'autore, ma d'altro canto una volta in esecuzione il SW cambia completamente natura: esegue dei passi e produce dei risultati e difatti nel gergo informatico assume persino un nome diverso (viene chiamato processo). Da questo punto di vista un programma diventa quindi, per certi versi, assimilabile ad un processo industriale.

Nel software la forma espressiva, intesa come struttura e sviluppo delle singole istruzioni che compongono un programma, è sempre sostanzialmente vincolata dalla necessità di raggiungere un determinato risultato. La natura tecnica dei programmi per elaboratore porta con sé un'altra importante conseguenza: l'assoluta impossibilità di valutare la somiglianza di due software alla stregua dei criteri applicabili a tutte le altre opere dell'ingegno. Il software, infatti, pur essendo un'opera dell'ingegno umano «scritta» in un determinato linguaggio non è destinato ad esser letto, eseguito o «fruito» dall'uomo ma, esclusivamente, dalla macchina.

Negli ultimi anni, l'orientamento di tutelare il software ai sensi della legge sul diritto d'autore è via via mutato, soprattutto per opera della giurisprudenza che ha ritenuto di interpretare il significato dell'espressione «programmi per elaboratore in quanto tali» in modo da lasciare aperto uno spiraglio alla brevettabilità del software almeno ogni qualvolta esso – in combinazione con una macchina – consenta di raggiungere un risultato tecnico nuovo e originale ai sensi della vigente normativa in materia di brevetto per invenzione industriale.

Negli Stati Uniti e in Giappone, ormai da diversi anni, i programmi per elaboratore vengono protetti mediante il brevetto industriale, mentre in Europa si continua a fare ricorso ad un artificio giuridico per il quale «i

programmi in quanto tali» non sono brevettabili mentre è brevettabile l'invenzione tecnica connessa ad un programma.

L'attuale assetto normativo, riconoscendo al software una tutela parabrevettuale sotto le mentite spoglie del diritto d'autore in assenza delle garanzie e dei requisiti previsti dalla legge invenzioni, tutela in maniera troppo ampia e poco puntuale.

Diritto d'autore e brevetto

Si tratta di due strumenti di tutela ben distinti, con differenti caratteristiche e con diversi campi di applicazione: il diritto d'autore attiene alla sfera delle opere dell'ingegno, mentre il brevetto attiene alle invenzioni industriali. Per meglio cogliere le differenze essenziali tra il concetto di brevetto e quello di diritto d'autore, è possibile ricorrere al seguente schema.

	DIRITTO D'AUTORE / COPYRIGHT	BREVETTO
oggetto della tutela	opere dell'ingegno di carattere creativo	invenzioni, modelli di utilità, varietà vegetali
acquisizione del diritto	automaticamente con la creazione dell'opera	attraverso una procedura di registrazione presso appositi uffici
requisiti per la tutelabilità	carattere creativo (originalità + novità)	attività inventiva, novità, liceità, applicazione industriale
titolare originario dei diritti	autore	soggetto che effettua la registrazione
durata della tutela	70 anni dalla morte dell'autore	20 anni dalla registrazione
costi	nessuno	costi di redazione e registrazione

Se si pensa alle peculiarità tecniche del software (strutturato in una serie di comandi in linguaggio informatico), la sua assimilazione ad un'opera letteraria non pare nemmeno molto forzata; possiamo infatti equiparare il programma in forma di codice sorgente ad un manuale di istruzioni tecniche redatte in un preciso linguaggio e destinate alla macchina (o ad altri sviluppatori che conoscono quel linguaggio).

La rilevanza dei requisiti di creatività e di originalità (tipici del diritto d'autore) tende quindi a prevalere sulla peculiarità della vocazione funzionale del software; infatti, la soluzione tecnica cui un programma è

preposto può essere raggiunta dal programmatore in diversi modi a seconda del linguaggio prescelto e di come le istruzioni sono disposte all'interno del codice.

Le forme di controllo dell'informazione digitale

L'era digitale ha reso possibile rappresentare testi, suoni, immagini in file di codice binario, elaborare questi file, trasmettere i file in tutto il pianeta in pochi secondi.

Con l'utilizzo delle tecnologie informatiche l'attività creativa coinvolge sempre più un gruppo di soggetti, si pensi ad esempio alle comunità che sviluppano software *open source*, oppure si pensi a un'opera multimediale il cui sviluppo coinvolge contemporaneamente più soggetti (il produttore, il grafico, l'informatico,ecc.). In alcuni casi le nuove tecnologie ridefiniscono i rapporti fra autore e fruitori, ad esempio negli ipertesti il lettore diventa protagonista insieme all'autore poiché può scegliere come proseguire la lettura.

È più semplice condividere le informazioni, ma anche più semplice copiarle. Secondo Lessig, famoso giurista statunitense, le norme sul *copyright* andrebbero riviste perché tendono a criminalizzare attività che discendono naturalmente dalle tecnologie attualmente in uso e che risultano assolutamente naturali per un'intera generazione che è cresciuta con le potenzialità delle tecnologie digitali.

Si pensi ad esempio al *sampling*, che si ottiene quando si genera (tramite computer) il codice digitale di un determinato brano musicale al fine di incorporare quel codice in una nuova composizione musicale ovvero al fine di combinare o manipolare quel codice binario (o più codici binari) per creare una nuova composizione musicale.

Il diritto d'autore dovrebbe cercare di attribuire nuovi contenuti alla nozione di creatività.

Il plagio costituisce l'altra faccia della creatività ed è stato modificato e incrementato con l'avvento delle tecnologie digitali. Risulta difficile dare una definizione esatta del concetto di plagio sebbene sia spesso citato nell'ambito del diritto d'autore. Da un lato la liquidità del codice binario (semplice da modificare e condividere) consente lo sviluppo di nuove forme di creatività (come il *sampling* di cui sopra), dall'altro la potenza di ricerca

insita nelle reti di computer consente di intercettare facilmente le forme più palesi di plagio (come i software in uso presso molte università per l'individuazione automatica di plagio nei lavori scritti degli studenti).

Nella società dell'informazione si stanno diffondendo modelli innovativi di produzione della conoscenza. Si pensi innanzitutto al sistema operativo Linux, il cui codice sorgente iniziale, creato dal programmatore finlandese Linus Torvalds, fu condiviso in rete affinché qualsiasi informatico nel mondo potesse modificarlo e migliorarlo.

La disponibilità di opere via web in forma digitalizzata affranca le medesime dai supporti informatici. Si pensi ad esempio ai database contenenti intere annate di quotidiani e riviste, all'interno dei quali ogni singolo articolo può essere consultato singolarmente. L'opera non rileva tanto come prodotto quanto come flusso di bit che può essere fruito.

Il *copyright* tradizionale ha subito numerosi adattamenti alle sfide tecnologiche che si sono succedute dal 1800 fino al secondo dopoguerra.

Con l'avvento di una nuova rivoluzionaria tecnologia, quella digitale, il meccanismo ha mostrato tutte le sue debolezze ed è entrato (forse definitivamente) in crisi.

Vediamo quali sono i tratti rivoluzionari delle tecnologie dell'informazione e della telecomunicazione.

1) È possibile superare il concetto di copia inteso come copia del supporto materiale. L'effetto finora più evidente di questo carattere delle tecnologie digitali sta nella possibilità di effettuare e distribuire su scala globale copie dematerializzate, che altro non sono che sequenze di bit.

2) Si è in grado di veicolare l'informazione in una lingua unica compresa dal computer (il codice binario) ed in un formato aperto (c.d. codice sorgente aperto), cioè modificabile dall'uomo (l'esperto informatico) che conosce i linguaggi di programmazione.

3) D'altra parte si ha il potere di chiudere totalmente l'informazione (ad esempio, si può tenere segreto il codice sorgente di un software) rendendola comprensibile solo alle macchine.

Possiamo affermare che le tecnologie digitali inducono un mutamento profondo dell'economia della creatività e si delineano in particolare due forme di produzione dell'informazione:

- il primo modello si basa su forme di produzione *gerarchiche* dove i titolari dell'informazione possono prevedere chi, dove, come e quando potrà fruire della stessa informazione. Tale modello può essere utile nell'attuazione di strategie commerciali, come la scelta di vendere a prezzo elevato mille ascolti di una canzone e a prezzo contenuto dieci ascolti della stessa canzone;
- il secondo modello genera forme di produzione *peer-to-peer* quindi non gerarchiche in cui gli attori coinvolti svolgono sia il ruolo di produttori che quello di consumatori, come nel caso di sviluppo di software a codice aperto o nella scrittura collaborativa di testi come le voci di un'enciclopedia *on-line*.

Ad ogni forma di produzione corrispondono differenti forme di controllo: una basata sulla chiusura (controllo rigido e accentrato) e l'altra basata sull'apertura (controllo flessibile e decentrato) dell'informazione.

Nella prima forma, il controllo si basa sulla chiusura dell'informazione e si presenta rigido ed accentrato. Tale tipologia di controllo prende avvio dal mercato del software c.d. proprietario e si fonda su una prima rudimentale misura tecnologica di protezione: la secretazione del codice sorgente. Sulla prassi della secretazione del codice sorgente si innestano il riconoscimento della protezione da *copyright* e la diffusione di EULAs finalizzati a rafforzare il controllo sul piano contrattuale.

Nella seconda forma il controllo si basa sull'apertura dell'informazione e si presenta flessibile e decentrato. Il primo modello compiuto di questa forma di controllo è rappresentato dalla GNU General Public License. L'informatica ha mosso alcuni dei suoi più significativi passi fuori dalla logica della secretazione del codice sorgente.

Si assiste alla nascita di nuove forme di controllo delle informazioni contenute nelle opere dell'ingegno. Come nel passato si utilizzano principalmente quattro strumenti:

1. il contratto;
2. le norme sociali;
3. la tecnologia;
4. la legge sulla proprietà intellettuale

La rivoluzione digitale pone l'accento sul contratto e sulla tecnologia, mentre la legge diventa uno strumento marginale, di supporto agli altri.

Lo strumento contrattuale della licenza d'uso è utilizzato sia da chi vuole garantirsi un controllo rigido e accentrato sull'informazione, sia da chi è invece favorevole a un controllo flessibile e decentrato.

Le licenze d'uso

Dal momento dell'ingresso del software nella schiera dei beni di consumo, gli autori iniziarono a stilare dei contratti di portata generale in cui esprimevano i termini della distribuzione e della riproduzione del software su cui essi vantavano i suddetti diritti esclusivi: nacque così il tipo contrattuale della *licenza d'uso* di software.

Normalmente queste licenze non consentono, di fatto, la libera riproduzione del programma, né la sua modifica inoltre, la licenza software esime il produttore o distributore da responsabilità per danni provocati dal software.

Quindi il software ha un proprietario (salvo il caso del software di dominio pubblico), che è tale in quanto «detiene i diritti di autore». L'utilizzo del software può essere concesso gratuitamente o a pagamento, per le operazioni stabilite nel contratto di licenza, o in sua mancanza per quanto stabilito dalla legge. Anche se la legge sul diritto di autore stabilisce già quali sono i diritti di chi produce l'opera e di chi la utilizza, per il software si è introdotto nella pratica un contratto, spesso non firmato, il contratto di licenza software, il cui scopo, è quello di regolamentare i rapporti fra il proprietario e l'utilizzatore del software nonché di limitare ulteriormente i diritti di quest'ultimo.

Da un punto di vista terminologico, c'è da notare che la scelta del termine licenza per il tipo di contratto sopra descritto è dovuta probabilmente alla funzione del permesso che viene acconsentito nel contratto, sebbene molti giuristi fanno notare che non sarebbe pienamente appropriata. In realtà la definizione autentica di "licenza" implicherebbe un contratto di concessione da parte di un soggetto (licenziante) ad un altro soggetto (licenziatario) non solo della facoltà di godere di una certa idea creativa, ma anche di sfruttarla economicamente. Ma nella maggior parte dei casi l'oggetto del contratto si limita al solo godimento personale del bene software, non estendendosi ai diritti di sfruttamento economico che restano invece in capo all'autore originario.

In considerazione di tale rapporto giuridico un gran numero di giuristi qualifica la relazione fra autore e utente del software come una locazione,

giacché con il tipo contrattuale che si usa chiamare impropriamente licenza d'uso non si cede alcun diritto di sfruttamento economico sul bene, ma ci si limita a concedere il solo godimento personale del programma.

End User License Agreement (EULA)

La licenza d'uso proprietaria mira a conferire al produttore del software il maggiore controllo possibile sul proprio bene.

Il software è rappresentato attraverso due codici: quello «sorgente», espresso in un linguaggio informatico di programmazione, e quello «oggetto», che è interpretabile solo dal computer. La misura più semplice di tutela delle proprie idee è rappresentata dalla secretazione del codice sorgente, con conseguente commercializzazione del software nella sola forma del codice oggetto.

L'EULA è il principale strumento di distribuzione del software; colui che acquista il software si limita ad accettare le condizioni generali di contratto predisposte dal produttore.

Tale software non è modificabile, infatti normalmente viene distribuito nella sola forma eseguibile.

General public license

Tale tipologia contrattuale si muove nella direzione opposta alla secretazione del codice sorgente. Lo scienziato informatico Stallman aveva contribuito allo sviluppo di una serie di programmi che emulavano le funzionalità del sistema operativo UNIX. Questi programmi furono etichettati con la formula GNU (che sta per «GNU is not UNIX»). Ma l'idea rivoluzionaria di Stallman fu quella di far leva sul *copyright* per garantire, a chi avesse voluto, la libertà di copiare – da qui il termine *copyleft* – , distribuire e sviluppare software a codice sorgente aperto. Stallman creò un particolare tipo di licenza non proprietaria, standardizzata e pubblica (cioè messa a disposizione di chiunque ne voglia fare uso) denominato GNU *GeneraI public license* (GPL).

Il concetto di *copyleft* (gioco di parole che si può tradurre come «permesso d'autore») costituisce un contributo originale ed innovativo alla disciplina del diritto d'autore, proveniente dall'ideologia del Software Libero. Il *copyleft* può essere usato da chi intende avvalersi della tutela del diritto d'autore per difendere la libertà della sua opera, imponendo che questa e

tutte le sue derivazioni restino libere. In pratica, una licenza appartenente alla categoria *copyleft* impedisce a chi ridistribuisce il software (originale o modificato che sia) di aggiungere delle restrizioni rispetto alla licenza ricevuta. La GNU-GPL è il classico esempio di licenza di questo tipo.

La GNU GPL nasce con lo scopo di proteggere alcune libertà fondamentali che sono così espresse sul sito ufficiale di riferimento;

- libertà di eseguire il programma (libertà 0);
- libertà di studiare come funziona il programma (libertà 1). L'accesso al codice sorgente ne è un prerequisito;
- libertà di ridistribuire copie in modo da aiutare il prossimo (libertà 2);
- libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti apportati, a beneficio dell'intera comunità (libertà 3). L'accesso al codice sorgente ne è un prerequisito.

Le licenze Creative Commons

La logica del *copyleft* si sta diffondendo anche alla produzione e distribuzione di contenuti digitali diversi dal software. CreativeCommons è una *nonprofit corporation* fondata nel 2001 negli Stati Uniti.

Uno degli ideatori del progetto Creative Commons è il giurista statunitense Lawrence Lessig, il quale, ispirandosi all'idea di Richard Stallman, ha trapiantato il modello della GNU GPL, sperimentato con successo per il *software*, nel campo più esteso dei contenuti digitali e delle opere dell'ingegno veicolate sui supporti tradizionali come i libri cartacei.

Con lo scopo di favorire la condivisione e la rielaborazione di opere dell'ingegno (letterarie, musicali, filmiche, ecc.) nel rispetto del *copyright,* Creative Commons mette gratuitamente a disposizione del pubblico una serie di licenze e altri strumenti giuridici per qualsiasi autore di un'opera che voglia, per scopi commerciali o non commerciali, consentire ai fruitori di copiare, ridistribuire e modificare l'opera stessa.

Le licenze Creative Commons sono espresse in tre forme differenti: 1) il *legal code,*versione che detta i termini della licenza nel linguaggio tecnico-giuridico; 2) il *commons deed,* che riassume in un linguaggio semplificato i contenuti del documento negoziale; 3) il *digital code,*che identifica la licenza mediante metadati gestibili attraverso sistemi informatici.

I tratti caratterizzanti delle licenze c.d. non proprietarie come la GNU GPL e le CC Licenses possono essere così sintetizzati.

La regola tecnologica rimane sullo sfondo (apertura del codice sorgente del *software* o apertura del contenuto) e soprattutto la tutela non è affidata (almeno negli archetipi delle licenze non proprietarie) alla tecnologia. La prevalenza è data invece ad un testo contrattuale standardizzato (pur sempre basato sulla legge del *copyright*). La scarsa litigiosità finora riscontrata nell'uso delle licenze non proprietarie può far ritenere che sia data all'opera anche una consuetudine la quale riconosce il carattere vincolante dei testi delle licenze di là dalla prospettiva della tutela giudiziale. La cosa non sorprende.

Le licenze non proprietarie formalizzano in testi contrattuali prassi che assomigliano alle norme sociali sperimentate da secoli dalle comunità scientifiche al fine di esercitare un controllo elastico sull'informazione prodotta dalla ricerca.

Le licenze di software libero o open source

Una licenza *open source* garantisce all'utente non solo la possibilità di utilizzare il relativo software senza restrizioni (ad esempio sul numero di copie, o sul tipo di uso), ma anche la facoltà di modificarlo, migliorarlo, adattarlo alle proprie esigenze, e ridistribuirlo modificato. Una condizione è che l'utente renda pubbliche le modifiche apportate, ad esempio attraverso Internet.

È bene ricordare che il software libero non è necessariamente gratuito: la libertà non si riferisce al prezzo, ma all'uso che se ne può fare.

Le licenza di software libero non protetto da copyleft

Non sempre il software libero è necessariamente di tipo *copyleft* e ciò accade quando la licenza non vieta espressamente l'aggiunta di restrizioni da parte di chi lo ridistribuisce. In generale sarebbe opportuno, se si utilizza del software distribuito con questo tipo di licenza, accertarsi con cura dei termini del contratto, in particolare per quanto riguarda la specifica copia della quale si è venuti in possesso.

Le licenza di software semi-libero

Questi tipi di licenze permettono di usare il software in oggetto, di copiarlo, modificarlo e distribuirlo anche modificato, per qualunque scopo, escluso

quello di trarne profitto. In altri termini, si tratta di licenze per software libero a cui è stato aggiunto un vincolo che ne impedisce l'uso e la distribuzione a scopo di lucro.

Le licenze di software freeware

Il termine freeware non è abbinato a una definizione precisa, ma viene inteso generalmente come software gratuito, che può essere usato e copiato liberamente, ma che non può essere modificato. Infatti tipicamente il codice sorgente non viene reso pubblico. In questo senso, il prefisso free serve solo a evidenziare la gratuità della cosa, ma non la libertà che invece richiede altri requisiti.

Le licenze di software shareware

Con il termine shareware si fa riferimento a software proprietario che può essere ridistribuito, ma per il quale viene richiesto espressamente il pagamento dopo un periodo di prova.

Il software «Public Domain»

Il software di dominio pubblico (o Public Domain) è il software non protetto da *copyright*. Questo tipo di software è completamente libero, nel senso che chiunque può usarlo, riprodurlo, modificarlo, e ridistribuirlo. Tuttavia, non essendo soggetto ad alcuna forma di tutela, chiunque può appropriarsene ridistribuendolo come proprio, limitandone in questo modo la libertà.

Le licenze di software commerciale

In base alle classificazioni viste in questo capitolo, il software commerciale è tale solo poiché è venduto per profitto. Lo sviluppo e la diffusione del Software Libero dipendono anche dalla possibilità di venderne delle copie, originali o modificate, per trarne profitto. Pertanto, il software che pur offrendo le quattro libertà fondamentali di cui al capitolo successivo, non consente la commercializzazione per trarne profitto, non viene considerato libero in modo completo. In questo senso, è importante evitare di confondere il software proprietario con il software commerciale, perché non sono la stessa cosa.

Si osservi che può esistere anche del software non-libero, che non è nemmeno commerciale.

Le norme sociali

Da sempre gli uomini regolano i propri comportamenti anche basandosi su norme informali, che svolgono un ruolo importante anche nel controllo dell'informazione digitale. Ad esempio la licenza GNU GPL (un contratto standard) intende costituire una formalizzazione della norma del comunismo (o della condivisione) nell'ambito dello sviluppo del software. Analogamente, le licenze Creative Commons (contratti standard) vengono utilizzate dal movimento dell'*open access* per formalizzare la norma del comunismo al fine di garantire la libera condivisione di articoli e libri scientifici in versione digitale.

La tecnologia come forma di controllo dell'informazione digitale: MTP e DRM

La vera rivoluzione del controllo basato sulla tecnologia avviene con l'informatica. La crittografia digitale consente di costruire software per il controllo dell'accesso e dell'uso di qualsiasi opera espressa in codice binario. La legislazione definisce tecnologie di questo tipo *misure tecnologiche di protezione* (MTP) e la loro applicazione è disciplinata dalla Legge sul Diritto d'Autore.

Il *digital rights management* (DRM) rappresenta una versione più moderna e sofisticata delle MTP. Le principali componenti dei sistemi di DRM sono:

1. le MTP basate principalmente sulla crittografia digitale, ma anche su altre tecnologie come il *watermarking* (identificazione tramite «marchio») e il *fingerprinting* (identificazione tramite «impronte») digitali;
2. i metadati che accompagnano il contenuto descrivendolo in un linguaggio comprensibile al computer, ovvero il titolare del contenuto, l'utente e le regole per l'utilizzo del contenuto.

Mediante il DRM, la clausola dell'EULA che vieta l'installazione del software su più di tre computer si traduce in un meccanismo che impedisce automaticamente all'utente la quarta installazione: al tentativo di procedere alla quarta installazione il computer reagirà disattivando del tutto la procedura.

La chiave di volta di questi sistemi è rappresentata dai Rights Expression Languages (RELs), tecnologie che si propongono di esprimere le regole in

un linguaggio comprensibile alle macchine, cioè agli apparecchi (computer, lettori MP3, telefoni cellulari, televisori, consolle per videogiochi, etc.). Ma da soli i RELs non sono sufficienti alla gestione dei permessi e dei divieti di accesso ed utilizzo delle informazioni. Vi è appunto bisogno di tecnologie basate sulla crittografia digitale.

In sintesi i sistemi di DRM:

a) sono finalizzati alla gestione di regole contrattuali e di contratti;
b) implicano sempre componenti crittografiche volte ad identificare informazioni dunque chiamano in causa la disciplina giuridica della protezione dei dati personali;
c) possono incorporare misure di autotutela tecnologica;
d) sono stati finora utilizzati prevalentemente per la tutela delle opere dell'ingegno, dunque chiamano in causa il diritto d'autore;
e) spesso si basano su alcune componenti segrete che sono potenzialmente suscettibili di essere protette da brevetti;
f) fanno leva su componenti standardizzate, dunque chiamano in causa il diritto della concorrenza.

Facendo leva sulle leggi in materia di proprietà intellettuale, sui contratti, sulle consuetudini e sugli standard tecnologici è possibile ottenere differenti forme di controllo delle informazioni digitali.

Nel DRM il controllo si estende da una forma espressiva del software (il codice sorgente) ad ogni informazione rappresentabile in codice binario (non solo software, ma file di testo, audio, video, etc.). Si presti attenzione al fatto che mentre la secretazione del codice sorgente è una forma di controllo relativa, in quanto è teoricamente possibile un procedimento di ingegneria inversa che porti dal codice oggetto ad un codice sorgente simile a quello segreto, nella criptazione digitale il controllo è – nel caso in cui l'algoritmo crittografico sia sicuro – assoluto ed esercitabile a distanza. Tuttavia, l'evoluzione non sta solo nel potenziamento del controllo dell'informazione, ma anche nella traduzione degli EULAs in un linguaggio comprensibile alle macchine. L'obiettivo del DRM è infatti che i termini della licenza per l'accesso e l'uso dell'informazione siano riconoscibili dai software e dagli apparecchi costruiti (in base agli standard del sistema di DRM) per la fruizione della medesima informazione.

Il DRM genera un controllo esclusivo dell'informazione digitale (anche un singolo dato, come una parola o una nota, non dotato di alcuna originalità). A differenza del controllo contrattuale dell'informazione, il controllo esclusivo basato sul DRM si rivolge ad una serie indeterminata di soggetti (assume di fatto una natura «reale», comportandosi come una sorta di «proprietà dell'informazione»). Chiunque vorrà fruire dell'informazione sarà (di fatto) soggetto alle regole incorporate e alla tutela (fondata su ciò che l'analisi economica del diritto definisce una «property rule», cioè su una tutela inibitoria) nella tecnologia. La tutela del diritto d'autore fa leva sulla materialità dell'attività che integra la violazione del diritto di esclusiva. La tutela del DRM fa leva sull'inalterabilità dell'architettura informatica (ad es., inviolabilità degli algoritmi crittografici, immodificabilità dell'*hardware*, *etc.*) e dunque in ultima analisi sulla conoscenza (un'entità immateriale). Le opere dell'ingegno sono espresse tradizionalmente in linguaggi aperti che consentono l'accesso e la conservazione (quanto meno quella parziale affidata alla memoria umana) dell'informazione. Se il DRM si basa su standard tecnologici espressi in formati chiusi (cioè segreti), un'eventuale obsolescenza dei formati rende di fatto inaccessibile l'informazione.

La legge in relazione con le altre forme di controllo dell'informazione

È evidente il tentativo di riservare al titolare dei diritti d'autore il potere di controllo su forme di copiatura come quelle temporanee e parziali che sono tipiche della dimensione digitale. Il punto è che le tradizionali tecniche di tutela del diritto di riproduzione nei fatti non sono in grado di assicurare l'efficacia del nuovo potere di controllo.

Si è preferito introdurre un regime in base al quale ad autori ed editori compete un diritto di credito – indicato dalla nostra legge come diritto ad un «equo compenso» – verso i produttori ed importatori di supporti di registrazione audio e video e di apparecchi di registrazione amministrato da società di gestione collettiva dei diritti (come la SIAE). In sostanza una percentuale del prezzo di acquisto di apparecchi e supporti per la copia privata viene attribuita alla comunità degli autori e degli editori.

Nell'ambito dell'ondata legislativa internazionale e nazionale volta a rafforzare la tutela della proprietà intellettuale e del diritto d'autore, si

segnalano alcune tendenze normative che spingono a un maggiore coinvolgimento degli *Internet service providers,* in particolare di quei *providers* che forniscono servizi di accesso e connettività alla rete.

La prima rilevante forma di tutela giuridica delle misure tecnologiche di protezione (MTP) si deve ai *World Intellectual Property Organization* (WIPO) *Treaties* del 1996. I legislatori statunitense ed europeo hanno dato attuazione al mandato internazionale emanando rispettivamente il *Digital Millennium Copyright Act* (DMCA) del 1998 e la direttiva 2001/29/CE. Semplificando, il nucleo comune delle norme sta nel triplice divieto:

a) di elusione delle MTP delle opere;
b) di produzione o diffusione di tecnologie «principalmente finalizzate» all'elusione delle MTP delle opere;
c) di rimozione o alterazione delle informazioni sul regime dei diritti.

Si tratta di normative assai complesse – per non dire confuse – e assistite da severe sanzioni penali.

Free software e open source: evoluzione e compatibilità con il diritto d'autore

Alla fine degli anni sessanta, una ristretta comunità di esperti programmatori, che fino ad allora avevano lavorato esclusivamente nei centri di ricerca e nelle università degli Stati Uniti, portarono all'attenzione mondiale il sistema operativo UNIX. Ken Thompson, il suo creatore, scrisse questo sistema operativo in un linguaggio di programmazione chiamato C che permise di sviluppare l'idea di portabilità e compatibilità del software: grazie a questi sviluppi, il ruolo del software si fece più dinamico e più facilmente gestibile. Negli stessi anni furono collegate in rete quattro grandi università degli Stati Uniti (Los Angeles, Santa Barbara, Stanford e Utah) dando origine al primo embrione della comunicazione Internet in ambito scientifico. Le conseguenze di questo continuo perfezionamento tecnologico portarono ad un espandersi a macchia d'olio dello sviluppo economico in questo settore. Nei primi anni ottanta, con l'arrivo del personal computer l'utenza divenne sempre più numerosa e quindi non più corrispondente ad una ristretta nicchia di operatori.

Di pari passo lo sviluppo del software, ormai in larga misura slegato dallo sviluppo dell'hardware, rese necessaria, come s'è visto, la formalizzazione degli strumenti per la sua tutela con l'estensione al software delle leggi sul

diritto d'autore che, in questa accezione, è inquadrabile tra i diritti di terza generazione, vale a dire, i diritti tecnologici. Tuttavia il software si distingue dalle altre opere d'ingegno per la sua particolare natura. Infatti, esso è composto da due parti fondamentali: il codice sorgente e il codice eseguibile: il primo è intelligibile, il secondo è adatto all'esecuzione su computer e non è intelligibile. Dal momento che per funzionare è sufficiente il codice eseguibile, le leggi di vari paesi che tutelano il diritto di autore per il software spesso permettono la distribuzione del solo codice eseguibile, lasciando all'autore il diritto di mantenere nascosto il codice sorgente. Inoltre le leggi di molti paesi considerano illecita la decodificazione, in altre parole lo studio del codice eseguibile volto a scoprirne il funzionamento.

Le forme di tutela del software basate sul diritto d'autore e su licenze tradizionali erano invise a diversi operatori del settore, in particolare a programmatori esperti che ritenevano frustrante dover utilizzare programmi prodotti da altri senza poterne conoscere il funzionamento e senza poterli modificare.

Nei primi anni ottanta si colloca l'attività di Richard M. Stallman, considerato il padre del Software Libero. In quegli anni Stallman, in aperta contrapposizione con le comuni prassi dei produttori di software, aveva iniziato a distribuire gratuitamente e liberamente i propri programmi, incoraggiando chiunque a modificarli e migliorarli.

La Free Software Foundation

Nel 1984 Richard M. Stallman creò la Free Software Foundation (Fondazione del Software Libero) per dare supporto logistico, legale ed economico al progetto GNU.

Il software sviluppato nell'ambito del progetto GNU, nonostante fosse ampio e funzionante, non era sufficientemente completo, poiché non aveva disponibile un vero e proprio kernel, vale a dire, il nucleo del sistema operativo che è un insieme di programmi e dati che permettono al computer di eseguire correttamente tutte le applicazioni che compongono l'intero sistema operativo. Solo nel 1992, il giovane studente dell'Università di Helsinki Linus Torvalds sviluppò un kernel compatibile con UNIX utilizzando gli strumenti software forniti dalla Free Software Foundation. Tale combinazione diede origine al sistema operativo chiamato LINUX.

Il concetto di «software libero» discende dalla cultura di libertà di scambio d'idee e d'informazioni. Negli ambienti scientifici, quest'ultimo principio è tenuto in alta considerazione per la fecondità che ha dimostrato; ad esso, infatti, è generalmente attribuita gran parte dell'eccezionale ed inimmaginabile crescita del sapere negli ultimi decenni.

Per la Free Software Foundation, la libertà di scambio di nozioni non è quindi una questione puramente pratica: essa è alla base dei concetti di libertà di pensiero e di espressione. Analogamente alle idee, il software è immateriale, e può essere riprodotto e trasmesso facilmente. In modo simile a quanto avviene per le idee, parte essenziale del processo che sostiene la crescita e l'evoluzione del software è la sua libera diffusione. Ai nostri giorni, come le idee, il software permea il tessuto sociale e lo influenza, produce effetti etici, economici, politici e, in un senso più generale, culturali.

È importante tenere presente che lavorare nell'ambito del software libero non significa fare solo del volontariato o rifiutare anticipatamente ogni forma di commercializzazione. Il primo a confermarlo è proprio Stallman. Naturalmente, il profitto derivato dalla vendita delle licenze proprietarie è nettamente superiore a quello derivabile dal software libero, tuttavia, i servizi collegati a quest'ultima forma di distribuzione prospettavano dei guadagni che, sebbene meno consistenti delle licenze di pacchetti chiusi, erano più elastici, duraturi e coadiuvati da investimenti ragionevoli sulle spese di produzione e distribuzione. L'effetto di questo fenomeno si estese velocemente grazie all'utilizzo di Internet. Il risultato è stato che il Software Libero, come modello di business, si è presentato al mondo degli affari non solo come un potenziale nemico per i detentori delle licenze proprietarie, ma anche come un'allettante valvola di sfogo per un nuovo orizzonte di sviluppo.

La figura di Stallman è importante anche per essere stato il primo in assoluto a servirsi della stessa tutela giuridica del *copyright* per tutelare questa forma anomala di distribuzione del software, che prevede per chiunque la possibilità di utilizzarlo, copiarlo e distribuirlo, nella forma originale o anche dopo averlo modificato, sia gratuitamente sia a pagamento. Il software libero può essere tale solo se viene messo a

disposizione assieme al codice sorgente, da cui il detto: se non è sorgente, non è software (*if it's not source, it's not software*).

È importante sottolineare che la libertà del software libero non sta tanto nel prezzo, che eventualmente può anche essere richiesto per il servizio di distribuzione, ma nella possibilità di usarlo senza vincoli, di copiarlo come e quanto si vuole, di poterne distribuire le copie, di poterlo modificare e di poterne distribuire anche le copie modificate. Il software che non può essere commercializzato, pur soddisfacendo i punti elencati qui, viene considerato software semi-libero.

L'enfasi ideologica che conteneva in sé il pensiero della Free Software Foundation darà origine a una serie di spaccature all'interno della loro organizzazione, che come conseguenza porterà alla costituzione del Movimento Open Source.

Il Movimento Open Source

Nel 1998 Bruce Perens, Eric Raymond e altre personalità nel settore del software libero si convinsero che i principi di libertà associati ad esso fossero malvisti nel mondo degli affari, a causa della loro carica ideologica. Decisero perciò di evitare accuratamente ogni riferimento a considerazioni politiche o di principio, e di lanciare una campagna di promozione del software libero che ne mettesse in luce i numerosi vantaggi pratici, come la facilità di adattamento, l'affidabilità, la sicurezza, la conformità agli standard, l'indipendenza dai singoli fornitori.

A tal fine scrissero la *Open Source Definition*, il documento fondamentale del loro movimento, che può essere considerata una sorta di carta dei diritti dell'utente di computer. La *Open Source Definition* stabilisce quali diritti una licenza software deve garantire per poter essere certificata come open source.

Il Movimento Open Source è stato un successo e ha contribuito a sdoganare il concetto di software libero in campo aziendale, dove era spesso guardato con sospetto.

La voluta neutralità del Movimento Open Source nei confronti degli aspetti etici e politici del software libero è la caratteristica sostanziale che lo distingue dalla filosofia del Software Libero, che al contrario pone l'accento sulle motivazioni ideali e ideologiche. Parlare di software libero piuttosto che di *open source* è una questione politica piuttosto che pratica; i due

movimenti concordano infatti sulle licenze considerate accettabili ed hanno obiettivi e mezzi comuni.

A differenza del software proprietario, dove l'utente dispone soltanto della licenza d'uso del programma, l'utente di un pacchetto software *open source* può migliorarlo o adattarlo alle proprie esigenze, purché renda le modifiche disponibili ad altri (tipicamente attraverso Internet). Questo ha creato una forte concorrenza ai produttori di software con licenze proprietarie, poiché molti utenti hanno imparato ad apprezzare le facilità offerte dalle licenze *open source*. Di conseguenza la vendita in questo settore di distribuzione è sempre più proficua.

La filosofia che sta alla base dell'*open source* è quella di mettere l'utente finale al centro dello sviluppo della tecnologia informatica e dell'interazione uomo-computer. I volontari che hanno sviluppato prodotti come LINUX cooperano con i singoli e le aziende del settore, grazie ai diritti garantiti dall'*open source*. Queste interazioni sono apprezzate perché l'*open source* garantisce:

- il diritto di fare copie del programma e di distribuirle;
- il diritto d'accesso al codice sorgente del software, condizione necessaria per poterlo modificare;
- il diritto di apportare migliorie al programma.

La rilevanza economica di questo settore dell'informatica è ancora abbastanza limitata, ma è in fortissima crescita ormai da alcuni anni, e tutto lascia supporre che tale crescita continui nel prossimo futuro, anche grazie ai vantaggi tecnici ed economici di questa forma di distribuzione. Ad oggi, la distribuzione *open source* è ampiamente diffusa in ambito accademico, industriale e fra gli appassionati di calcolatori, soprattutto grazie ai sistemi che sono riusciti a realizzare. Questi sistemi sono disponibili a costi molto bassi, ben inferiori a quelli di analoghi sistemi proprietari. Tuttavia, a causa delle loro caratteristiche, il loro uso richiede una buona cultura di base di tipo informatico. In ambito accademico viene molto apprezzata la possibilità di personalizzare i programmi; in ambito industriale, si apprezza l'affidabilità dei sistemi aperti, dovuta al fatto che quando un utente corregge un errore in un programma solitamente rende disponibile la correzione a tutta la comunità degli utenti Internet.

Compatibilità ed evoluzione nel rapporto fra tutela del diritto d'autore ed elaborazione del software, con specifico riferimento al software open source

La titolarità dei diritti d'autore derivanti dallo sviluppo e dalle elaborazioni del software in generale e di quello con connotazione *open source* in particolare, pone un problema essenziale che riguarda l'individuazione della migliore protezione da adottare per l'opera derivata a favore degli autori delle predette elaborazioni.

La questione dei miglioramenti apportati alle varie opere non è certamente un problema nuovo, nuove sono la forma e l'estensione del fenomeno. Probabilmente molto più di qualsiasi altra opera dell'ingegno un software vale quanto più è possibile adattarlo, modificarlo, aggiornarlo alle realtà industriali, commerciali, pubbliche nelle quali è utilizzato. L'aspetto è tanto più rilevante se teniamo conto della semplicità con cui l'utente esperto può modificare, trasformare, migliorare il software ed immetterlo in commercio. A differenza delle altre opere dell'ingegno, inoltre, dove l'integrità costituisce un valore, nel software la possibilità di successive elaborazioni rende ancora più conveniente e funzionale ciò che è stato creato.

Nell'ordinamento italiano i diritti patrimoniali e morali d'autore spettano a titolo originario innanzitutto alla persona fisica che ha creato l'opera. La partecipazione invece di più soggetti all'attività creativa può dar luogo a seconda dei casi a un'opera in comunione, collettiva, composta o ad un'elaborazione creativa in cui la titolarità originaria dei diritti è diversamente ripartita fra i vari coautori a seconda delle modalità del contributo. Nelle opere in comunione si presume che i diritti appartengano in parti uguali a tutti i coautori, salvo prova contraria (art. 10 Legge Diritto d'Autore). Nelle opere collettive i diritti sulla raccolta nel suo complesso spettano a chi ne abbia organizzato e diretto la creazione (art. 7 Legge Diritto d'Autore) mentre quelli sulle singole parti appartengono ai diversi collaboratori. Nelle opere composte i diritti d'autore appartengono secondo alcuni separatamente ai diversi autori, secondo altri invece in comunione ex art. 10. Nelle creazioni derivate i diritti sull'opera derivata spetterebbero soltanto all'autore dell'elaborazione, secondo la tesi che sembra prevalere, o in comunione con l'autore dell'opera base, secondo un diverso orientamento. L'art. 4 Legge Diritto d'Autore, dedicato all'opera derivata,

dispone che: «Senza pregiudizio dei diritti esistenti sull'opera originaria, sono altresì protette le elaborazioni di carattere creativo dell'opera stessa, quali le traduzioni in altra lingua, le trasformazioni da una in altra forma letteraria od artistica, le modificazioni ed aggiunte che costituiscono un rifacimento sostanziale dell'opera originaria, gli adattamenti, le riduzioni, i compendi, le variazioni non costituenti opera originale». Si intende per elaborazioni creative le opere protette in cui insieme al contributo creativo dell'autore sono presenti gli elementi espressivi dell'opera originaria. Le predette elaborazioni si distinguono anche per la minore o maggiore caratterizzazione rispetto all'opera base: secondo l'opinione espressa in dottrina, nel primo caso l'opera derivata beneficia di una protezione autonoma ma il suo sfruttamento economico può essere sottoposto al veto dell'autore; se invece l'elaborazione ha assunto un suo valore autonomo, ben distinguibile, la stessa beneficerebbe di una protezione autonoma ma l'esercizio delle corrispondenti facoltà patrimoniali non sarebbe neppure esposto al veto dell'autore dell'opera base.

Di fronte alla sistematica attività di riadattamento, modifica, sviluppo, aggiornamento del software, realizzata in tutti i settori produttivi, alla necessità di dare certezza ai titolari dei diritti, nonché a coloro che apportano miglioramenti di natura consistente, è necessario, a livello comunitario, predisporre una specifica normativa di armonizzazione relativa alla disciplina dell'opera derivata. Tale disciplina dovrebbe avere ad oggetto, e quindi definire, nei contenuti essenziali, in quali casi possa parlarsi di elaborazioni creative e quindi l'opera derivata assume un carattere autonomo, meritevole di una tutela, non subordinata all'autore dell'opera stessa, e quando invece le predette elaborazioni creative costituiscono un semplice aggiornamento ed evoluzione, di carattere tecnico o commerciale, meritevole di tutela, ma non sganciato dall'opera originale e perciò imprescindibile dal consenso del titolare dei diritti sulla relativa opera.

Nel settore *open source* l'indicazione di una specifica normativa che protegga gli autori degli aggiornamenti del software potrebbe spingere verso un ulteriore sviluppo di tale forma di utilizzo, i cui effetti benefici sono ormai noti in tutti gli ambiti, specialmente nelle pubbliche amministrazioni. È certo che nel caso del software *open source* si possono prospettare

molteplici situazioni di conflitto. Occorre tuttavia porsi il problema di come gestire il diritto d'autore in un contesto di manifesta e continua modifica di un determinato prodotto che normalmente, per la sua stessa natura, ha già come creatori diversi coautori. Tale considerazione sottolinea, da un lato, l'importanza di un accordo armonioso da parte di tutti i coautori, che hanno i diritti a titolo originario, nella concessione dei diritti che vengono estesi agli utenti quali titoli derivati e dall'altro la necessità di normative *ad hoc* per questa particolare forma dell'ingegno.

La tutela delle banche dati

Le banche dati spaziano dagli elenchi telefonici alle enciclopedie multimediali passando per gli archivi delle biblioteche. Con la direttiva 1996/9/CE, attuata in Italia con d.lgs. 169/1999, la copertura del diritto d'autore è stata estesa anche alle raccolte di dati che nella loro struttura sono originali.

Le banche dati e il cosiddetto *diritto sui generis*

Le banche dati hanno causato minori problemi interpretativi grazie alla loro natura giuridica più chiara e delineata.

Il fenomeno della banca dati nel senso generico di «raccolta di informazioni» possiede una storia decisamente radicata se pensiamo a tutte le opere che raccolgono altre opere (ad esempio il museo, inteso come opera indipendente dalle singole opere che contiene, si avvicina moltissimo all'idea moderna di banca dati). La stessa contiguità concettuale è correttamente individuabile nel complesso delle opere di compilazione: le antologie di poesie, racconti, immagini, le opere enciclopediche e le rassegne di massime giurisprudenziali (o addirittura gli elenchi di indirizzi e numeri telefonici disposti per settori commerciali, come ad esempio le Pagine Gialle).

La particolarità di questa categoria di opere sta nel fatto che il requisito della creatività è da ricercarsi non nelle caratteristiche espressive delle singole opere raccolte (le quali restano indipendentemente sottoposte alla loro specifica tutela) quanto piuttosto nel criterio con cui l'autore-compilatore ha operato la raccolta e ne ha disposto il risultato.

Conferma di questo principio si riscontra nella maggioranza delle definizioni giuridiche attribuite al fenomeno, come quella seguente: «Banca

dati è una raccolta di informazioni o elementi, costituenti o meno opere dell'ingegno, scelti e/o disposti secondo determinati metodi o sistemi in modo da consentire all'utilizzatore di accedere alle singole informazioni e al loro insieme».

L'aspetto però più problematico di questa categoria di opere riguarda una sua sottocategoria: le banche dati elettroniche, ossia le opere compilative realizzate con l'elaboratore ed usufruibili per mezzo di metodi informatici. Le banche dati elettroniche possono essere sia statiche che dinamiche. Si consideri come esempio di opera compilativa elettronica statica una raccolta di testi legislativi (oppure di fotografie, oppure di definizioni enciclopediche) edita su CD-ROM: con questo supporto si mantengono tutte le caratteristiche di malleabilità e liquidità dei dati, ma l'integrità ontologica dell'opera è garantita.

Si consideri invece come esempio di opera compilativa elettronica dinamica un repertorio di massime giurisprudenziali pubblicato su Internet e aggiornato costantemente: quale sarà il nucleo dell'opera da cui esigere il requisito della creatività?

Come tutelare ogni singola modifica? Il requisito della creatività è soddisfatto dalla messa in rete di un primo *stock di dati* i quali sono già disposti in un determinato criterio scelto dall'autore-compilatore e costituiscono già un'opera sufficientemente definita; invece, «ogni memorizzazione successiva di dati condurrà ad una modificazione (non creativa) dell'opera iniziale».

Un ultimo rilievo, molto importante a livello di classificazione giuridica, riguarda l'inserimento delle banche dati nel tipo delle opere collettive ai sensi dell'art. 3 Legge Diritto d'Autore.

Vi è però una importante considerazione da fare. Quanto detto fin qui vale solo nei casi in cui nella banca dati si possa riscontrare quel famigerato «carattere creativo» che la fa rientrare a tutti gli effetti nella categoria delle opere dell'ingegno in senso più classico, e dunque la rende tutelata dal diritto d'autore come tutte le altre opere elencate nell'articolo 2 Legge Diritto d'Autore.

Vi è però una speciale forma di tutela giuridica anche per le banche dati che non denotano un livello minimo di carattere creativo ma che hanno

comunque richiesto un sostanziale investimento per la raccolta, l'organizzazione e pubblicazione dei dati.

Infatti, con l'avvento delle nuove modalità di memorizzazione e di gestione tecnologica delle informazioni, i database sono diventati una parte fondamentale dell'attività di produzione culturale e tecnica. Dunque il mondo del diritto ha iniziato ad interrogarsi se fosse necessario prevedere specifiche forme di tutela di questa nuova categoria di creazioni, o se al contrario fosse sufficiente applicarvi (in maniera estensiva) le categorie e i principi già esistenti nel diritto d'autore.

L'inadeguatezza della tutela di diritto d'autore in senso stretto

Già da una prima lettura della norma si può afferrare agevolmente che la definizione di opere collettive (nel senso di collezioni di opere) si riferisce a fenomeni non sempre equiparabili ad una banca dati. Non tutte le banche dati possiedono il requisito della scelta e della disposizione del materiale secondo criteri creativi; «non in particolare quelle che, proponendosi di fornire tutte le informazioni disponibili su un dato argomento, non attuano alcuna selezione e che presentano le informazioni stesse secondo un ordine banale o imposto da esigenze informative».

Inoltre esiste un altro «tallone di Achille» del diritto d'autore nella sua applicazione ad opere atipiche come le banche dati: il principio per cui il diritto d'autore copre solo la forma espressiva di un'opera, cioè il modo con cui l'autore ha espresso la sua idea e non l'idea in sé. Dunque specialmente in questo caso, sulla base del solo diritto d'autore, un altro soggetto potrebbe utilizzare i contenuti della banca dati modificandone il criterio di disposizione e organizzazione, realizzando a tutti gli effetti un'opera diversa dal punto di vista giuridico, ma ripetitiva e "parassitaria" nella sostanza.

Con la sola applicazione del diritto d'autore un'ampia fetta di banche dati rimarrebbe priva di tutela giuridica; rimarrebbe solo la tutela derivante dai principi della concorrenza sleale o l'eventuale applicazione di sistemi tecnologici di protezione.

Ciò è stato considerato insufficiente da parte del legislatore comunitario, il quale, dopo un acceso dibattito sull'opportunità di questa scelta, ha deciso di attivarsi con un'apposita direttiva.

Tale scelta è stata sostenuta dall'idea secondo cui certi tipi di banche dati, che per loro natura sarebbero escluse dal campo d'azione del diritto

d'autore, richiedono comunque un grande investimento da parte di soggetti specializzati e quindi questo investimento rimane di per sé meritevole di essere tutelato e di conseguenza incentivato.

Un duplice livello di tutela: la direttiva del 1996 e il *diritto sui generis*

Dunque il legislatore europeo nel 1996 ha deciso di delineare un particolare modello di tutela, secondo il quale le banche dati devono essere sottoposte a un duplice livello di protezione. Con la Direttiva n. 96/9/CE, da un lato le banche dati sono state formalmente inserite tra le categorie di opere dell'ingegno tutelate da diritto d'autore previste dalla normativa comunitaria; dall'altro lato sono stati creati appositi diritti per il costitutore della banca dati.

La parte davvero innovativa (e anche la più criticata) della direttiva è il Capitolo III nel quale vengono istituiti nuovi diritti per la tutela delle banche dati prive di carattere creativo e quindi non considerate a pieno titolo opere dell'ingegno.

Tali diritti (generalmente denominati con la locuzione latina *diritto sui generis*, proprio ad indicare la loro peculiarità rispetto ai diritti d'autore e ai diritti connessi) sono diritti esclusivi che sorgono in capo ad un soggetto definito dalla norma *costitutore della banca dati*, si riferiscono all'investimento sostenuto per la realizzazione del database (e non all'apporto creativo come nel caso dei diritti d'autore e dei diritti connessi) e durano 15 anni dalla costituzione della banca dati. I principi della direttiva sono poi stati recepiti dagli stati membri della UE e sono divenuti parte integrante nelle normative nazionali, rendendo così l'assetto normativo di tutta l'Unione Europea abbastanza uniforme.

Il costitutore ha il diritto esclusivo di controllare per 15 anni tutte le attività di estrazione e modifica dati sul database (o su una sua parte sostanziale) da lui realizzato e messo a disposizione del pubblico. Ciò – appunto – avviene anche quando si tratti di un database senza carattere creativo, ma che abbia comunque richiesto un investimento rilevante sotto il profilo qualitativo o quantitativo.

Il plagio: una nuova pirateria

Nonostante l'enorme contributo dell'industria del software all'economia mondiale, la crescita di questo segmento continua ad essere frenata, in maniera particolare proprio in Europa, dalla cd. Pirateria Informatica: la copia e la distribuzione illegale di programmi per elaboratore.

Per pirateria informatica si intendono due fenomeni piuttosto diversi:

A) la pirateria cd. Commerciale che si realizza mediante la riproduzione non autorizzata e la successiva distribuzione di copie di software;

B) la pirateria cd. Industriale che consiste nella riscrittura del codice sorgente di un programma in maniera e secondo una struttura sostanzialmente analoga a quella del software copiato ma caratterizzata da alcune, più o meno rilevanti, operazioni di maquillage estetico volte a dissimulare il plagio così realizzato.

Il plagio può essere considerato a tutti gli effetti una nuova forma di pirateria anche se compiere azioni di plagio non significa necessariamente violare il diritto d'autore.

Il plagio invece consiste nel rubare e spacciare come proprie le idee o le parole di qualcun altro, senza citare la fonte; citazioni brevi o attribuite generalmente non costituiscono plagio. Tipicamente non ci sono leggi che regolano espressamente il fenomeno del plagio, così questo reato finisce spesso col non essere perseguito.

Le accuse di plagio non hanno garanzie: non richiedono registrazione e non richiedono che l'accusatore abbia delle prove. Nella maggior parte dei casi terze parti identificano potenziali azioni di plagio, rendono pubbliche le loro accuse poi lasciano che l'opinione pubblica tragga le proprie conclusioni. L'accusatore non viene mai convocato per verificare la fondatezza o meno delle sue dichiarazioni.

Le accuse di plagio possono derivare da motivazioni discutibili dell'accusatore, ad esempio uno studente potrebbe accusarne un altro di plagio perché è invidioso dei suoi risultati.

Il plagio richiede e intrinsecamente presuppone che l'accusato sia colpevole, ma in molti casi di plagio gli accusati hanno dichiarato di non averlo fatto intenzionalmente.

La mancanza di regolamenti standard nei casi di plagio rende virtualmente impossibile la difesa dell'accusato e un'accusa di plagio spesso rappresenta

una macchia indelebile contro i valori professionali e l'etica personale dell'accusato. Questa moderna versione di lettera scarlatta evidenzia uno dei principali problemi connessi al plagio: senza uno standard chiaro, senza onere della prova, né possibilità di difesa, una mera accusa di plagio può rovinare la reputazione più velocemente di qualunque violazione del *copyright*.

Il modo migliore per sfuggire ai luminosi e poco lusinghieri riflettori che si accendono sui casi di plagio, è quello di capire quali sono gli standard nelle modalità di inserimento delle citazioni (che saranno diversi a seconda del settore), impararli e, quando è possibile, cercare di ottenere formazione e assistenza nell'applicazione di questi standard.

La nuova legge antipirateria

La l. 248/2000 ha innovato la disciplina della materia inasprendo sensibilmente il regime sanzionatorio introducendo *ex novo* l'obbligo di apporre su tutti i supporti contenenti opere protette – ivi incluso, ovviamente il software – uno speciale bollino.

In particolare per quanto concerne la duplicazione, riproduzione e detenzione di programmi per elaboratore non originali il legislatore ha definitivamente statuito che integra gli estremi della fattispecie criminale di cui all'art. 171bis anche la condotta posta in essere «al fine di trarne profitto».

Con tale espressione che è andata a sostituire quella «a fini di lucro» contenuta – come si è detto – nel vecchio testo dell'art. 171bis il legislatore recependo le istanze delle grandi software house ha inteso colpire – in modo particolare – quella prassi assai diffusa nel nostro paese – specie in aziende di rilevanti dimensioni – di acquistare un numero di licenze software sensibilmente inferiore a quello effettivamente necessario e di porre poi in circolazione tutta una serie di copie «pirata» dei medesimi programmi.

La nuova legge ha, inoltre, inasprito le sanzioni penali previste per tali illeciti dalla disciplina previgente e introdotto agli artt. 174bis e 174ter tutta una serie di sanzioni accessorie di carattere amministrativo e pecuniario che vanno dal pagamento di una multa pari al doppio del prezzo di mercato dell'opera o del supporto oggetto della violazione sino alla possibile sospensione dell'attività per un periodo non inferiore a 15 giorni e non

superiore a 3 mesi che può essere disposta dal questore su segnalazione del pubblico ministero.

Nel tentativo di indurre gli utenti di software non originale a porsi in regola e a pagare il prezzo delle licenze, all'art. 171 novies è stata poi introdotta una figura sui generis di «pentimento operoso» prevedendo che la pena principale per i reati in danno ai diritti relativi ai programmi per elaboratore possa essere diminuita da un terzo alla metà e che non si applichino le pene accessorie nel caso in cui il soggetto agente, prima che la violazione sia stata specificamente contestata in un atto dell'autorità giudiziaria, la denunci spontaneamente.

Come si è anticipato, l'altra rilevante novità introdotta con la l.248/2000 è rappresentata dalla previsione dell'art. 10 con la quale si è resa obbligatoria l'apposizione di un contrassegno (il c.d. «bollino SIAE») su ogni supporto contenente un'opera protetta o anche solo una sua porzione.

Ai sensi di tale previsione, il contrassegno dovrà essere richiesto alla SIAE e apposto – a spese, evidentemente di chi commercializza il supporto – su ogni supporto contenente software, opere multimediali, suoni, voci o immagini in movimento nonché opere letterarie, musicali, teatrali, cinematografiche o di altri genere rientranti nell'ambito di applicazione della l. 633/1941 comunque destinato alla commercializzazione a fini di lucro.

Sempre secondo quanto disposto dall'art. 10 il contrassegno viene apposto ai soli fini della tutela dei diritti relativi alle opere dell'ingegno e, quindi, previa attestazione da parte del richiedente dell'assolvimento dei relativi obblighi previsti dalla disciplina vigente.

Importante, anche se di difficile interpretazione e sicura fonte di equivoci e fraintendimenti, è poi la previsione secondo cui il «bollino SIAE» può non essere apposto sui supporti contenenti programmi per elaboratore utilizzati esclusivamente mediante elaboratore elettronico e sempre che tali programmi non contengano suoni, voci o sequenze di immagini in movimento tali da costituire opere fonografiche, cinematografiche o audiovisive intere non realizzate espressamente per il programma per elaboratore o loro brani o parti superiori al 50% dell'opera dalla quale sono tratti.

Giurisprudenza: alcuni casi

La Corte di Giustizia nega la tutela a funzionalità, linguaggio di programmazione e formato dei file di dati

(Corte di Giustizia, grande sez., 2-5-2012, C-406/10, Sas Institute Inc. c. World Programming Ltd.)

La sentenza citata è di grande rilevanza sia teorica che pratica, poiché chiarisce il ruolo di funzionalità, linguaggio di programmazione e formato dei file di dati e sottolinea l'importanza dell'interoperabilità fra programmi diversi. Sembra prevalere una maggiore attenzione agli interessi generali rispetto ai diritti esclusivi sui beni immateriali.

La sentenza è l'esito di una controversia che opponeva le società SAS Institute Inc. e World Programming Ltd. SAS Institute è una società che sviluppa programmi per elaboratore analitici ed ha creato il c.d. sistema SAS, un insieme integrato di programmi che consente agli utenti di effettuare analisi statistiche, scrivendo ed eseguendo scripts in un linguaggio *ad hoc*, chiamato «linguaggio SAS».

Per realizzare software alternativi in grado di eseguire applicazioni scritte il tale linguaggio, WPL ha creato il «World Programming System» (WPS), un sistema in grado di emulare molte delle funzionalità di SAS attraverso il quale gli utenti del sistema SAS possono eseguire su WPS gli script scritti in linguaggio SAS.

SAS Institute aveva accusato WPL di contraffazione e di aver violato il suo diritto d'autore sui programmi per elaboratore e sui manuali d'uso relativi al suo sistema informatico.

Effettivamente numerosi elementi di WPS sono identici a SAS ma la decisione della Corte si è basata sul fatto che WPL non ha avuto accesso al codice sorgente dei moduli SAS per realizzare il proprio programma.

Dal codice sorgente si possono conoscere la struttura del programma e la sua logica di base inoltre si può realizzare qualunque modifica; è per questo che nei tradizionali modelli di concessione in uso di software non è consentito all'utente di accedere al codice sorgente.

WPS non ha avuto accesso al codice sorgente del sistema SAS, bensì ne ha osservato e analizzato i meccanismi di funzionamento per ricostruire un sorgente che producesse effetti simili.

SAS ha accusato WPL non solo di aver copiato il sistema SAS ma anche di aver violato i diritti d'autore esistenti sui manuali di SAS pubblicando nei manuali di WPS istruzioni molto simili a quelle di SAS.

La Corte ha valutato innanzitutto cosa si intendesse con forma espressiva del software e ha concluso che, in base all'art. 1 della Direttiva 1991/250/CE, la funzionalità di un programma, nonché il linguaggio di programmazione e il formato dei file di dati utilizzati rappresentano una forma espressiva del programma medesimo e, in quanto tali, possono essere protetti dal diritto d'autore.

Dottrina e giurisprudenza concordano sul fatto che il diritto d'autore non tuteli le idee ma la forma in cui esse sono espresse, poiché è con la forma che si manifesta la creatività degli autori. Nella sentenza in oggetto, la Corte di Giustizia ribadisce che tutelare il software mediante il diritto d'autore significa tutelarne la forma espressiva individuale, lasciando spazio sufficiente ad altri autori per creare programmi simili o anche identici, purché non siano copiati.

Si potrebbe obiettare che in un programma i singoli passaggi e le loro concatenazioni sono finalizzati all'idea che il procedimento mira a realizzare quindi non è possibile scindere la forma dal contenuto, anzi sono proprio le istruzioni più che la forma a rappresentare la creazione intellettuale di un software.

In base alla Convenzione di Berna, i programmi per elaboratore sono protetti come opere letterarie, sia in codice sorgente, che in codice oggetto.

L'art. 1 della Direttiva 1991/250/CE va interpretato nel senso di considerare il codice sorgente come oggetto del diritto d'autore.

Numerose sentenze chiariscono che non può esserci violazione del diritto d'autore se la somiglianza fra due codici dipende dalla limitatezza delle forme di espressione disponibili: se l'interfaccia presenta delle limitazioni che impediscono di rappresentare idee simili in modalità diverse, si deve intendere che idee e forma si sono fuse insieme perciò non si può parlare di violazione del diritto di autore.

Sebbene il codice sorgente possa essere considerato forma di espressione e quindi vada tutelato dal diritto d'autore, tale tutela non può offrire all'autore la possibilità di monopolizzare le idee poiché ciò rappresenterebbe un freno inaccettabile allo sviluppo industriale e al progresso tecnico.

Nel caso SAS Institute/WPL viene chiarito che né le funzionalità di un programma, né il linguaggio di programmazione né il formato dei file di dati utilizzati possono essere considerati forma di espressione del programma.
WPL non ha avuto accesso al codice sorgente del programma di SAS Institute e non ha effettuato una decompilazione del codice oggetto dello stesso ma è riuscito a riprodurne le funzionalità grazie all'osservazione, allo studio e alla sperimentazione del comportamento del sistema SAS; la sentenza della Corte ha chiarito che tale attività di studio e ricerca ha reso pienamente lecita la creazione del programma WPS.
La sentenza ha affrontato anche altre due questioni.
La prima riguarda il diritto dell'utente in possesso di regolare licenza d'uso di utilizzare una copia del programma, senza chiederne l'autorizzazione al titolare del diritto, allo scopo di osservare e studiare il funzionamento del programma; la Corte ha ribadito l'esistenza di tale diritto ed evidenziato l'importanza di tutelare gli interessi generali, limitando l'autonomia privata e il conseguente monopolio delle idee.
L'altra questione concerne la riproduzione del manuale d'uso: poiché il manuale d'uso realizzato da SAS a supporto del proprio software è un'opera letteraria, occorreva chiarire se la realizzazione del manuale per WPS, che in parte riproduce quello di SAS, rappresentasse o meno una violazione del diritto d'autore.
Il punto cruciale della decisione sta nella valutazione di quali parti siano state riprodotte: il diritto d'autore non tutela i manuali d'uso, a meno che non riproducano parti sostanziali del programma e non era questo il caso dei manuali SAS, dal momento che i tecnici di WPL per poter creare WPS hanno dovuto studiare e sperimentare il funzionamento di SAS.
Risulta ormai evidente che tutte le forme di proprietà intellettuale costituiscono elementi fondamentali della nuova economia basata sulla conoscenza e che gran parte del successo delle imprese europee dipenderà in futuro dalle attività immateriali.
In questa rivoluzione sarà importante tutelare in primo luogo gli interessi generali riconducibili, ancor prima che ai consumatori, ai cittadini della società dell'informazione, mentre la c.d. proprietà intellettuale e i diritti di sfruttamento economico ad essa connessi tenderanno sempre più a passare in secondo piano.

Il «caso Oracle»

(Corte giust., 3 luglio 2012, c. 128/11, UsedSoft GmbH c. Oracle International Corp., in Il Foro Italiano, 2012, p. 377 e ss.)

Tra le recenti sentenze rientranti nel multiverso del diritto dell'informatica, che hanno avuto maggior risonanza, vi è sicuramente quella della Grande Sezione della Corte di giustizia europea n. C-128/11 del 3 luglio 2012, nota ai più come «caso Oracle» (Corte giust., 3 luglio 2012, c. 128/11, UsedSoft GmbH c. Oracle International Corp., in Il Foro Italiano, 2012, p. 377 e ss.).

Con la sentenza C-128/11 la Corte si è pronunciata sull'interpretazione della direttiva 2009/24/CE del Parlamento Europeo e del Consiglio, del 23 aprile 2009, relativa alla tutela giuridica dei programmi per elaboratore.

La sentenza è stata emessa nell'ambito di una controversia tra la Oracle International Corporation (in prosieguo: la «Oracle») e la UsedSoft GmbH (in prosieguo: la «UsedSoft») in merito alla commercializzazione da parte di quest'ultima di licenze di programmi per elaboratore usati della Oracle.

La Corte ha stabilito che la direttiva 2009/24/CE deve essere interpretata nel senso che il diritto di distribuzione della copia di un programma per elaboratore è esaurito qualora il titolare del diritto d'autore, che abbia autorizzato, eventualmente anche a titolo gratuito, il download della copia su un supporto informatico via internet, abbia parimenti conferito, a fronte del pagamento di un prezzo diretto a consentirgli di percepire una remunerazione corrispondente al valore economico della copia dell'opera di cui è proprietario, il diritto di utilizzare la copia stessa, senza limitazioni di durata.

Con la medesima pronuncia ha inoltre ribadito che, secondo la direttiva di cui sopra, in caso di rivendita di una licenza di utilizzazione che implichi la rivendita di una copia di un programma per elaboratore scaricata dal sito internet del titolare del diritto d'autore (licenza che era stata inizialmente concessa al primo acquirente dal titolare medesimo senza limitazione di durata ed a fronte del pagamento di un prezzo) il secondo acquirente della licenza stessa, al pari di ogni suo acquirente successivo, potrà avvalersi dell'esaurimento del diritto di distribuzione previsto dalla direttiva medesima e, conseguentemente potrà essere considerato quale legittimo acquirente di una copia di un programma per elaboratore, beneficiando del diritto di riproduzione previsto dalla direttiva in questione.

Questo noto caso ha avuto evidenti effetti anche su tutto il mercato europeo del software.

La Oracle ha impostato la sua difesa sostenendo di non procedere alla vendita di copie dei software, ma bensì di metterle gratuitamente a disposizione per tutti i propri clienti che avessero concluso uno specifico contratto di licenza, a seguito del quale sarebbero entrati in possesso della chiave di accesso per poter effettuare il download gratuito dal loro sito web.

La Corte invece ha ritenuto non plausibile questa tesi difensiva, ritenendo impossibile scindere il momento del download da quello della sottoscrizione del contratto di licenza, sottolineando che ove si effettuasse il download di un programma senza stipulare un contratto di licenza, il programma stesso non potrebbe essere utilizzato legittimamente da chi lo abbia scaricato.

E come prima conseguenza dell'affermazione di questo principio di inscindibilità fra contratto di licenza e download, c'è quella dell'abolizione della distinzione, formatasi nella prassi, tra la vendita di software su supporto fisico e quella su supporto immateriale. E ciò suggerisce una riflessione.

La decisione del legislatore Europeo nel tutelare i programmi per elaboratore con il diritto d'autore è stata presa perché in esso fu individuata la disciplina più idonea a regolare gli interessi di chi effettuava ingenti investimenti nella realizzazione dei prodotti informatici.

Ma se si va ad analizzare la portata di questa pronuncia della Corte di giustizia, non ci si può non rendere conto che qui ad essere regolati non sono solo ed esclusivamente gli interessi delle parti in causa, bensì quelli di tutti i cittadini. La Corte si è nettamente distaccata dal rigore dello schema classico del diritto d'autore, indicando, di fatto, una nuova strada da percorrere.

Risulta evidente, infatti, che con questa pronuncia la Corte voglia creare i presupposti per regolare l'ipotetica nascita di un mercato dell'usato dei software, anche di tipo immateriale (cioè via download), la qual cosa non dovrebbe destare stupore alcuno, essendoci stata analoga conseguenza anche per le altre opere dell'ingegno (si pensi, ad esempio, al mercato dei libri o dei film usati).

Del resto bisogna considerare che ormai, oltre alle attività professionali, per ottimizzare l'organizzazione dei propri affari anche la quasi totalità delle

attività aperte al pubblico si stanno sempre più affidando ai più svariati software, come i ristoranti o i bar che vogliono gestire telematicamente le ordinazioni o come i negozi di vestiti o scarpe che vogliono controllare il loro inventario.

E l'elenco si potrebbe estendere anche ai software che rientrano nelle sfere quotidiane della vita cittadina, come i software per il controllo dei semafori o dei treni metropolitani, quelli per la navigazione satellitare o quelli per la gestione di elettrodomestici (non a caso oggi si sta cominciando a parlare sempre di più di «Internet delle cose»).

Insomma il software sta diventando sempre di più parte della vita quotidiana di ciascun cittadino dell'Unione Europea e proprio per questo è fortemente ipotizzabile che a breve si avvertirà sempre di più l'esigenza della nascita di un vero e proprio mercato del software usato (si pensi a un soggetto che decide di aprire un ristorante, e che per limitare i costi di inizio attività, decida di acquistare un software usato).

A ulteriore riprova di questa chiave di lettura vi è poi la chiara affermazione della Corte con la quale viene tolto ogni dubbio sulla qualifica di *legittimo acquirente* dei successivi acquirenti di un programma per elaboratore, e sul fatto che il titolare del diritto d'autore di un software possa o meno, attraverso apposite disposizioni contrattuali, opporsi alla rivendita del software stesso.

Alla luce delle suesposte considerazioni, la Corte di giustizia ha dunque legittimato l'attività commerciale della UsedSoft, che quindi avremmo ragione di definire come la pioniera del mercato del software usato nell'Unione Europea. In virtù di ciò appare evidente che, al fine della risoluzione di tale caso, determinante sia stato l'intervento del legislatore, che se non avesse introdotto, a suo tempo, nel novero del diritto d'autore, il c.d. principio di distribuzione, fino ad allora sconosciuto, avrebbe visto oggi l'esito della causa certamente capovolto.

Ma come già asserito, questo non basta. Il multiverso del diritto dell'informatica, infatti, necessita di ulteriori tutele normative e nello specifico la tutela del software non può e non deve rimanere ancorata *in toto* al diritto d'autore, e sembrerebbe rendersi dunque necessaria la continuazione di quest'opera di «integrazione normativa» avviata dal legislatore europeo.

A supporto di tale affermazione possono considerarsi i numerosi dubbi interpretativi nati in dottrina a seguito della scelta del legislatore italiano di adattare la disciplina del diritto d'autore ai programmi per elaboratore, sulla scorta di quanto accadde nella maggior parte dei paesi anglosassoni. Vi è chi ha accolto questa scelta come uno snaturamento del diritto d'autore, o chi, addirittura, ha ritenuto che l'idea di ricondurre i programmi alla letteratura era talmente peregrina da non meritare neppure considerazione. E a dire il vero non sembra così difficile condividere tali idee.

Ed è dunque per questi motivi che la pronuncia della Corte nel «caso Oracle», essendosi, come detto, discostata dalla rigidità formale del diritto d'autore, la si può certamente considerare come una vera e propria scossa al diritto d'autore stesso.

Oggi è più che mai da rilevare che con il proliferare delle nuove tecnologie, e in particolar modo di quelle portabili, come smartphone e tablet, la definizione di software, non può certamente rimanere ancorata a princìpi oramai datati (si pensi che la definizione di software è stata frutto di una elaborazione pluriennale conclusasi nel 1984 a Canberra) e di conseguenza anche la relativa tutela andrebbe rivista.

Il diritto d'autore, dunque, non sembrerebbe oggi rappresentare il mezzo di tutela più adatto per il software; sarebbe opportuno, piuttosto, sulla base delle considerazioni appena fatte, creare una forma di tutela *ad hoc*, senza assimilare il software né alle opere dell'ingegno, né tantomeno a quelle dell'industria, creando bensì una nuova categoria.

Il lavoro può sembrare alquanto oneroso, dovendosi creare, per poter categorizzare il software, un'ibrida fusione fra il diritto d'autore e il diritto industriale, ma appare comunque necessario compiere questo sforzo per poter offrire il tipo di protezione più adeguata.

Sebbene la casistica oggi, in special modo in Italia, sia ancora alquanto ridotta, nel resto d'Europa è in costante aumento, ed è destinata ad incrementarsi sempre di più, dal momento che oggi, per ogni attività della vita comune, vi sono uno o più software che ne possono regolare telematicamente lo svolgimento. Lasciare che il diritto ignori tutto questo, potrebbe rappresentare un grave e pericoloso rischio per tutti i cittadini dell'Unione Europea.

Capitolo 4 – I PRINCIPALI TOOL PER L'INDIVIDUAZIONE AUTOMATICA DI PLAGIO FRA CODICI SORGENTE

Il plagio è un enorme problema nel mondo accademico e riguarda sia documenti testuali che programmi per computer. L'individuazione manuale è complessa, non accurata e richiede troppo tempo. Analizzare il software è più complesso che analizzare dei testi scritti ed è per questo che, se nell'individuazione del plagio di testi può essere sufficiente il controllo umano, la ricerca di plagio nei software non è possibile senza l'ausilio di uno strumento automatico di rilevazione.

Per tale ragione, negli ultimi anni sono stati sviluppati numerosi tool per la rilevazione automatica di plagio fra codici sorgente. Nella tabella 2 sono elencati alcuni tool tra i più noti e utilizzati.

	JPlag	MOSS	CODEMATCH	COPY/PASTE DETECTOR (CPD)	MARBLE	SHERLOCK
Anno di creazione	1997	1994	2005	2003	2002	1994
Costi	Free con account	Free con account	Tool commerciale; free per documenti con meno di 1 M-byte di dimensione	Open source	Tool commerciale	Open source
Open source	NO	NO	NO	SI	NO	SI
Velocità	Veloce	Veloce	Molti file richiedono molto tempo	Veloce		Molti file richiedono molto tempo
Servizi	Web service	Internet	Stand alone	Stand alone		Stand alone
Risultati di memorizzazione	Memorizzazione in locale	Memorizzazione in un server remoto	Memorizzazione in locale	Memorizzazione in locale		Memorizzazione in locale
Algoritmi	Greedy String Tiling	Winnowing	String Matching	Greedy String Tiling		Token matching

Tabella 2 – Alcuni dei principali tool per l'individuazione automatica del plagio fra codici sorgente

JPlag

È stato sviluppato da Guido Malpohl all'Università di Karlsruhe nel 1997. Converte il codice sorgente di un programma in stringhe di token che rappresentano la struttura del programma poi applica l'algoritmo di Greedy String Tiling, proposto da Michael Wise. **JPlag** supporta Java, C#, C, C++, Scheme e il linguaggio naturale.

JPlag è un servizio web che cerca coppie di programmi simili in un set di programmi. Prende in input un set di programmi, li confronta a coppie e fornisce in ouput un set di pagine HTML che consentono di esplorare e comprendere a fondo e in dettaglio la similarità.

JPlag utilizza un approccio basato sul confronto della struttura dei codici: converte i codici in sequenze di token poi confronta i token alla ricerca di sottostringhe comuni.

Questo tool introduce una interfaccia utente unica per presentare le regioni simili individuate nei due programmi.

Dopo essersi loggato nel web service, l'utente sottomette una directory di file al sistema perché la analizzi; in output **JPlag** restituisce una directory di file HTML che contengono i risultati dei confronti effettuati. Per ogni coppia selezionata dall'utente è possibile visualizzare i dati in modo più dettagliato. Le linee rilevate come simili sono evidenziate con uno stesso colore, come mostrato in figura 22.

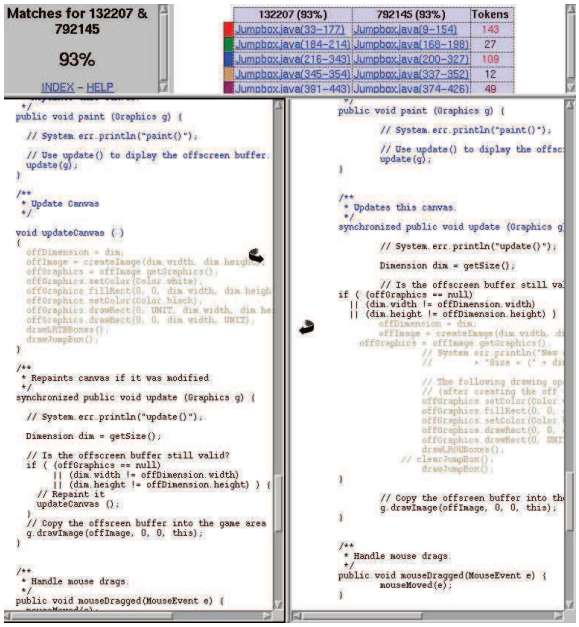

Figura 22 – Parte di JPlag che mostra il dettaglio dei risultati per una coppia di programmi selezionata dall'utente

Il funzionamento di JPlag

La fase di tokenizzazione è l'unica di **JPlag** dipendente dal linguaggio: nel caso di programmi Java o Scheme è necessario il parsing; nel caso di programmi C o C++ si effettua uno scanning.

JPlag ignora gli spazi, i commenti e i nomi degli identificatori.

Dopo la conversione i programmi sono tradotti in sequenze di token, che saranno sottoposte al confronto con l'algoritmo di Greedy-String-Tiling.

L'algoritmo prevede due step:

1. si confrontano le due stringhe per cercare la sottostringa più lunga in comune;
2. i token coinvolti dal matching nella fase 1 vengono marcati e così saranno esclusi da confronti futuri. In questo modo ci crea un *tile*, cioè una sottostringa di token costruita come associazione uno-a-uno tra la sottostringa di un codice e la sottostringa dell'altro.

Questi due step di ripetono finché non si trovano più match. La complessità di questo algoritmo è abbastanza alta: nel caso peggiore è $O((|A|+|B|)^3)$, mentre nel caso migliore (quello in cui tra A e B non ci sia nemmeno un token in comune) l'algoritmo richiede $O((|A|+|B|)^2)$ step.

La complessità media può essere ridotta utilizzando l'algoritmo di Karp-Rabin che cerca le occorrenze di una piccola stringa (un pattern *P*) in una stringa più lunga (un testo *T*) usando una funzione hash.

Valutazione del tool

JPlag ignora completamente la formattazione, i commenti, le parole e i nomi perciò resiste efficacemente alle seguenti tecniche di offuscamento del codice:

- modifica degli spazi;
- inserimento, modifica o cancellazione di commenti;
- modifica dell'output del programma o della sua formattazione;
- cambiamento dei nomi di variabili, metodi o classi;
- divisione o fusione di liste di dichiarazioni di variabili.

Rispetto al riordinamento di dichiarazioni all'interno di blocchi di codice o dell'intero programma **JPlag** risulta sensibile solo nel 15% dei test effettuati.

Le modifiche strutturali sono più difficili da rilevare ma su 134 casi, **JPlag** non è riuscito nell'individuazione solo 12 volte. Ecco i principali attacchi di questo tipo:

- modifica, cancellazione, inserimento o spostamento di strutture di controllo, come i cicli *for* o gli *if*;
- spostamento di variabili ed espressioni;
- modifica dello scope delle variabili;
- modifica delle strutture dati.

Nei casi di plagio esplicito, **JPlag** risulta un tool di rilevamento praticamente perfetto, come anche per alcuni programmi parzialmente copiati. È insensibile alla maggior parte delle tecniche di offuscamento del codice ed è abbastanza robusto anche se non sono stati scelti valori ottimali per il set di token e la lunghezza minima del match.

JPlag è stato utilizzato in contesti accademici ma anche per individuare codice duplicato in sistemi software complessi.

Per misurare la similarità fra due stringhe *A* e *B*, **JPlag** usa la seguente formula:

$$sim(A, B) = \frac{2 \sum_{match(a,b,lunght) \in tiles} length}{|A| + |B|}$$

MOSS

MOSS, acronimo di Measure Of Software Similarity, è stato sviluppato nel 1994 all'Università di Stanford da Aiken e altri. È disponibile come servizio web, accessibile tramite l'attivazione di un account. Supporta diversi linguaggi: C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086, HCL2.

Può processare file di Java, C, C++, Pascal o Ada o anche testo. Internamente converte codici sorgente in token poi usa un robusto algoritmo di *winnowing* per selezionare una sottostringa di token che rappresenta

l'impronta digitale del documento (secondo la logica del *fingerprinting*) ottenuta applicando una funzione hash.

Gli autori dell'algoritmo di winnowing a cui **MOSS** si ispira asseriscono che se contro gli attacchi di plagio di Tipo 1 e 2 (piccole modifiche relative ai commenti o ai nomi di variabili e identificatori) il loro algoritmo è robusto, nel caso di modifiche più profonde (cambiamento dell'ordine di operandi nelle espressioni o aggiunta di dichiarazioni ridondanti) funziona ancora meglio.

Quando confronta un gruppo di file, **MOSS** crea un indice inverso per mappare l'impronta del documento e la sua posizione nel documento. Quindi ogni file di programma viene usato come query con questo indice e restituisce una lista di documenti che hanno impronte in comune con la query. Il risultato è un punteggio (cioè numero di impronte in comune) per ogni coppia di documenti presente nel gruppo.

Per poter utilizzare il tool **MOSS** è sufficiente scaricare dalla pagina ufficiale lo script, salvarlo in locale e poi eseguirlo digitando il comando `moss`. I programmi sottoposti a **MOSS** devono trovarsi all'interno della directory in cui abbiamo salvato lo script moss.

Ad esempio per confrontare tutti i sorgenti nella directory corrente del sistema Unix, assumendo che siano scritti in C e che anche `moss` si trovi nella directory corrente, possiamo digitare la seguente riga di comando:

```
moss -l c *.c
```

La richiesta di confronto viene inviata a un server remoto e quando i risultati sono pronti l'utente riceve un'e-mail all'indirizzo corrispondente al proprio account che risulta aver lanciato il comando moss; l'e-mail contiene un link alla pagina web che contiene i risultati.

Generalmente se nel confronto fra due programmi C di poche centinaia di linee emerge almeno il 50% di sovrapposizioni fra codice, quasi certamente ci troviamo di fronte ad un caso di plagio. Osservando i risultati ottenuti lanciando il comando moss, si osserva che le sezioni simili sono evidenziate con un punto all'inizio e con uno stesso colore.

Utilizzando **MOSS** rilevare il plagio nelle esercitazioni di programmazione dei propri studenti è semplice: una volta che lo script di **MOSS** è stato installato, è sufficiente invocare il comando e aspettare che arrivi l'e-mail dal server (approssimativamente spedendo da 75 a 120 programmi C di

poche centinaia di linee ciascuno si ottengono i risultati richiesti nell'arco di una giornata).

Una volta che dai risultati di **MOSS** emerge il sospetto di un caso di plagio, il docente deve scrivere una e-mail allo studente per chiedere che spieghi dettagliatamente e per iscritto il modo in cui ha proceduto per realizzare il proprio lavoro, al fine di chiarire cosa sia realmente accaduto.

Al Dipartimento di Ingegneria e Scienze dell'Informazione dell'Università della Florida si applicano politiche severe di fronte ai casi di plagio: se lo studente confessa il proprio reato, riceverà un pessimo voto; se commette ulteriori azioni di plagio, verrà espulso dal corso di studio o addirittura dall'università. Gli studenti sono informati di questo regolamento fin dal primo anno di corso.

MOSS è utilizzato con successo in numerose università americane.

L'approccio PIY

Una ricerca pubblicata nel 2015 propone un approccio chiamato PIY (Program It Yourself) che sfruttando il tool **MOSS** ne migliora l'accuratezza.

Si possono analizzare i codici sorgente in base a tre livelli di granularità: progetto, form, sub-routine. A livello di progetto il confronto è semplice: due progetti che appaiono simili al di sopra di una certa soglia, potrebbero indicare la presenza di plagio.

Analizzare i codici a livello di sub-routine incrementa il numero di confronti poiché si confrontano porzioni di codice molto piccole.

In questa ricerca si sceglie di lavorare a livello di form, ovvero di confrontare ogni form di un progetto in maniera indipendente.

Sia PIY che **MOSS** sono basati su una rappresentazione del codice di tipo *k-gram*. L'idea di base di questa tecnica è di creare una lista di tutte le sottostringhe di lunghezza *k* che appaiono in un particolare documento insieme alle rispettive frequenze. Un *k-gram* è una sottostringa contigua di lunghezza k, quindi il numero di *k-gram* generati da una stringa di lunghezza *N* è esattamente $N - (k - 1)$, come mostrato in figura 23.

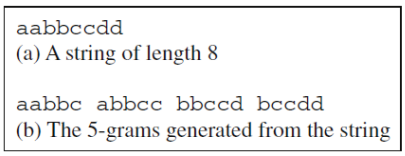

Figura 23 – Processo di generazione di *k-gram*

L'idea base del sistema PIY è confrontare ogni sorgente con ogni altro usando la modalità del confronto a coppie.

Pulizia

Prima di tutto vengono rimossi commenti, spazi e tutte le porzioni di codice irrilevanti. Questa fase di *cleaning* è necessaria per difendersi dagli attacchi di Tipo 1 e per rendere più precisa la rappresentazione *k-gram* che verrà realizzata nelle fasi successive del procedimento.

Tokenizzazione

Dopo la pulizia, ogni codice viene convertito in una sequenza di token, cioè una piccola porzione di codice che corrisponde ad una particolare parola chiave o a una frase. L'efficacia del procedimento dipende dalla scelta del token set. Una volta che il sorgente è stato convertito in token, le fasi successive di PIY diventano completamente indipendenti dal linguaggio di programmazione usato.

Rappresentazione *k-gram*

Le stringhe di token sono rappresentate in *k-gram*. Per ogni sorgente, PIY memorizza i *k-gram* corrispondenti con la relativa frequenza.

Confronto a coppie

PIY è in grado di memorizzare moltissimi documenti visualizzati in precedenza. Per effettuare la comparazione, la lista delle frequenze dei *k-gram* del documento è considerata come un vettore in cui il *k-gram* rappresenta la posizione nel vettore e la relativa frequenza è il valore a quella posizione.

Per testare l'accuratezza di PIY prima di tutto si usa il punteggio F_1 , la misura che quantifica la combinazione di precisione e recupero, come definito in figura 24.

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2PR}{P + R}$$

Figura 24 – Definizione di precisione (*P*), recupero (*R*) e punteggio F_1

TP(True Positive) è il numero di documenti copiati che sono stati correttamente classificati come copiati dal documento originale; *FP(False Positive)* è il numero di documenti copiati che sono stati classificati come copiati da un altro documento diverso dall'originale; *FN(False Negative)* è il numero di documenti copiati che non sono stati classificati correttamente.
Non esiste un valore di k corretto in assoluto; per gli esperimenti legati alla presente ricerca, per il calcolo della similarità sono state usate la distanza Coseno con k=13 e la distanza Manhattan con k=4.

Confronto fra PIY e MOSS

MOSS viene eseguito lanciando uno script in Perl e inviando i documenti ad un server alla Stanford University. L'utente può specificare alcuni parametri come il linguaggio dei documenti e il numero di occorrenze di ogni passaggio prima che questo venga ignorato (di default il valore è 10).
Sia PIY che **MOSS** riportano le coppie di documenti più simili in ordine decrescente di similarità ma ci sono piccole differenze nell'output. Mentre PIY riporta la misura della non similarità tra due documenti, **MOSS** mostra il numero di linee in comune e la percentuale di similarità tra ogni documento e un altro.
In entrambe le due configurazioni di PIY di cui sopra, PIY sorpassa **MOSS** nel rilevamento di plagio, dal momento che, a differenza di **MOSS**, ottiene buoni risultati anche nel rilevare documenti che contengono diversi tipi di plagio. PIY risulta competitivo anche dal punto di vista dell'accuratezza.
Per ottimizzare il tempo di esecuzione di PIY si è implementata l'esecuzione in parallelo. Si è effettuato un test che ha previsto la memorizzazione di 180 documenti plagiati in database di varie dimensioni e la sincronizzazione dei confronti a coppie quando venivano eseguiti

sequenzialmente e poi quando venivano eseguiti in parallelo. L'esecuzione in parallelo ha determinato un significativo incremento della velocità di esecuzione.

Un'ulteriore ottimizzazione è stata ottenuta usando il clustering dei dati per ridurre il numero necessario di confronti a coppie. A tale scopo si sono testate due tecniche: DBSCAN (density-based spatial clustering of application with noise) e PAM (partitioning around medoids).

Il primo usa due parametri, *minPts* ed ε, per classificare tutti i punti in punti dato, bordo o rumore, rimuovendo poi quelli classificati come rumore. PAM invece rientra fra gli algoritmi di clustering classificati come K-medoid in cui ogni cluster è rappresentato dai suoi medoid (un medoid è il centro rappresentativo di un cluster che è anche un punto dati valido).

Per migliorare ulteriormente le prestazioni si propone un nuovo algoritmo che assegna dei punti indice ai cluster; di seguito lo pseudo codice che lo descrive:

```
For ogni cluster i
      For ogni punto p in i
            If p non è incluso nella distanza δ scelta per
                  ogni punto indice corrente di i
            p è un punto indice di i
```

DBSCAN ha ottenuto buone prestazioni usando la distanza di Manhattan per la similarità, mentre PAM ha lavorato meglio con la distanza Coseno. I tempi di confronto dopo il clustering sono stati molto peggiori con DBSCAN perciò si sceglie di adottare PAM come tecnica di clustering per PIY. Nonostante i risultati promettenti, il sistema PIY richiede ulteriori sperimentazioni.

CodeMatch

È stato sviluppato nel 2005 da Bob, membro dell'IEEE e presidente della Zeidman Consulting, un'azienda che si occupa di ricerca e sviluppo. **CodeMatch** usa la conoscenza dei linguaggi di programmazione e della struttura dei programmi per migliorare i risultati del matching. Usa una combinazione di cinque algoritmi per cercare il plagio: source line matching; comment line matching, word matching; partial word matching;

semantic sequence matching. Supporta BASIC, C, C++, C#, Delphi, Flash ActionScript, Java, Javascript, MASM, Pascal, Perl, PHP, PowerBuilder, Ruby, SQL, Verilog, VHDL.

CPD e CPDP

Copy Past Detector è una variante di PMD che usa l'algoritmo Karp-Rabin per cercare codice copiato. Supporta Java, JSP, C, C++, FORTRAN, PHP e C#. Il suo funzionamento prevede diversi step: tokenizzazione del codice; costruzione della tabella di occorrenze in base ai token; ricerca di duplicati nella tabella delle occorrenze.

Negli approcci basati sul testo ogni blocco di istruzioni nel codice sorgente è trattato come stringa e il programma è considerato una sequenza di stringhe. Due programmi sono considerati simili se le relative stringhe corrispondono.

Negli approcci che si basano su token ogni codice viene sottoposto a parsing per essere poi suddiviso in una serie differente di token che rappresentano identificatori, parole chiave, ignorando commenti, ecc. Successivamente le sequenze di token vengono confrontate usando un algoritmo di matching delle stringhe.

CPD (Copy Past Detector) è un tool *open source* di rilevamento del plagio sviluppato in Java che utilizza l'algoritmo RKR-GST.

Negli approcci basati su alberi, due programmi sono confrontati usando gli Abstract Syntax Tree (AST) che contengono informazioni strutturali sui programmi.

Ci sono anche altri approcci per il rilevamento del plagio come il grafo delle dipendenze del programma, PDG.

CPDP

CPDP (Code Plagiarism Detection Platform) è una piattaforma realizzata dall'Università di Bangalore in India che usa una intelligente tecnica di tokenizzazione e una versione ottimizzata di matching che migliora i valori di precisione e recupero.

Questo tool prevede prima di tutto la tokenizzazione di un set di programmi detto *corpus* e la conseguente memorizzazione delle sequenze di token in un indice. Per un dato progetto da verificare, dividiamo in token ogni

programma del progetto in input e cerchiamo l'indice al di sopra delle sequenze di token di questi programmi.

A questo punto utilizziamo l'algoritmo RKR-GST per confrontare i programmi in input con i possibili match e cerchiamo porzioni di codice copiato.

Per la tokenizzazione usiamo eclipse JDT per il parsing e otteniamo l'AST che può essere visitato per generare token per gli elementi del sorgente.

Il listato 1 mostra un semplice codice Java e il listato 2 è la corrispondente sequenza di token.

```
public class Test { //classe test
      public static void main(String []args){
      Integer k = (new java.util.Random())
            .nextInt(1000);
      for(int i=0; i<100; i++){
            k++;
            if((k%5) == 0)
                  System.out.println(k+"5 is 0!");
            }
      }
}
```

Listato 1 – Un semplice programma Java

```
hF
wEHNFGPQO
PG7NuFNOO5HNIO0
kNP7I0GXI0GSSO
GSS0
rNNGWIO77IO
F4G4HNGSJO0
```

Listato 2 – Sequenza di token che rappresenta il semplice programma in listato 1.

Indicizzazione

La sequenza di token risultante deve essere memorizzata in un indice per consentire il suo utilizzo per verificare un potenziale match con un dato file. Usiamo una tecnica *k-gram* per tradurre la sequenza di token in una collezione *k-gram*. Memorizziamo tutti i *k-gram* di ogni file come un indice del documento per file in un indice.

Confronto di file

Dato in input un set di programmi, suddividiamo in token ogni file in input e poi rappresentiamolo come una collezione *k-gram* come abbiamo fatto per i file del *corpus*. Per ogni file in input costruiamo una query indexer con questi *k-gram* per individuare il matching dei documenti dall'indice di cui sopra.

La piattaforma CPDP consiste in diverse componenti (figura 25):

- Tokenizer e Indexer;
- Memorizzatore dell'indice;
- Tokenizer e scanner;
- Rilevatore di plagio;
- Database orientato ai documenti.

Il sistema accetta inizialmente un set di programmi, li suddivide in token e li memorizza in sequenze *k-gram* in un indice basato su Apache Solr. Poi i file indicizzati vengono memorizzati in un database orientato ai documenti per ottimizzare la memorizzazione e la portabilità.

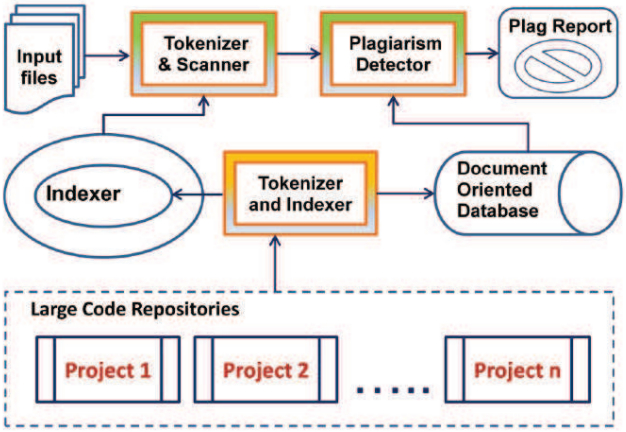

Figura 25 – Diagramma dell'architettura di CPDP

Il sistema è stato testato su un set di file di Java.

Ci sono due componenti usate per generare dati per i test:

- Random-File-Piker, che sceglie un numero di file in maniera random a partire da una serie di dati in input per poi modificarli con codice copiato;
- Plagiarism Injector, che si usa per simulare le azioni svolte durante il plagio: copia del codice, modifiche, ecc.

Calcolando la precisione (livello di accuratezza dei risultati) e il recupero (livello di completezza dei risultati) del tool **CPDP** si scopre che la precisione media è 11,5% e il suo recupero medio del 10,8%, molto di più di quelli di **CCFinder**, che è il tool con i valori più alti rispetto a quelli rappresentati nelle Figure 26 e 27. **CPDP** dimostra prestazioni migliori degli altri tool considerati (**CCFinder**, **CPD**, **Plaggie** e **Sherlock**), probabilmente grazie all'innovativa tecnica di tokenizzazione e alla tecnica di matching migliorata.

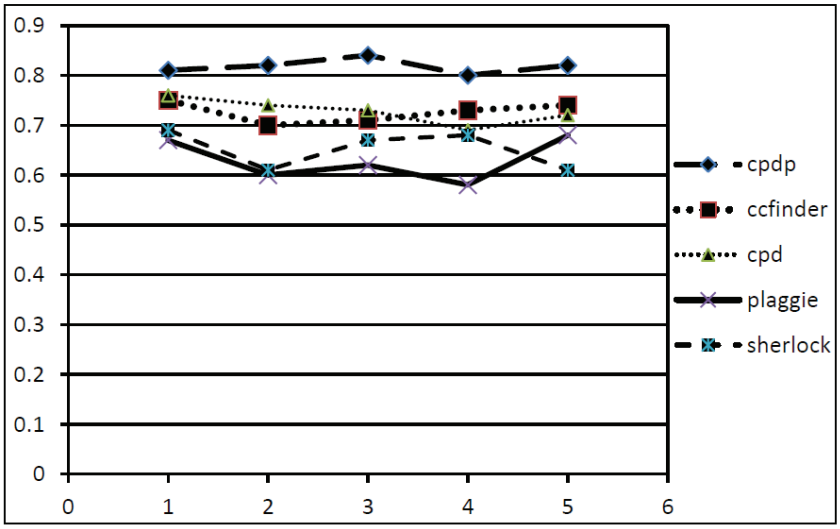

Figura 26 – Confronto dei valori di precisione

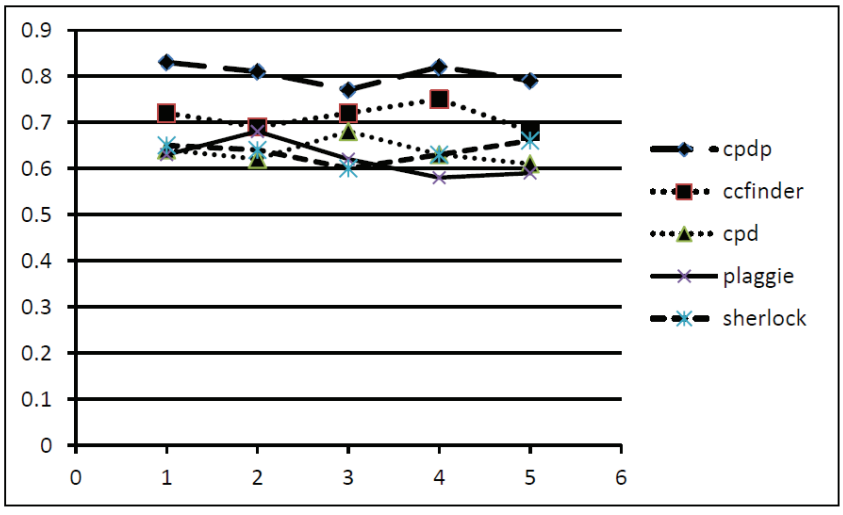

Figura 27 – Confronto dei valori di recupero

Plaggie

È stato sviluppato nel 2002 all'Università di Helsinki da Ahtiainen e altri. È stato studiato per la verifica di esercizi di programmazione in Java, molto simile a **JPlag** con la differenza che deve essere installato localmente e il suo codice sorgente è open. Supporta solo Java.

Plaggie è un tool standalone pensato per applicazioni Java che può essere usato per verificare esercizi di programmazione in Java.

JPlag o **MOSS** sono probabilmente i tool più utilizzati in ambito universitario ma **Plaggie** può essere molto più utile in alcune situazioni: a) quando il dipartimento non vuole correre il minimo rischio di divulgare per errore al di fuori dell'istituzione informazioni confidenziali in merito alla frode; b) se alcuni vogliono integrare **Plaggie** con altri software già in uso; c) se qualcuno vuole confrontare differenti tool di rilevamento in modo dettagliato.

Plaggie può analizzare programmi scritti in Java 1.5. È un tool *open source* con licenza GNU e può essere scaricato dalla pagina web della Università di Tecnologia di Helsinki. **Plaggie** è stato testato in Linux; teoricamente può essere installato anche su Windows ma non è mai stato testato.

Plaggie è stato il primo tool *open source* per il rilevemento del plagio; è simile a **JPlag** ma a differenza di **JPlag** viene eseguito come applicazione standalone.

Marble

È stato sviluppato nel 2002 per programmi java. È usato al Dipartimento di Informatica e Scienze dell'Informazione dell'Università di Utrecht per assistere i docenti nell'individuazione di plagio fra gli elaborati degli studenti. **Marble** supporta Java e si sta sperimentando anche il supporto per Perl, PHP e XSLT. Restituisce i risultati come file suspects.nf (elenco non ordinato) o suspects.nfs (elenco ordinato).

Parikshak

Al centro di ricerca e sviluppo C-DAC (Centre for Development of Advanced Computing) con sede a Mohali in India, nel 2012 è stato sviluppato un tool per rilevare il plagio in codici sorgente di studenti di

programmazione. Il tool, chiamato **Parikshak**, supporta sei linguaggi di programmazione: C, C++, Java, Perl, Python e Php. Lavora in tre fasi: tokenizzazione, rappresentazione n-gram dei sorgenti, confronto dei codici con l'algoritmo Greedy-String-Tiling.

Parikshak è un tool online che supporta diversi linguaggi di programmazione. Prevede un approccio basato sulla struttura ed è stato implementato pensando alla sua estendibilità: nuovi linguaggi di programmazione potranno essere aggiunti facilmente. Attualmente ne supporta sei: C, C++, Java, PHP, Python, Perl. Per accedere al servizio i docenti devono semplicemente loggarsi; non sono necessarie installazioni extra di software e l'interfaccia è molto intuitiva. L'output è una lista di file con la relativa percentuale di matching.

L'architettura del tool consiste in un'interfaccia web in cui i file selezionati vengono verificati e i risultati compaiono in output. La tokenizzazione è dipendente dal linguaggio mentre il confronto è indipendente. **Parikshak** è composto da tre moduli: *interfaccia web*, *tokenizer*, *comparatore*.

L'*interfaccia web* è ciò che l'utente vede quando invia la richiesta di verifica di un documento; il *tokenizer* è preposto alla suddivisione in token dei codici sorgente; i token sono inoltrati al *comparatore*, che li analizza per valutarne la similarità.

Il processo di tokenizzazione traduce il codice sorgente in una sequenza di token seguendo tre step: prima converte il codice in un formato lessicale, poi traduce il testo in token e infine sostituisce i token con numeri appropriati. Il codice si può dividere in diversi tipi di token: header, parole chiave, identificatori, operatori, numeri. La sequenza di token viene rappresentata con la tecnica *n-gram*, un modo di rappresentare i token raggruppandoli in gruppi di misura *n*, da cui il nome *n-gram*. Aumentando il valore di *n* le possibilità di trovare codice copiato potrebbero diminuire perché non siamo in grado di individuare similarità piccole; diminuendo troppo il valore i due codici dovrebbero essere totalmente differenti per evitare falsi positivi poiché verrebbero rilevate anche differenze molto piccole. Meglio impostare *n* a un valore pari a 4 o 5.

In **Parikshak** ogni sequenza di n numeri è detta coppia; la lista di coppie viene confrontata con l'algoritmo di Running-Karp-Rabin Greedy-String-Tiling.

Dopo il confronto, si estrae il numero di linee di ogni coppia e queste linee vengono marcate con uno stesso colore e con un apposito tag nel codice sorgente corrispondente.

A questo punto si calcolano il numero di linee di codice copiato in entrambi i file e il numero totale di righe nel codice sorgente originale per calcolare la percentuale di matching. Si usa il coefficiente di similarità Jaccard per trovare la percentuale di similarità:

$$\textit{coefficiente Jaccard di similarità (A, B)} = \frac{\textit{numero totale di frammenti comuni a entrambi i sorgenti}}{\textit{numero totale di frammenti distinti nei due sorgenti}}$$

Confronto con altri tool

Ogni tool ha i suoi pro e i suoi contro.

JPlag è vulnerabile al cambiamento dell'ordine delle linee di codice nel sorgente; in Parikshak anche se cambiamo l'ordine del codice è possibile rilevare il plagio.

MOSS considera i codici sorgente come file generici di testo, perciò vanno perdute molte informazioni strutturali che potrebbero essere usate per il matching. **Parikshak** classifica le informazioni suddividendole in categorie (ad esempio quella degli operatori) e in questo modo memorizza alcune informazioni strutturali utili.

Plaggie supporta solo Java, mentre **Parikshak** attualmente supporta sei linguaggi.

CodeMatch non offre una visualizzazione completa dei file sorgente mentre in **Parikshak** è possibile visualizzare tutti i codici sorgente in una volta. **Parikshak** è stato testato in un corso post laurea al CDAC di Mumbai e i risultati dimostrano che è un approccio molto efficace e semplice da utilizzare.

SIM

SIM analizza programmi scritti in Java, C, Pascal, Modula-2, Lisp, Miranda e può processare anche normali file di testo. Utilizza la tecnica di allineamento delle stringhe sviluppata per rilevare la similarità fra sequenze di DNA. I codici sorgente di **SIM** sono pubblici.

SIM è una diretta applicazione delle tecniche di allineamento delle stringhe sviluppate per rilevare la similarità fra sequenze di DNA. Dati due programmi, **SIM** prima di tutto li traduce nei corrispondenti alberi sintattici usando un analizzatore lessicale. Successivamente, visualizzando gli alberi sintattici come stringhe, **SIM** li allinea inserendo degli spazi in ognuno per ottenere la sottosequenza di token comune più lunga. Il grado di similarità considerato è un valore compreso fra 0.0 e 1.0. **SIM** viene eseguito in un tempo $O(s^2)$, dove s è la dimensione massima degli alberi.

L'allineamento di due stringhe s e t si ottiene inserendo degli spazi (cosiddetti simboli *gap*) in ogni stringa fino a fare in modo che abbiano la stessa lunghezza. Si noti che sono possibili diversi allineamenti.

A questo punto si verifica se i caratteri nella stessa posizione in ogni stringa corrispondono o no; si otterrà il punteggio m se corrispondono, d se non corrispondono e g se i caratteri sono simboli gap. Per ogni blocco di coppie di caratteri il punteggio è la somma dei punteggi individuali e il punteggio massimo ottenuto da un blocco di coppie rappresenta il punteggio dell'allineamento.

L'allineamento è adatto nel caso di match inesatto ed è molto usato nella biologia computazionale per studiare le relazioni tra sequenze di DNA.

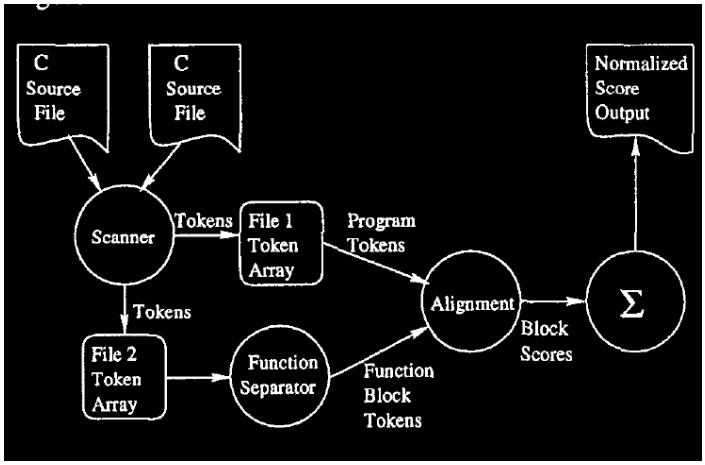

Figura 28 – Diagramma a blocchi del design di SIM

Ogni programma in input C passa prima attraverso un analizzatore lessicale che genera la forma compatta della sua struttura in termini di stream di interi chiamati token. L'analizzatore lessicale viene generato automaticamente dal comando Unix `flex` dato un appropriato subset della grammatica C, così SIM può essere modificato facilmente per lavorare con programmi scritti in

altri linguaggi. Ogni token rappresenta sia operazioni logiche che matematiche, simboli di punteggiatura, macro di C, parole chiave, costanti numeriche o di tipo stringa, commenti o identificatori.

Per esempio la linea di codice

```
for (i=0; i < max; i++)
```

può essere tradotta nella seguente sequenza di token:

TKN_FOR TKN_LPAREN TKN_ID_I TKN_EQUALS TKN_ZERO ...

Dopo che due codici sorgente sono stati suddivisi in token, lo stream di token del secondo viene diviso in sezioni, una per ogni modulo del programma originale. Ogni modulo poi viene allineato separatamente con lo stream di token del primo programma.

Un tool online per rilevare il plagio fra codici sorgente

Molte università cooperano per realizzare sistemi di rilevamento di plagio condivisi; in occasione dell'ICETA (International Conference of Emerging eLearning Technologies and Applications) che si è svolta in Slovacchia a ottobre 2013, è stato presentato un tool online molto efficace nel rilevare plagio all'interno di codici brevi.

Il sistema descritto ha utilizzato il tool SIM per rilevare il plagio, un software free col quale si dialoga da riga di comando e che può confrontare ricorsivamente numerose sottodirectory. Il sistema online viene eseguito su server Apache e l'applicazione lato server è stata realizzata usando il linguaggio PHP.

L'applicazione lato client gestisce la comunicazione fra utente e server; i file da verificare vengono caricati sul server mediante upload dall'interfaccia disponibile online (vedi figura 29) ed estratti in specifiche cartelle nel caso in cui siano stati caricati in formato zip.

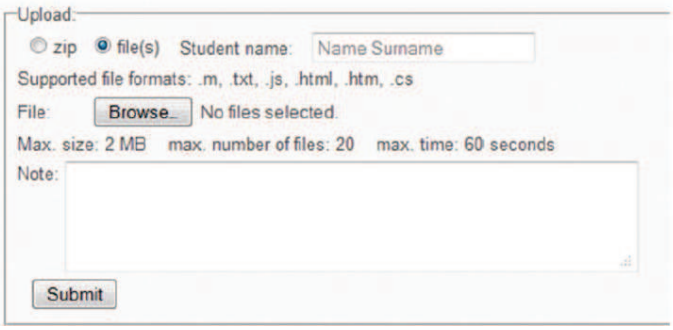

Figura 29 – Interfaccia grafica per l'uploading dei file

Successivamente parte il confronto dei file, dal quale potrà emergere se i file sono esattamente identici, cioè se l'originale è stato copiato senza apportare alcuna modifica (in questo caso parliamo di confronto veloce) oppure se hanno in comune solo alcune parti e in quest'ultimo caso sarà possibile anche visualizzare il dettaglio delle parti condivise (confronto approfondito), come nell'esempio in figura 30.

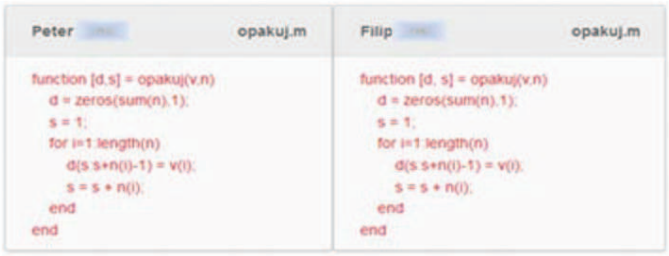

Figura 30 – Dettaglio del risultato del confronto approfondito. In questo caso i file coincidono al 100%; le parti in rosso sono quelle risultate completamente identiche

Ogni docente che usa questo software può aggiungere qui i propri commenti e condividere informazioni con altri colleghi.

Sherlock

Sherlock implementa un algoritmo simile al RKR-GST, usato da **JPlag** e **YAP3**. Uno dei pregi di Sherlock è che, a differenza di JPlag, i file non devono essere sottoposti a parsing per essere inclusi nel confronto e non occorrono parametri definiti dall'utente che possano influenzare le performance del sistema. Sherlock è un tool *open source* e la sua procedura di matching dei token è semplice da personalizzare per altri linguaggi diversi da Java. L'interfaccia utente di **Sherlock** mostra una lista di coppie di file simili e il loro grado di similarità, inoltre offre una visualizzazione veloce dei risultati sotto forma di grafo in cui ogni vertice rappresenta un singolo file sorgente e ogni arco mostra il grado di similarità tra due file.
Sherlock è un tool standalone e non è disponibile come servizio web, a differenza di **MOSS** e **JPlag**. Un tool standalone è preferibile per gli accademici al fine di verificare che i file degli studenti non contengano plagio, se si considera la questione della riservatezza.

PlaGate

Esiste un tool, chiamato **PlaGate**, che può essere integrato con i tool esistenti per migliorare le performance del rilevamento di plagio. **PlaGate** implementa anche un nuovo approccio per valutare la similarità tra file sorgente, usando la tecnica di raccolta di informazioni tramite LSA (Latent Semantic Analysis).

La similarità tra codici sorgente può esistere anche in assenza di plagio, magari perché gli studenti che li hanno realizzati si sono ispirati agli stessi esempi presentati a lezione.

La similarità tra due file è stabilita investigando sui frammenti di codice trovati all'interno dei file con caratteristiche comuni. È una procedura comune durante questa investigazione, mentre gli accademici confrontano due frammenti di codice simili alla ricerca di plagio, prendere in considerazione altri fattori che potrebbero avere causato il verificarsi di questa similarità.

Ecco di seguito elencati alcuni di questi fattori:

- i requisiti richiesti dai docenti (per esempio gli studenti potrebbero aver dovuto usare alcune strutture dati specifiche per realizzare un determinato elaborato);
- esempi di codice sorgente presentati in classe agli studenti (come base da cui partire per sviluppare i loro elaborati);
- la natura dei problemi di programmazione e dei linguaggi stessi (che spesso propongono un numero limitato di modi in cui la soluzione a un dato problema può essere tradotta in codice).

Possiamo avere due livelli di similarità tra file:

- Livello 0 – Innocente: i file analizzati non contengono frammenti di codice simile;
- Livello 1 – Sospetta: i due file hanno in comune frammenti di codice.

Ma l'esistenza di similarità non sempre corrisponde alla presenza di plagio. Per favorire la distinzione fra similarità *innocente* e *sospetta*, possiamo classificare i frammenti di codice in base al livello di contribuzione che hanno nel caratterizzare il codice che li contiene:

- Livello di contribuzione 0 – Nessun contributo: frammenti di codice non modificati da nessuno, di solito proposti dai docenti come scheletro o template del codice;
- Livello di contribuzione 1 – Bassa contribuzione: frammenti di codice modificati da tutti alla stessa maniera, probabilmente perché relativi a scheletri/template modificati in un secondo momento dai docenti;
- Livello di contribuzione 2 – Alta contribuzione: la similarità compare solo nei file oggetto di indagine e riguarda importanti funzionalità del programma.

Il funzionamento di PlaGate

Il tool mira ad accrescere i processi di individuazione del plagio e di investigazione. Include diversi componenti: PGDT, lo strumento per il rilevamento di file simili; PGQT, una query integrata con PGDT e con un tool esterno, utilizzato per il calcolo della similarità; PGLM, lo strumento per la gestione della lista dei risultati restituiti dal tool esterno.

Il tool esterno calcola la similarità e restituisce in output una lista delle coppie di file rilevate come simili. A questo punto PGLM memorizza i file nella lista principale che contiene tutti le coppie di file rilevate. Ognuna di queste coppie viene poi elaborata da PGQT che si occupa di gestire la visualizzazione della similarità fra i due file. La figura 31 illustra il funzionamento della procedura di rilevamento nel tool **PlaGate**.

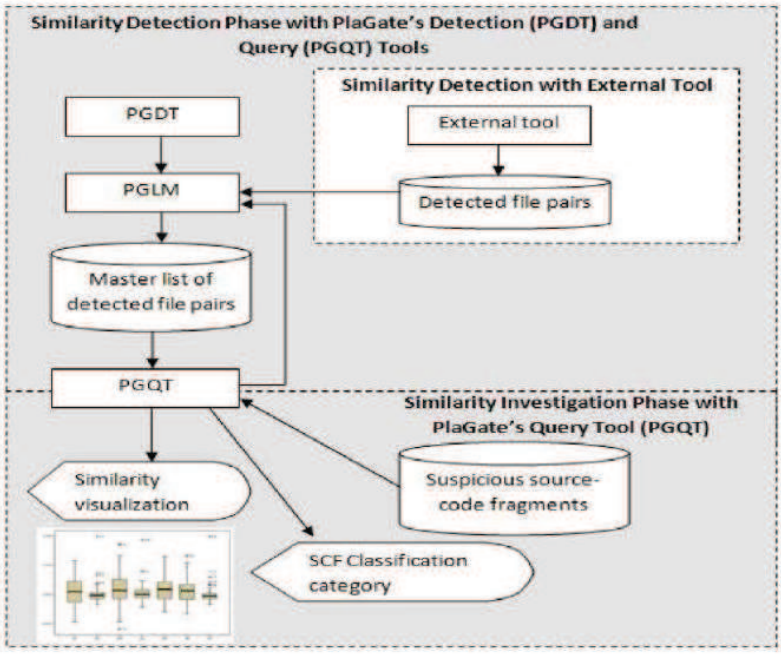

Figura 31 – Funzionalità di rilevamento e investigazione di PlaGate, integrata con un tool esistente di rilevamento del plagio

Il procedimento LSA in **PlaGate** segue diversi step:

- innanzitutto i file vengono preprocessati, per rimuovere i termini non necessari;
- LSA avvia il processo di trasformazione di ogni gruppo di file sorgente in una matrice $m \times n$ $A = [a_{ij}]$ nella quale ogni riga rappresenta un termine e ogni cella a_{ij} della matrice contiene la frequenza con cui il termine i appare nel file j;
- alla matrice A si applica un algoritmo per la pesatura dei termini;
- la matrice pesata A viene decomposta in tre matrici, U, $\sum$ e V;
- si riduce la dimensione delle matrici a k, selezionando le prime k colonne di ogni matrice e settando a 0 il valore delle altre.

Dopo aver applicato LSA ai codici sorgente, i componenti PGQT e PGDT possono essere utilizzati per rintracciare i file sospetti. Usando PGQT un file sospetto in input viene trasformato in un vettore query che viene poi proiettato in uno spazio k-dimensionale. La similarità fra il vettore proiettato e tutti gli altri file del codice sorgente viene poi calcolata come misura di similarità.

YAP3

YAP3 è un tool basato su matching delle stringhe, come **JPlag** e **Sherlock**. Converte i programmi in stringhe di token e le confronta usando l'algoritmo RKR-GST, per trovare frammenti di codice simili. Questo algoritmo è stato sviluppato principalmente per individuare interruzioni nelle funzioni del codice all'interno di funzioni multiple e rilevare il riordinamento di segmenti di codice sorgente indipendenti.

YAP3 è costituito da una fase di generazione, durante la quale viene creato un file di token per ogni codice sorgente. Segue la fase del confronto di ciascun file di token. Il risultato di ogni comparazione è un valore, definito come percentuale di matching, compreso fra 0 e 100.

Esistono tre diversi versioni del tool **YAP**, ciascuno dei quali utilizza un algoritmo diverso nella fase di comparazione: **YAP1** usa l'algoritmo LCS; **YAP2** l'algoritmo di Heckel; **YAP3** l'algoritmo di Running Karp-Rabin Greedy-String-Tiling.

EPLAG

La maggior parte degli approcci esistenti prevede l'uso della tecnica di conteggio degli attributi: un programma P può essere considerato come una sequenza di token caratterizzata da operatori e da operandi: ogni programma corrisponde cioè a una tupla di 4 elementi (*n1, n2, N1, N2*), in cui *n1* è il numero di operatori distinti, *n2* il numero di operandi distinti, *N1* il numero di occorrenze di operatori e *N2* il numero di occorrenze di operandi. Programmi che hanno la stessa tupla sono considerati sospetti. Mentre queste tecniche di conteggio degli attributi sono facili da implementare, non sono molto efficaci in termini di rilevamento del plagio.

Le tecniche basate sulla struttura del programma sono più robuste e sono alla base di molti tool di successo come **MOSS**, **YAP** in tutte le sue versioni e **JPlag**.

I sistemi che usano le tecniche di conteggio attributi e basate sulla struttura hanno due limiti principali:

- il sistema richiede un file sorgente da confrontare con molti file e i confronti possono richiedere tempi molto lunghi se i file contengono migliaia di righe di codice;

- c'è un forte bisogno di cluster di file piuttosto che di confronti uno-a-molti.

Una ricerca dell'Università di Bahawalpur ha combinato due tecniche che possono migliorare radicalmente le prestazioni degli strumenti di rilevamento di plagio fra codici sorgente. Il lavoro si è svolto in due fasi: 1) comparazione uno-a-molti di file di codice sorgente; 2) confronto del contenuto dei file con un'implementazione personalizzata dell'algoritmo GST.
La fase 1) mira a ridurre la quantità di dati usando strumenti di misurazione della similarità. Gli strumenti sperimentati a questo scopo sono molteplici:

- Complessità Ciclomatica di McCabe (due programmi sono simili se hanno la stessa complessità ciclomatica);
- Conteggio delle linee logiche (una linea logica indica un singolo blocco di istruzioni);
- Conteggio delle linee fisiche (una linea fisica è una linea del codice sorgente);
- Conteggio delle linee di commento;
- Conteggio delle linee bianche o vuote;
- Conteggio delle parole chiave;
- Conteggio dei campi;
- Conteggio delle funzioni.

La similarità fra file viene calcolata con la formula della distanza Euclidea, la distanza fra due punti data dalla formula pitagorica, come di seguito riportata:

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

dove p e q sono due punti nello spazio Euclideo di dimensione n, q_i e p_i sono gli assi dei due punti (o gli attributi di due oggetti) e $d(p,q)$ e $d(q,p)$ sono le distanza da p a q e da q a p rispettivamente.
Se gli attributi sono normalizzati (cioè i valori sono nel range 0-1) allora la similarità = (1 – distanza).
Definiamo il numero K e selezioniamo i file Top K che sono sospettati di essere copiati da un particolare file. In questo modo abbiamo ridotto a K files la quantità di dati da analizzare.

La fase 2) prevede l'applicazione del GST. Prima di tutto si effettua la tokenizzazione con la quale il file da verificare e il codice originale sono divisi in token; segue la fase di *tiling*, cioè di creazione dei *tile*. Un *tile* è un'associazione uno-a-uno tra una sottostringa di un codice sorgente e quella di un altro. Alcuni concetti importanti nel GST sono:

- la lunghezza minima di match (sottostringhe con una lunghezza al di sotto di tale numero vengono ignorate);
- la lunghezza massima di match (simile a tile ma è una corrispondenza temporanea, è la lunghezza massima della sottostringa comune ai due codici in una particolare iterazione);
- la formula *diceScore* (usata per quantificare i punteggi di plagio), ovvero:

$$diceScore(sFile, tFile) = \frac{2 * \sum_{i \, \varepsilon \, tiles} (lenght_i)}{|sourceFileSize| + |targetFileSize|}$$

La formula *diceScore* misura la similarità delle porzioni di sequenza di token dei due file che sono risultate coperte da match. Consideriamo ad esempio i due file File1 e File2:

File1	*int i; static double j;*
File2	*static double j; int i;*

Applicando la *diceScore* otteniamo *diceScore(sFile, tFile) = 2 * (3+4) / 7+7 = 1* e questa equazione ci mostra che i due file risultano identici anche se la sequenza dei token è differente.

EPLAG è un prototipo implementato per due tipi di utente: i docenti e gli studenti.

Il docente: si registra nel sistema; assegna le esercitazioni agli studenti del suo corso; visualizza le esercitazioni consegnate dagli studenti; può selezionare un particolare elaborato e testarlo per rilevare eventuale plagio; visualizza i risultati finali.

Lo studente: si registra nel sistema; seleziona un corso e presenta una soluzione all'esercitazione assegnata. Il flusso del sistema **EPLAG** è illustrato esaustivamente nella figura 32.

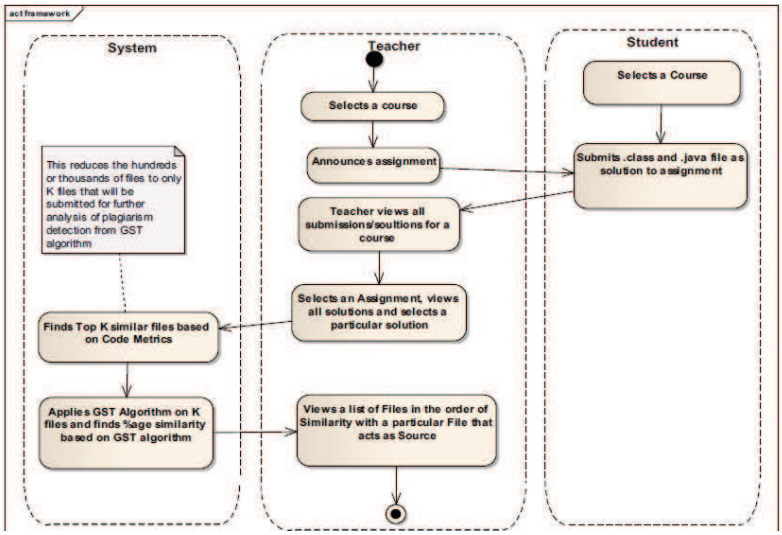

Figura 32 – Flusso del sistema EPLAG

CCS

Ad oggi può analizzare codici sorgente scritti in 3 linguaggi differenti: C, C++ e Java. I vantaggi di **CCS** sono molteplici: converte la struttura di memorizzazione dell'albero in una lista lineare per incrementare la velocità di sondaggio; raggruppa le informazioni relative all'albero sintattico in base al numero di sotto nodi per limitare le operazioni di comparazione non necessarie.

Usando **CCS** possiamo rilevare plagio come copia di un blocco, ridenominazione di funzioni, classi o variabili, aggiunta di dichiarazioni o parametri, ecc. In più **CCS** è molto preciso nelle operazioni come sottrazioni e divisioni.

Per testare l'accuratezza di **CCS** sono stati realizzati una serie di esperimenti usando codici sorgente presi da software *open source*. In particolare sono stati verificati i casi di cambiamento dell'ordine degli switch, delle funzioni, dei parametri e delle variabili e il caso di aggiunta della dichiarazione `typedef`. I risultati mostrano che **CCS** può individuare molto efficacemente questi tipi di plagio.

Consideriamo una coppia di codici Codice 2 e Codice 3. Il Codice 2 è un codice *open source* estratto dall'ambiente Puredata, dedicato alla programmazione e alla condivisione di audio, video e grafica. Il Codice 3 è stato ottenuto modificando il 2. Nelle figure 33 e 34 sono rappresentati i

risultati del confronto effettuato applicando rispettivamente **CCFinder** e **CCS** sui due codici.

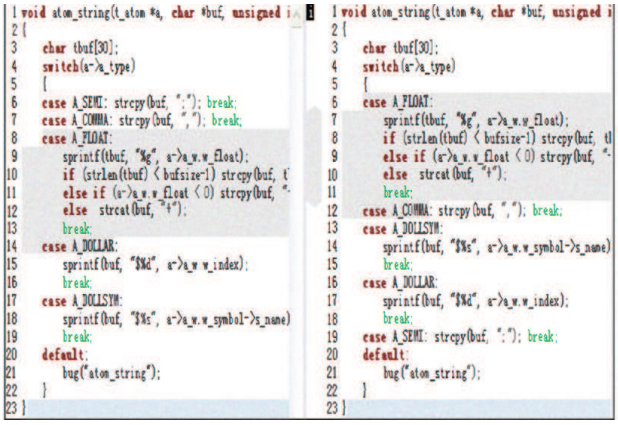

Figura 33 – Risultati dell'applicazione di CCFinder ai Codici 2 e 3; il codice simile è evidenziato dal background scuro

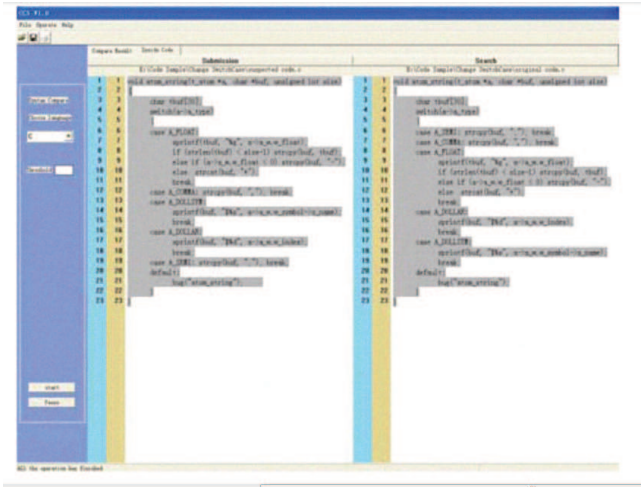

Figura 34 – Risultati dell'applicazione di CCS ai Codici 2 e 3; il codice simile è evidenziato dal background scuro

Confrontando le figure 33 e 34 possiamo notare che **CCFinder** può individuare solo i blocchi di codice esattamente identici anziché l'intero plagio. **CCS** invece si è comportato meglio in questo caso. Nella tabella 3 vediamo un confronto fra le capacità di rilevazione dei casi di plagio da parte di **CCFinder** e di **CCS**.

Types of Plagiarism Case	The Number of Plagiarism Case	The Number of the Detected out Plagiarism Case	
		CCFinder	CCS
Breaking the sequences of switch cases statements	12	0	12
Breaking the sequences of function definition statements	4	0	4
Breaking the sequences of function parameters	18	7	17
Breaking the sequences of variable definition statements	16	6	16
Adding "typedef" statement	15	0	13
Replacing the variable names	25	25	25
Replacing the function names	10	10	10
Total Number of each Column	**100**	**48**	**97**

Tabella 3 – Statistiche dei risultati di rilevazione del plagio in diversi casi usando CCFinder e CCS

Se la struttura delle dichiarazioni modificate è rimasta la stessa, anche **CCFinder** rileva il plagio efficacemente ma se la struttura dei codici confrontati è cambiata **CCS** lavora meglio.

CCS inoltre può limitare il numero di falsi positivi.

Consideriamo i Codici 4 e 5, questa volta in ambiente Microsoft XP. Il Codice 5 è stato ottenuto copiando due volte il 4 e cambiando posizione al sottraendo e al minuendo in una delle sottrazioni.

Codice 4

```
1 void fun1()
2 {
3       int x = 10, Y = 5, z = 3;
4       int m;
5       m = x - y*z;
6       printf("%d" ,m);
7 }
```

Codice 5

```
1 void fun2()
2 {
3     int x = 10, y = 5, z = 3;
4  int m;
5       m = y*z - x;
6       printf("%d" ,m);
7 }
8
9 void fun3()
10 {
11     int x = lO, Y = 5, z = 3;
12     int m;
13      m= x - y*z;
14     printf("%d",m);
15 }
```

Nella figura 35 sono evidenziati i risultati dell'applicazione di **CCS** ai due codici. La `fun2()` nel Codice 5 (rappresentata a sinistra della figura) non è simile alla `fun2()` del Codice 4 dal punto di vista semantico poiché sono state cambiate le posizioni di minuendo e sottraendo nella sottrazione. **CCS** può evitare falsi positivi adottando un trattamento speciale per le operazioni matematiche.

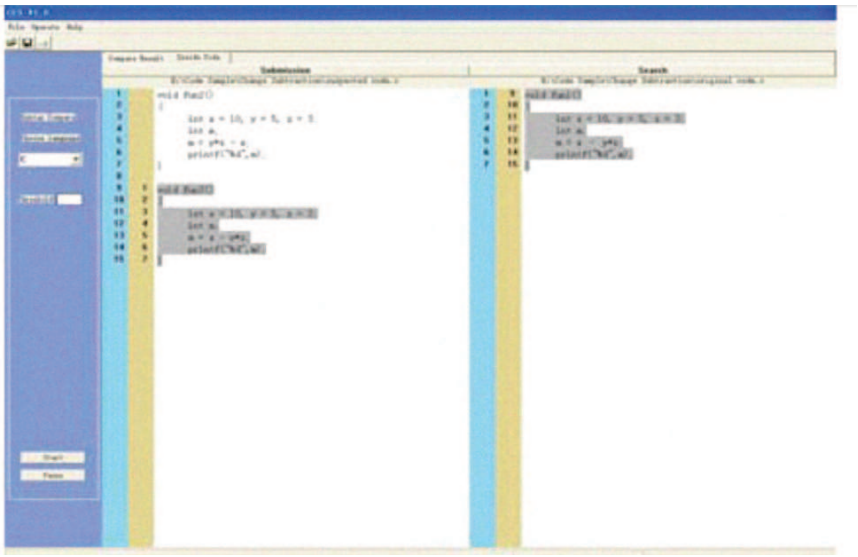

Figura 35 – Risultati dell'applicazione di CCS ai Codici 4 e 5. Il codice con il background scuro si considera plagiato

Un tool per il riconoscimento basato su AST

Numerose ricerche dimostrano che una parte significativa dei software è costituita da codice clonato. Fortunatamente sono state proposte numerose tecniche per l'individuazione di codice clonato, come quella basata sulla generazione di alberi sintattici che cerca sintatticamente cloni significativi ma tende ad essere molto pesante, poiché richiede un parser completo e un metodo di comparazione per i sottoalberi.

Il metodo proposto è implementato come tool di java; la sua architettura è mostrata in figura 36. Lo strumento sviluppato inizialmente analizza il codice sorgente ricevuto in input e identifica i vari metodi presenti. Ogni metodo viene classificato con l'aiuto dei valori di metrica e vengono individuate le potenziali coppie di codice clonato.

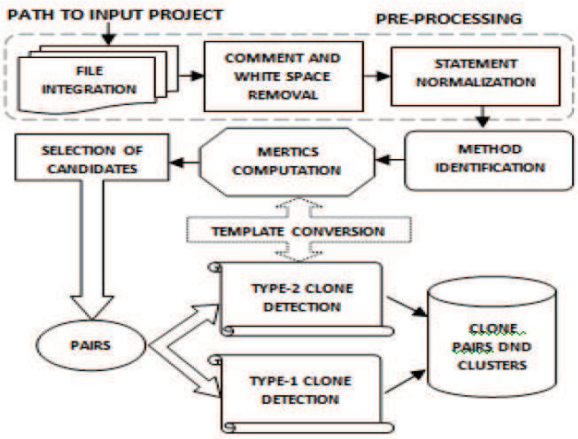

Figura 36 – Architettura del tool sviluppato

Le coppie che appaiono simili nel confronto testuale sono codificate come clonate. Il tool di rilevazione sviluppato è più leggero perché per esempio non impiega parser esterni e richiede un confronto meno approfondito di altri metodi.

Il processo di rilevazione di cloni prevede diverse fasi, di seguito descritte.

Input e preprocessing

Questa fase prevede l'integrazione del file, la standardizzazione del codice sorgente e la normalizzazione. L'integrazione del file prevede la concatenazione di tutti i file dello stesso progetto in un unico file per un parsing più efficace. Il file integrato viene poi parserizzato per poter rimuovere i commenti, gli spazi e le dichiarazioni al preprocessore. Il codice sorgente è ristrutturato in un formato standard che è importante per stabilire la similarità dei frammenti clonati.

Conversione del template

La conversione del template non è altro che la trasformazione del codice sorgente ricevuto in input in un set predefinito di dichiarazioni. La trasformazione consiste per esempio nel cambiare i tipi di dato, rinominare le variabili, le funzioni, ecc. come illustrato in figura 37.

Questo tipo di formato è detto *template* ed è usato nella comparazione testuale fra le coppie individuate durante l'applicazione del metodo di individuazione del plagio di tipo 2.

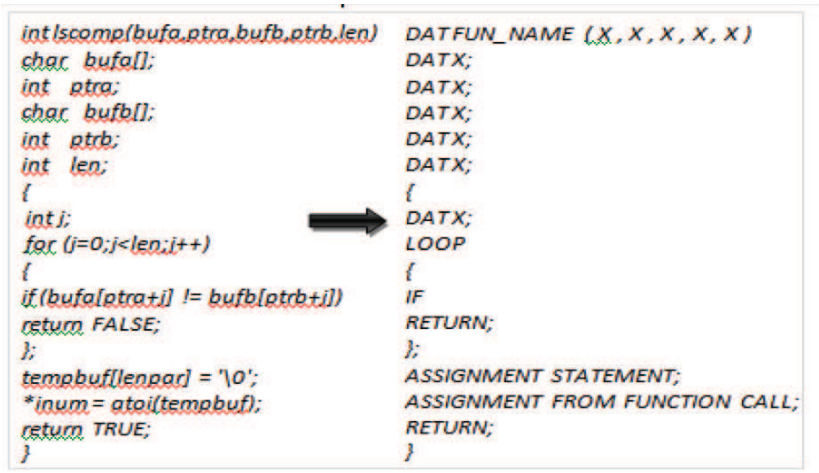

Figura 37 – Esempio di metodo modificato dopo la conversione del template

Identificazione del metodo

Una volta standardizzato, il codice sorgente viene scannerizzato per il match con gli altri metodi presenti nel file adottando un approccio *island-driven parsing*. La definizione della funzione viene estratta e raccolta attraverso un parser poi salvata per ulteriori riferimenti. Vengono annotati anche la posizione del comando `end` nella funzione e il numero di linee presenti in ogni metodo.

Computazione della metrica

Vengono usate sette metriche a livello di metodo per individuare metodi clonati di tipo 1 e 2. Sono le seguenti:

- Numero di linee di codice in ogni metodo;
- Numero di argomenti passati al metodo;
- Numero di funzioni chiamate in ogni metodo;
- Numero di istruzioni condizionali in ogni metodo;
- Numero di cicli in ogni metodo;
- Numero di dichiarazioni di return in ogni metodo

Queste metriche vengono calcolate per ciascuno dei metodi identificati e i valori sono memorizzati in un database. La tabella 4 mostra i diversi valori di metrica calcolati per il frammento di codice in figura 37. Le statistiche dei valori ottenuti per i vari metodi sono descritte nella tabella 5. Una volta calcolati i valori di metrica, le coppie di metodi con un set di valori uguale o simile verranno confrontate per verificare l'effettiva presenza di plagio.

Sl. No.	Metrics	Values
1.	No. of lines of code	17
2.	No. of arguments passed	5
3.	No. of local variables declared	6
4.	No. of function calls	1
5.	No. of conditional statements	1
6.	No. of looping statements	1
7.	No. of return statements	2

Tabella 4 – Valori di metrica calcolati per il metodo in figura 37

	No. of Lines	No. of Args	No. of Fn. Calls	No. of Conditions	No. of Loops	No. of Ret Stmts	No. of Local Var
No. of '0's	0	44	9	31	54	38	1
Min	4	0	0	0	0	0	0
Max	386	10	159	65	27	29	45
Mean	88.05	2.61	32.51	11.96	5.02	1.23	13.46
Median	37	2	12	4	1	1	7
25%	17	0	6	0	0	0	5
75%	145	5	61	16	6	1	23
Std. Dev	98.07	2.48	38.60	16.11	8.14	2.74	13.45
Var	9618.19	6.19	1489.79	259.58	66.27	7.50	181.04

Tabella 5 – Statistiche descrittive dei valori di metrica ottenuti

Il tool è stato testato con un progetto in C di media dimensione. Il programma, chiamato **Wetlab**, comprende circa 11.000 linee. Su di esso sono stati ricercati metodi clonati di tipo 1 e 2 e i risultati sono stati confrontati con due tool esistenti.

Il primo si basa su un metodo di individuazione basato su linguaggio Phoenix, una tecnica di individuazione automatica di codice clonato sviluppata come plugin del framework Phoenix di Microsoft che individua i metodi clonati utilizzando gli alberi sintattici e gli alberi dei suffissi. Il secondo è NICAD, un linguaggio specifico basato su parser che impiega TXL e semplice confronto di linee di testo per individuare funzioni clonate.

I risultati ottenuti con l'utilizzo di questi tool confrontati con il metodo descritto nel nostro paper sono rappresentati nella tabella 6 mentre in figura

38 indichiamo le percentuali di recupero (completezza) e precisione (esattezza) raggiunte con i 3 metodi.

Type	Phoenix-based			Nicad			Proposed method		
	FN	*CP*	*CC*	*FN*	*CP*	*CC*	*FN*	*CP*	*CC*
Type-1	123	21	4	123	27	8	123	27	8
Type-2		-	-		98	18		98	18

Tabella 6 – Clone Pairs e Clone Cluster per Wetlab

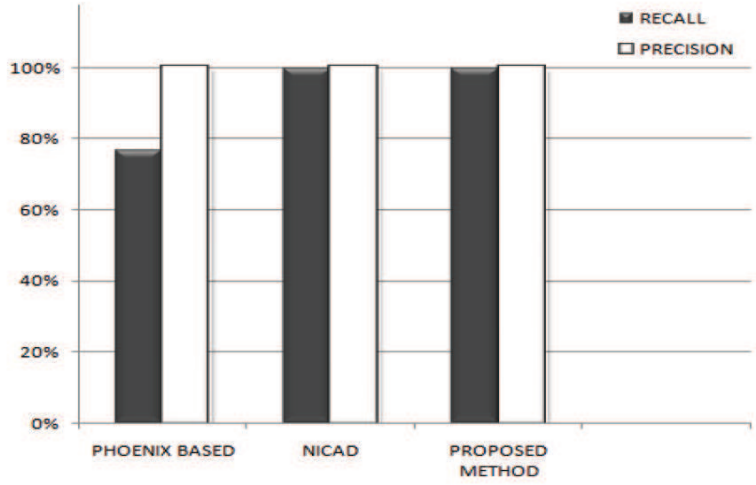

Figura 38 – Recupero e precisione con Wetlab

Nel caso di **Wetlab** il tool basato su Phoenix riporta l'esatto match fra i cloni raggruppati nelle 6 classi ammettendo differenze solo nei nomi delle funzioni e nei tipi di dato; lo strumento proposto invece ha ottenuto 8 match esatti su 27 coppie di cloni usando le tecniche di standardizzazione e normalizzazione.

NICAD riporta 8 match esatti per il tipo 1 e 18 cloni simili di tipo 2. NICAD prometteva di raggiungere il 100% di accuratezza nei risultati ma anche il metodo proposto ha dimostrato di poter soddisfare questa aspettativa. Anche se NICAD ha individuato efficacemente i metodi clonati, la fase di estrazione ha impiegato un parser esterno TXL, mentre il metodo proposto ha usato solo un parser incorporato e nessuno strumento esterno.

Con l'uso delle metriche il tasso esponenziale di complessità esistente nella comparazione delle varie funzioni è stato evitato, migliorando la precisione.

Dal matching delle stringhe e dalla comparazione testuale sui metodi selezionati, si può ottenere un aumento del recupero.

Il tool Deimos

Il plagio è una forma di disonestà accademica della quale molto spesso gli studenti dei corsi di programmazione si macchiano. Kustanto e Liem, della Scuola di Ingegneria Elettronica ed Informatica di Bandung (Indonesia) hanno sviluppato un prototipo, **Deimos**, dedicato proprio al rilevamento di plagio nel codice sorgente.

Deimos lavora in due fasi: nella prima trasforma il codice sorgente in token; nella seconda confronta ogni coppia di token ottenuta usando l'algoritmo Running Karp-Rabin Greedy-String-Tiling.

Il plagio del codice sorgente si differenzia dagli altri tipi di plagio. Facile da eseguire, è difficile da individuare anche perché spesso è il risultato della combinazione di tante porzioni di codice plagiato perciò riconoscere le fonti può essere estremamente complesso. Le modifiche possibili sul codice sorgente sono di tipo lessicale oppure strutturale. Una modifica lessicale è un semplice cambiamento nel testo effettuato con un qualunque text editor (aggiunta/rimozione di commenti, modifica del nome di una variabile, ecc.) ed è molto facile da rilevare. Per effettuare modifiche strutturali invece è necessaria una certa conoscenza della programmazione; sono modifiche strutturali quelle che interessano le iterazioni, i cicli, le istruzioni interne alle funzioni, le dichiarazioni.

Ai primi anni di università sono iscritti moltissimi studenti e hanno un'alta frequenza di esercitazioni da svolgere, perciò in questa situazione è complicato identificare i casi di plagio manualmente.

Esistono diversi tool per il rilevamento che si basano su metodi strutturali come **YAP** nelle sue diverse versioni e **JPlag**.

YAP3 e **JPlag** usano lo stesso algoritmo per il confronto. L'algoritmo di Running Karp-Rabin Greedy-String-Tiling cerca il matching delle sotto stringhe di due token di stringhe massimizzando il numero di token rilevati. Riceve come parametro in input la lunghezza minima di matching che può essere usata per definire la sensibilità del rilevamento. Tutte le sottostringhe con una lunghezza di matching inferiore a quella minima saranno ignorate. La complessità computazionale è ridotta dall'applicazione della tecnica

Karp-Rabin di hashing: si calcolano i valori hash di ogni sottostringa e poi si confrontano; a valori hash simili è possibile che corrispondano coppie di sottostringhe simili. Nel caso peggiore la complessità dell'algoritmo RKR-GSTè $O(n^3)$ anche se nella realtà, grazie alla tecnica hashing Karp-Rabin la complessità è quasi sempre compresa tra $O(n)$ e $O(n^2)$.

Deimos è stato testato per rilevare il plagio in codici scritti in due linguaggi di programmazione con paradigmi di programmazione diversi come LISP e Pascal. **Deimos** presenta questi obiettivi principali: rilevare il plagio; mostrare i risultati del rilevamento in forma leggibile; cancellare i risultati.

Come la maggior parte dei tool di rilevamento di plagio anche Deimos prevede una fase di tokenizzazione e una di comparazione. Per la comparazione **Deimos** utilizza l'algoritmo RKR-GST. Per velocizzare l'esecuzione dell'hashing, ogni token viene rappresentato come un intero. Se i token sono rappresentati come stringhe, i valori hash devono essere calcolati per primi. Per rendere il programma più accessibile gli autori hanno pensato ad un'interfaccia web, che memorizzi in un database i risultati dei rilevamenti per poterli usare nei confronti successivi. Per ottenere buoni risultati dall'applicazione dell'algoritmo RKR-GST è importante scegliere accuratamente il valore della lunghezza minima di matching: se è settata a un valore troppo alto, i segmenti simili più brevi saranno ignorati e questo potrebbe evidenziare una similarità accidentale. Per scegliere il valore più corretto è bene tenere in considerazione la lunghezza media dei codici sorgente e il livello di difficoltà dell'esercitazione di programmazione assegnata agli studenti.

Deimos comprende due applicazioni. La prima è un'applicazione web, alla quale l'utente accede dal proprio browser, la seconda è un'applicazione backend, che si occupa di generare le risposte richieste dagli utenti attraverso l'interfaccia web.

La figura 39 mostra l'architettura del sistema **Deimos**.

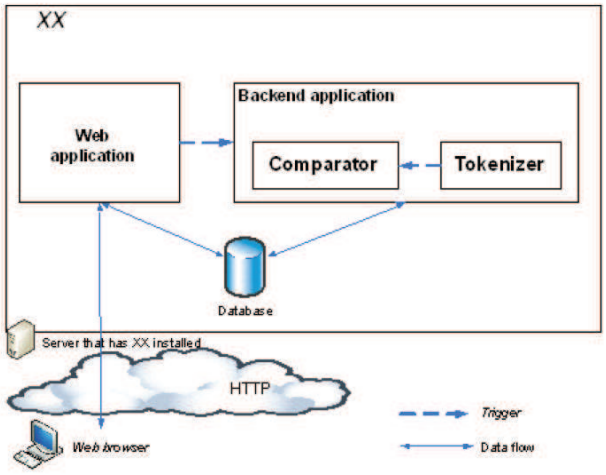

Figura 39 – Architettura del tool Deimos

L'applicazione web è stata implementata con la programmazione object-oriented come PHP, mentre per la parte backend è stato utilizzato Java. Il tokenizer legge un set di codici sorgente dalla directory in input; ogni codice viene scansionato e parserizzato mentre la rappresentazione dei segmenti di codice tramite `integer` viene memorizzata in un database. A questo punto il comparatore confronta i token di stringhe per ogni coppia di codici sorgente. Per ogni comparazione viene calcolato e memorizzato nel database il valore di similarità.

Per calcolare la similarità Deimos utilizza la seguente formula:

$$Sim=t\_tiled/min\_prg\_lenght*100\%$$

dove *Sim* è il valore percentuale di similarità per ogni coppia, *t_tiled* è il numero di token simili nella coppia ottenuto applicando l'algoritmo RKR-GST e *min_prg_lenght* è il numero di token nel programma con lunghezza minore nella coppia.

Poiché gli studenti dei corsi di programmazione lavorano sugli stessi semplici problemi, è normale che alcune parti dei loro codici siano molto simili. Per programmi semplici in Pascal è raccomandata una lunghezza minima di matching compresa fra 3 e 5; per programmi in LISP meglio un valore intorno a 10. Deimos può processare un centinaio di codici sorgenti in LISP in 12 minuti; per rilevare il plagio in un centinaio di codici in Pascal sono sufficienti 2 minuti.

CloneDetective

Molti dei tool di rilevazione di plagio usati nella ricerca e nella pratica sono solo prototipi e software privati. I pochi strumenti di rilevamento *open source* che rimangono non sono molto estensibili. Questo riduce notevolmente la loro applicazione e costringe i ricercatori a creare nuovi tool.

Per alleviare il problema, alcuni ricercatori dell'Institut Für Informatik dell'Università di Monaco hanno realizzato **CloneDetective**, una sorta di banco di lavoro per la ricerca nel campo della rilevazione dei cloni con enormi capacità di estensione. Questo tool è stato applicato con successo non solo nell'ambito della ricerca ma anche in diversi progetti commerciali.

Configurabilità ed estensibilità

Il rilevamento dei cloni comprende diverse fasi; innanzitutto la fase di input e preprocessing, durante la quale viene letto il codice e si crea una rappresentazione del programma. Durante il rilevamento vengono identificati frammenti di codice simili. Successivamente i risultati sono processati e presentati all'utente in vari formati nella fase di output.

Rilevamento di cloni di codice

Input. I file sorgenti in input vengono trasformati in una sequenza di unità di programma normalizzate. CloneDetective offre in input al processore linee indipendenti dal linguaggio. La normalizzazione di base può essere specificata usando espressioni regolari.

Rilevamento. La fase di rilevamento ricerca sottosequenze simili nelle unità di programma create nella fase di input. Per effettuare questa ricerca sono attualmente disponibili due algoritmi differenti. Entrambi lavorano su un albero di suffissi creato dalle unità di programma. Il rilevamento *Ungapped* ricerca sotto sequenze identiche così rileva frammenti di codice che differiscono solo per gli spazi, i nomi degli identificatori, i commenti. L'individuazione avviene semplicemente attraversando l'albero dei suffissi e viene eseguita in tempo e spazio lineari. Il rilevamento *Gapped* rileva i cloni con modifiche, aggiunte o rimozioni nelle dichiarazioni. Le prestazioni di questi algoritmi dipendono dai sistemi analizzati.

Post processing. Questo step lavora sui risultati ricevuti dalla fase precedente utilizzando filtri e diverse computazioni metriche. La copertura dei cloni esprime il rapporto fra il sistema affetto da clonazione e così stima la probabilità che un cambiamento in una qualunque dichiarazione del programma richieda di essere effettuato in più di un luogo. Il numero di occorrenze di una dichiarazione esprime il numero medio di volte in cui una dichiarazione viene clonata.

Output. In output il processore prepara i risultati perché possano essere usati dagli utenti o da altri tool. Può essere scritto un report XML perché lo utilizzino altri tool oppure i risultati possono essere restituiti anche in HTML.

Ecosistema. Esistono vari visualizzatori che implementano la sintassi per tutti i linguaggi supportati e ci sono anche plugin per Eclipse e per Visual Studio.NET che supportano il rilevamento di cloni all'interno dell'IDE e inviano notifiche agli sviluppatori quando hanno manipolato del codice clonato.

CloneDetective implementa il primo algoritmo di rilevamento dei cloni per linguaggi di modellazione basati su grafi come Matlab e parti della struttura sono già state utilizzate da numerosi ricercatori come base per migliorare i modelli rilevatori di cloni.

Confronti fra tool

Tipicamente i tool per l'individuazione del plagio restituiscono la misura della similarità per ogni coppia di programmi ed è necessario l'intervento umano per decidere se la similarità è dovuta a plagio oppure no.

Molti approcci partono dalla trasformazione del codice sorgente in una sequenza di token, in cui i token sono entità lessicali che costituiscono il programma (ad esempio parole chiave o identificatori).

Per calcolare la similarità alcuni approcci si basano sugli attributi e rappresentano i sorgenti come sequenze di numeri; altri invece analizzano la struttura dei programmi e utilizzano diversi algoritmi per il confronto delle sequenze di token.

Per confrontare i tool si valutano le caratteristiche e le performance di ognuno. Il confronto delle caratteristiche è di tipo qualitativo (si valutano le proprietà dei tool, come l'algoritmo usato per i confronti, i linguaggi

supportati, ecc.) mentre quello delle performance è quantitativo (si analizzano i risultati dei test effettuati).

Il confronto qualitativo valuta i seguenti criteri:

1. linguaggi supportati;
2. espandibilità (la capacità di adattarsi a linguaggi di programmazione non supportati);
3. presentazione dei risultati (una buona presentazione dei risultati dovrebbe contenere almeno questi dati: sommario, elenco dei match in ordine di similarità, presenza di un editor per effettuare ulteriori confronti fra i risultati ottenuti);
4. semplicità d'uso del tool;
5. esclusione di eventuale codice template dal confronto;
6. esclusione di file molto piccoli;
7. confronto storico (possibilità di confrontare un nuovo set di elaborati con vecchie versioni degli stessi elaborati);
8. valutazione (per file o per elaborato: quando un elaborato comprende più file è importante che l'individuazione del plagio avvenga per ognuno dei file incluso nell'elaborato);
9. utilizzo (come servizio web o in locale);
10. *open source* (se il tool è *open source* sarà possibile estenderlo o migliorarlo a seconda della situazione in cui si desidera utilizzarlo).

Confronto delle caratteristiche

JPlag

JPlag converte i programmi in sequenze di token che ne rappresentano la struttura e poi utilizza l'algoritmo Greedy-String-Tiling per rilevare la similarità fra i codici.

1. linguaggi supportati: Java, C#, C, C ++, Scheme e linguaggio naturale;
2. espandibilità: non valutabile;
3. presentazione dei risultati: pagine HTML. Una caratteristica particolare di JPlag è il clustering delle coppie, che permette di vedere facilmente se un elaborato è simile ad altri;
4. semplicità d'uso del tool: molto facile da usare;
5. esclusione di eventuale codice template dal confronto: è possibile;

6. esclusione di file molto piccoli: sì;
7. confronto storico: no;
8. valutazione: per elaborato;
9. utilizzo: servizio web;
10. open source: no.

Marble

Usa un approccio basato su struttura. Per ogni file calcola due versioni normalizzate: una in cui l'ordine di campi, metodi e classi interne è esattamente identico a quello del codice originale e un'altra in cui campi, metodi e classi interne sono suddivisi in gruppi ordinati.

1. linguaggi supportati: Java e C#. Si sta studiando il supporto per Perl, PHP e XSLT;
2. espandibilità: l'unica parte di codice dipendente dal linguaggio è la normalizzazione, facilmente adattabile;
3. presentazione dei risultati: uno script chiamato suspects.nf (risultati non ordinati) oppure suspects.nfs (ordinato), che può essere aperto con qualsiasi editor di testo;
4. semplicità d'uso del tool: è disponibile come script di Perl ed ha un'interfaccia a linea di comando;
5. esclusione di eventuale codice template dal confronto: non è possibile;
6. esclusione di file molto piccoli: sì;
7. confronto storico: sì, se i risultati precedenti sono stati memorizzati in un file system opportunamente ordinato;
8. valutazione: per coppie di file;
9. utilizzo: in locale;
10. open source: no.

MOSS

Usa un approccio basato su *fingerprinting* e l'algoritmo di *winnowing* per il calcolo della similarità.

1. linguaggi supportati: C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086, HCL2;

2. espandibilità: sì;
3. presentazione dei risultati: pagine HTML accessibili tramite web;
4. semplicità d'uso del tool: al momento della registrazione su MOSS viene inviato uno script che può essere usato per caricare gli elaborati;
5. esclusione di eventuale codice template dal confronto: è possibile;
6. esclusione di file molto piccoli: sì;
7. confronto storico: sì;
8. valutazione: per elaborato;
9. utilizzo: servizio web;
10. open source: no.

Plaggie

Nato per il rilevamento di plagio in esercizi di Java, è simile a **JPlag**, tranne che per il fatto di lavorare in locale e di avere un codice sorgente open. Come **JPlag**, **Plaggie** utilizza il Greedy-String-Tiling per il confronto.

1. linguaggi supportati: Java 1.5;
2. espandibilità: apparentemente no;
3. presentazione dei risultati: pagine HTML con tabelle che mostrano le statistiche dei risultati;
4. semplicità d'uso del tool: per configurarlo è sufficiente salvare i file di configurazione nella directory che contiene i file degli elaborati e per utilizzarlo basta usare la relativa interfaccia a linea di comando;
5. esclusione di eventuale codice template dal confronto: è possibile;
6. esclusione di file molto piccoli: sì;
7. confronto storico: no;
8. valutazione: confronta per file ma memorizza i risultati per elaborato;
9. utilizzo: in locale;
10. open source: sì con licenza GNU.

SIM

Sviluppato nel 1989 da Dick Grune all'Università di Amsterdam, può testare programmi scritti in Java, C, Pascal, Modula-2, Lisp, Miranda e linguaggio naturale.

1. linguaggi supportati: Java, C, Pascal, Modula-2, Lisp, Miranda e linguaggio naturale;
2. espandibilità: è possibile fornendo una descrizione delle caratteristiche lessicali del nuovo linguaggio;
3. presentazione dei risultati: un file di testo;
4. semplicità d'uso del tool: interfaccia a linea di comando abbastanza semplice;
5. esclusione di eventuale codice template dal confronto: no;
6. esclusione di file molto piccoli: no;
7. confronto storico: sì;
8. valutazione: confronta per file ma può farlo anche per elaborato;
9. utilizzo: in locale;
10. open source: sì.

Confronto delle performance

Da alcuni esperimenti effettuati su programmi in Java per testare la sensibilità alle singole modifiche, tutti i tool sono risultati insensibili ai cambiamenti nei layout e nei commenti mentre solo Marble risulta insensibile agli spostamenti delle funzioni. Nessuno dei tool è completamente insensibile agli spostamenti delle classi. Le performance dei tool di fronte ad attacchi di plagio realizzati con la combinazione di diverse modifiche sono descritte di seguito:

- **JPlag:** rispetto ai 5 casi di plagio presenti nei codici testati, JPlag ne ha rilevati solo 2;
- **Marble:** si è comportato piuttosto bene rilevando quasi tutti i casi di plagio con un punteggio di similarità molto alto;
- **MOSS:** ha rilevato tutti i casi di plagio presenti;
- **Plaggie:** ha ottenuto punteggi di similarità molto alti nel caso di falsi allarmi inoltre non ha rilevato 2 casi di plagio su 5;
- **SIM:** ha ottenuto risultati molto diversi per i vari codici.

I tool spesso lavorano su un numero molto alto di elaborati che a loro volta possono contenere diversi file. Il tempo di esecuzione varia da alcuni minuti fino a 30 minuti.

In conclusione: molti tool sono sensibili a numerosi piccoli cambiamenti; tutti lavorano bene per la maggioranza delle modifiche singole; JPlag,

Marble e **MOSS** si comportano in modo abbastanza simile di fronte a file sottoposti ad una combinazione di modifiche.

Rilevare il plagio nelle piattaforme di applicazioni mobile

Anche il mondo mobile non è immune al grave problema del plagio. La facilità di accesso alle applicazioni mobile grazie al mercato delle app favorisce il comportamento disonesto di alcuni sviluppatori che riutilizzano illegalmente il codice sorgente violando la proprietà intellettuale degli autori. La logica di tipo open su cui si basano i sistemi operativi mobile e la disponibilità di licenze free dei Software Development Kit (SDK) rendono molto semplice e veloce la realizzazione di applicazioni, inoltre anche la commercializzazione delle app è in costante aumento, data la facilità d'accesso ai mercati delle app da parte degli utenti di smartphone e/o tablet.
Purtroppo è semplice anche riutilizzare illegalmente i codici sorgente scritti da altri, anche grazie ai tool di decompilazione che tentano di ricostruire i sorgenti a partire dai file usati per la distribuzione delle app (ad esempio i file *.jar, nel caso di applicazioni scritte in linguaggio Java).
Considerato lo sviluppo esponenziale del mercato mobile negli ultimi anni, rilevare i casi di plagio nelle applicazioni è molto importante per proteggere i diritti di proprietà intellettuale degli autori ed evitare loro una perdita economica anche considerevole, soprattutto quando nel caso in cui un'applicazione scritta in un linguaggio viene copiata e riadattata ad un'altra piattaforma.
Una ricerca dell'ITB (Institut Teknologi Bandung), istituto indonesiano che afferisce alla Scuola di Ingegneria Elettrica ed Informatica, ha confrontato due applicazioni realizzate in Symbian e in Android per valutarne la similarità.
Il procedimento ha previsto diverse fasi:

- decomposizione del package di installazione: con conseguente analisi della struttura dell'applicazione per capire quali parti eseguono quali azioni;
- preselezione: per ogni parte sospetta, evidenziata nella decomposizione, si selezionano alcuni documenti (classi, funzioni, librerie, ecc.) che potrebbero contenere parti di codice copiate;

- valutazione della similarità: dopo aver compreso la struttura di ogni package e aver selezionato alcune parti sospette, si confrontano le parti che contengono codice sorgente per rilevare eventuale similarità. Per questo confronto si può effettuare un confronto lessicale, attraverso la tokenizzazione del codice, oppure si può seguire un approccio strutturale che prevede la conversione di ogni codice in una forma rappresentativa come gli alberi sintattici e il successivo confronto delle forme corrispondenti a ciascun codice.

L'esperimento in questione ha preso in esame i package di installazione di due applicazioni di gioco per mobile. La versione originale era in formato Symbian Java (*.jar) ed era possibile scaricare online i file di installazione. A partire da novembre 2012 era apparsa online una applicazione che si sospettava fosse la copia della Symbian di cui sopra, tanto che lo sviluppatore che l'aveva pubblicata, rimosse dalla rete tutti i riferimenti. Usando la cache di Google, è stato possibile recuperare il materiale che era disponibile prima di novembre 2012 ed è così che è emersa un package di installazione Android (*.apk) illegale.

La decomposizione dei due package *.jar e *.apk ha mostrato che l'applicazione Android era stata realizzata illegalmente poiché la struttura delle due app era sostanzialmente identica, come mostrato nella figura 40, inoltre l'esecuzione di ogni app generava lo stesso flusso di istruzioni e le funzioni avevano nomi molto simili.

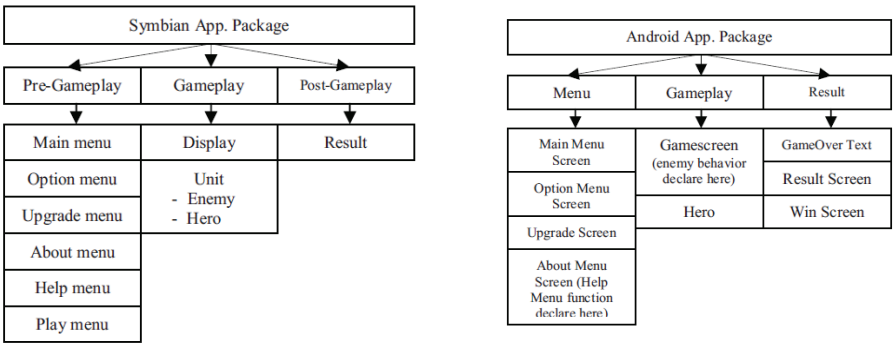

Figura 40 – Struttura dei package delle app per Symbian e per Android

È risultato evidente che l'applicazione originale sviluppata in ambiente Symbian era stata successivamente tradotta in ambiente Android.

La legge indonesiana sulle transazioni informatiche ed elettroniche (UUITE) condanna il plagio come atto criminale e tutela il software (giochi compresi) col diritto d'autore, includendo nel concetto di software il codice sorgente, il layout, l'architettura e l'interfaccia grafica.

L'individuazione del plagio fra schemi di database

L'individuazione manuale di plagio nei database è molto dispendiosa perciò è meglio usare un tool specifico. Quello affrontato nel testo di Abd El-Whaled, Elfatatry e Abougabal dell'Università di Alessandria d'Egitto si basa su un algoritmo che genera un'impronta digitale per ogni singola tabella dello schema.

Il matching dello schema di un database consiste nell'identificazione di due elementi di due schemi diversi come sintatticamente equivalenti. Ciò può avvenire utilizzando due tecniche diverse.

Tecniche basate su regole. Si possono impostare regole basate sui nomi degli elementi, sui tipi di dato, sulle strutture, sui numeri dei sottoelementi. Queste tecniche presentano diversi vantaggi: sono relativamente economiche e non richiedono formazione; operano solo sugli schemi e non sulle istanze quindi sono molto veloci; possono lavorare molto bene in certi tipi di applicazioni.

Tecniche basate sull'apprendimento. Queste tecniche di solito usano sia elementi dello schema che dati per stabilire il matching, quindi sono più precise ma richiedono un set di dati consistente per generare match corretti.

Come per il rilevamento di plagio nel codice sorgente, anche il rilevamento di plagio nei database richiede una fase di suddivisione in token, per ottenere un'impronta digitale per ogni tabella presente nel database.

Il meccanismo sul quale si basa questo tool è riassunto nella figura 41, di seguito riportata.

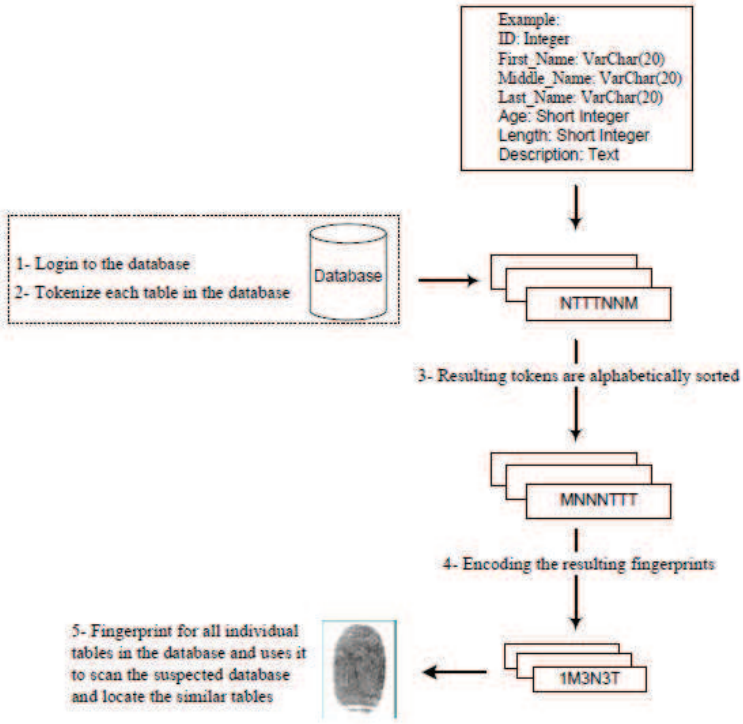

Figura 41 – Individuazione di plagio nello schema di un database

Un tool per individuare il plagio fra progetti Microsoft Access

Casi di plagio si verificano non solo fra codici sorgente ma anche fra applicazioni grafiche come Microsoft Access. McCart e Jarman, dell'Università della Florida del Sud, hanno implementato un tool per il rilevamento di plagio fra progetti realizzati in Microsoft Access, il cui utilizzo ha decrementato in modo significativo i casi di plagio nel corso di appena un semestre.

Il tool è stato utilizzato nell'ambito del corso di Information System, destinato agli studenti di economia presso una grande università pubblica. Il corso era diviso in tre moduli, ciascuno contenente progetti ed esami, con un numero di progetti da valutare compreso fra 100 e 250 per modulo.

Al primo anno di corso gli studenti vengono ampiamente informati sulle politiche universitarie nei casi di frode, dove con frode si intende sia il plagio che la copia che ogni altra contraffazione del proprio lavoro. La collaborazione fra studenti è proibita poiché ogni esercitazione deve essere svolta individualmente. Si considerano violazioni di queste regole tutti i casi seguenti:

- copiare un progetto col permesso di un altro studente;

- copiare un progetto dal file di qualcun altro che è rimasto inavvertitamente aperto su un computer pubblico;
- copiare parte di un progetto;
- importare un altro progetto all'interno del proprio;
- collaborare con altri studenti per completare il proprio progetto.

Nel caso in cui uno studente presenti un progetto che si scopre essere copiato, riceve una penalizzazione di -50 punti.

La ricerca si è concentrata sui progetti realizzati in Microsoft Access, applicativo che consente la creazione di database con tabelle, form di input, query, report.

Per verificare manualmente la presenza di plagio, ogni progetto realizzato in Access viene aperto e si annota la data di creazione del database ricavandola dalle proprietà. Progetti con date di creazione identiche vengono esaminati più a fondo per determinare se sono stati effettivamente copiati.

Ovviamente questo metodo manuale richiede molto tempo e non garantisce l'individuazione di tutti i duplicati; con il tool CCPE (Cheater Cheater Pumpkin Eater), la cui interfaccia è rappresentata nella figura 42, si è reso automatico il procedimento.

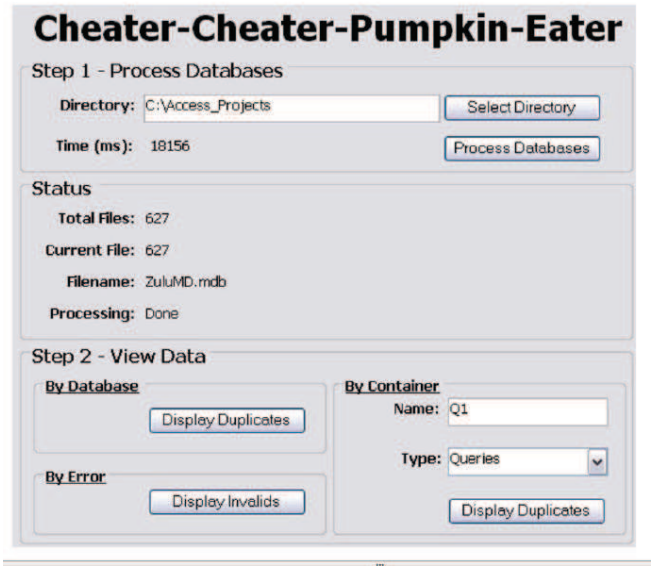

Figura 42 – Screenshot del tool CCPE

Quando si genera un database, parallelamente viene creato anche un documento (DO, Document Object) che riassume le proprietà principali del database (data di creazione, data dell'ultimo aggiornamento e nome).

Quando si copia un database, si copiano automaticamente anche i relativi DO. Per confrontare i progetti, CCPE accede ai DO e li estrae.

Nel primo step del procedimento CCPE apre i progetti da confrontare, che devono essere collocati tutti nella stessa directory, ed estrae le proprietà di ciascun progetto (vedi figura 43, step 1).

Prima di tutto CCPE confronta le date di creazione dei database (figura 43, step 2) e, se è la stessa, confronterà anche i titoli (figura 43, step 3a).

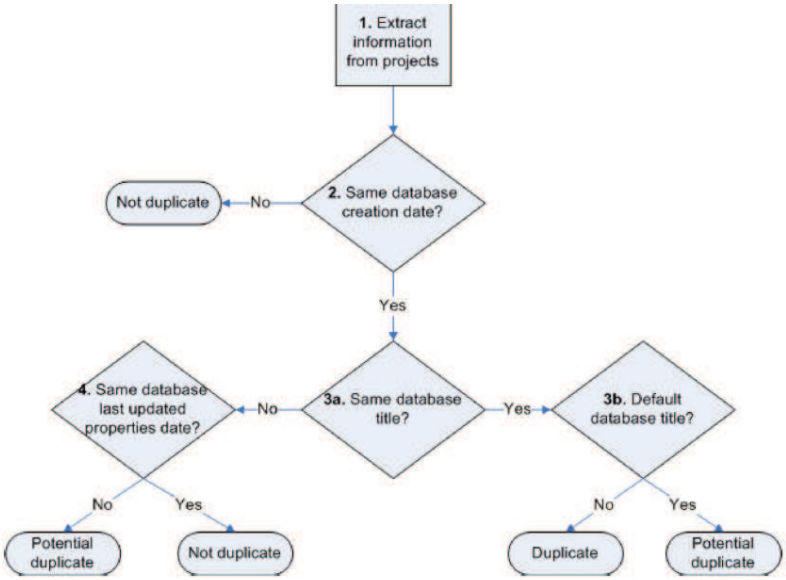

Figura 43 – Procedimento di confronto dei database

Il caso in cui due progetti abbiano la stessa data di creazione è raro ma non impossibile; in questo caso CCPE confronta anche la data dell'ultimo aggiornamento. Ad esempio, come mostrato nella tabella 7, data di creazione e di ultimo aggiornamento coincidono, il che significa che nessuno dei due studenti ha modificato titolo o altre proprietà del database dopo averlo creato pertanto i due progetti non sono copiati.

File Name	*Database Title*	*Database Creation Date*	*Last Updated Properties Date*	*Result*
AdamsBW.mdb	AdamsBW	9/7/06 4:00:00 PM	9/7/06 4:00:00 PM	Not Duplicates
JonesPS.mdb	JonesPS	9/7/06 4:00:00 PM	9/7/06 4:00:00 PM	

Tabella 7 – Progetti con la stessa data di creazione ma non copiati

CCPE assegna un punteggio da 0% a 100% a seconda di quanti elementi simili sono emersi dal confronto. La figura 44 descrive il procedimento usato per calcolare la percentuale di similarità.

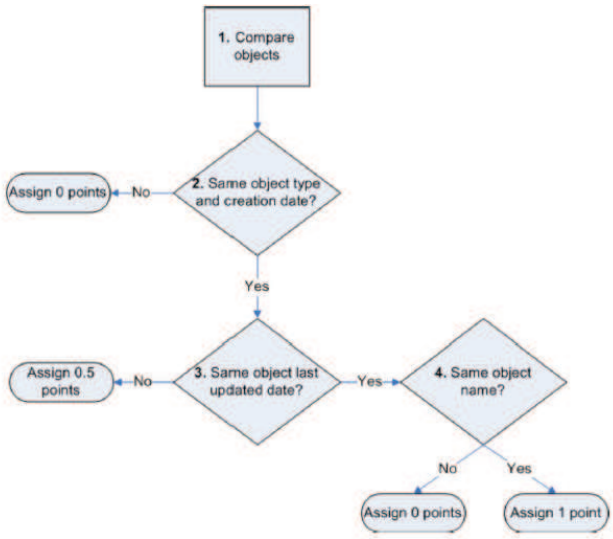

Figura 44 – Confronto fra oggetti del database

Allo step 4, se gli oggetti hanno lo stesso nome, si assegna 1 punto e l'oggetto è considerato copiato mentre in caso contrario l'oggetto non è copiato quindi il punteggio è 0.

Gli studenti sono stati informati dell'introduzione di CCPE nel loro corso per individuare database copiati; se nell'autunno 2004 i casi di plagio erano il 6,7%, nella primavera 2005, dopo un semestre di utilizzo di CCPE, questa percentuale è scesa a 0,5%. CCPE è uno strumento efficace ma semplice da ingannare; probabilmente la diminuzione così rilevante di casi di plagio è dipesa anche dall'introduzione della penalizzazione di 50 punti per ogni azione di plagio scoperta, che ha costretto gli studenti ad assumere un comportamento più rispettoso delle regole.

Capitolo 5 – CONCLUSIONI

Il plagio nel mondo accademico

Il sistema SSID

I sistemi di rilevamento di plagio esistenti si concentrano sul confronto fra coppie di elaborati. Alcuni docenti e ricercatori dell'Università Nazionale di Singapore hanno sviluppato SSID (Student Submission Integrity Diagnosis) un sistema *open source* che consente di rilevare il plagio basandosi sulle istruzioni. SSID individua i cluster sospetti di plagio cercandoli fra tutti gli elaborati assegnati in un corso oltre a evidenziare gli elaborati più sospetti di un singolo studente.

Nel sistema SSID si utilizza un metodo basato sulla struttura ma modificandolo per migliorarne accuratezza ed efficienza. Dato un set di elaborati, SSID segue tre step per giudicare la presenza o meno di plagio:

1. la divisione in token è il primo step. SSID è implementato per programmi C e Java. In particolare, ricevuti in input degli elaborati in Java, si definiscono quattro tipi di token: costanti, parole chiave, simboli e variabili. Numeri e caratteri vengono classificati come costanti, ad ogni stringa viene assegnato un valore hash e, alla fine del procedimento, verrà indicizzato in una tabella hash un token *N-grams* che rappresenta ogni elaborato;
2. per ogni coppia di elaborati, si calcola la similarità, usando l'algoritmo Greedy-String-Tiling. Due elaborati *A* e *B* si considerano identici se e solo se i loro token contigui sono identici; l'algoritmo si applica ricorsivamente fino ad ottenere un punteggio di similarità scelto come il valore più alto ottenuto. Questo step crea una rappresentazione hash per ogni elaborato;
3. l'ultimo step è la determinazione di cluster-plagio. SSID raggruppa gli elaborati che hanno il punteggio di similarità più alto e il

clustering usa la soglia di similarità per classificare ogni elaborato nel gruppo cluster-plagio o meno.

Il sistema SSID è stato testato coinvolgendo 28 studenti che hanno generato versioni plagiate di codici sorgente che avevano realizzato precedentemente. Le tipologie di plagio realizzate sono state di tre tipi:

- attacchi immutabili: nessuna modifica salvo modifiche, inserimenti o cancellazione di commenti o costanti. SSID è risultato immune a questi attacchi;
- attacchi influenzati dalla dimensione: in base alla dimensione del segmento di codice modificato (ad esempio con riordino dei metodi, inserimento o rimozione di parentesi) SSID è stato messo più o meno in difficoltà;
- attacchi efficaci: si tratta di quegli attacchi che hanno confuso SSID in tutti i casi, come ridondanza, modifica dello scope, modifica di parametri e variabili nelle funzioni, ristrutturazione del codice.

SSID presenta un web log che tiene traccia di tutte le attività di plagio compiute dagli studenti e offre diverse interfacce per la visualizzazione dei vari tipi di informazioni memorizzate. Questa caratteristica differenzia SSID da quasi tutti gli altri sistemi simili. Ogni rapporto relativo a coppie sospette viene memorizzato nel sistema, con un riferimento allo studente che ha realizzato l'elaborato. Dopo aver verificato una serie di elaborati con SSID si ottiene un elenco significativo degli studenti che più spesso hanno compiuto azioni di plagio, come illustrato in figura 45, e questa informazione risulta un valido aiuto per il docente nella lotta contro il plagio fra codici sorgente.

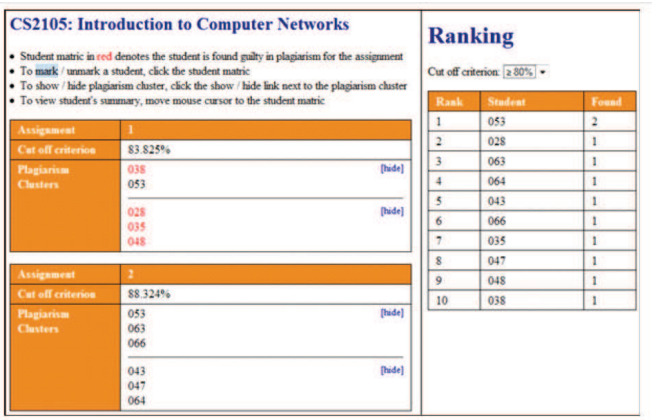

Figura 45 – Snapshot dell'interfaccia *Assignments* del sistema SSID. Il frame di sinistra mostra una panoramica di tutti i cluster-plagio individuati negli elaborati. Il frame di destra mostra le statistiche dei 10 studenti che compaiono più frequentemente nei cluster-plagio

L'integrazione di MOSS e JPlag nell'ambiente di apprendimento virtuale Moodle

Alla Monash University di Melbourne è stato studiato un plugin di **Moodle**, ambiente di apprendimento virtuale, che integra le funzionalità di **JPlag** e **MOSS** nel tentativo di realizzare un più efficace strumento di individuazione di plagio fra codici sorgente.

Il plagio fra codici sorgente è un fenomeno in continuo aumento; questa tendenza può essere spiegata con la competitività esistente fra gli studenti, il timore di sbagliare, la facile reperibilità di codici già pronti e l'abitudine a riutilizzare codici già predisposti per esercitazioni precedenti.

Con l'aiuto di moderni ambienti di sviluppo, è molto semplice modificare un codice sorgente con pochi click e realizzare un codice apparentemente molto diverso da un altro.

Molti tool per l'individuazione automatica del plagio fra codici sorgente, come **MOSS** e **JPlag**, possono essere di grande aiuto nell'analisi di codici sospetti. Il tema della ricerca proposta alla Monash riguarda l'integrazione di **MOSS** e **JPlag** nell'ambiente virtuale di apprendimento Moodle, al fine non solo di promuovere l'utilizzo dei tool per il rilevamento del plagio ma anche di educare gli studenti ad una più profonda integrità accademica.

La ricerca è iniziata con un'intervista agli accademici (docenti e tutor) finalizzata a comprendere le pratiche in uso per il rilevamento di plagio e a determinare il livello di soddisfazione e le difficoltà incontrate.

La maggior parte degli accademici coinvolti ha dichiarato di non avvalersi di strumenti per il rilevamento automatico del plagio, puntando più alla prevenzione del plagio, attraverso l'educazione degli studenti e l'adozione di varie tecniche:

- impostare una struttura complessa per le esercitazioni affidate, per fare in modo che non sia facile trovare le soluzioni su Internet;
- dare maggior peso agli esami piuttosto che alle esercitazioni, affinché queste ultime siano considerate meri strumenti per arrivare preparati all'esame e, quindi, non si senta più bisogno di copiare per eseguirle;
- evitare di riutilizzare esercitazioni già affidate in corsi precedenti, affinché gli studenti non possano copiare dai loro colleghi;

- offrire agli studenti esercitazioni collettive durante le lezioni, affinché tutti gli studenti possano verificare il proprio livello di apprendimento e chiedere al docente eventuali chiarimenti.

I docenti hanno affermato di non riuscire a controllare in maniera soddisfacente il fenomeno del plagio, a causa del numero elevato di studenti per ogni corso, del tempo necessario per valutare ogni studente e della difficoltà ad individuare codice copiato quando gli studenti utilizzano lo stesso scheletro di codice.

MOSS e **JPlag** godono di ottima reputazione nel mondo accademico, perciò sono stati scelti per sviluppare un plugin che rispettasse tali requisiti:

- *semplicità e velocità di utilizzo*: l'uso di questo strumento non avrebbe dovuto comportare sforzi per gli utenti o rallentamenti nel processo di assegnazione e consegna dei compiti;
- *pubblicazione di una versione limitata dei rapporti di similarità tra codici degli studenti*: questo per rispettare la privacy degli studenti stessi, dal momento che il plugin vuole essere non uno strumento di pubblica accusa ma un supporto all'educazione e all'apprendimento.

Per realizzare questi obiettivi, il plugin è stato dotato delle seguenti funzioni:

- *filtro automatico dei codici*: il plugin individua automaticamente i sorgenti fra i documenti consegnati dallo studente;
- *scansione automatica e generazione di un report*: i sorgenti consegnati vengono analizzati automaticamente dal sistema che restituisce in un report i risultati della scansione;
- *pubblicazione dei report*: i docenti possono abilitare gli studenti ad una visualizzazione parziale dei report.

In base all'opinione di studenti e docenti, il plugin proposto è risultato un valido strumento per assistere i docenti nel rilevamento del plagio e anche per scoraggiare gli studenti a copiare, considerata l'elevata probabilità di venire scoperti.

Una nuova strategia nell'insegnamento della programmazione: il codice scheletro

Gli studenti copiano per diverse ragioni: perché è facile; perché si organizzano male e si ritrovano ad avere troppo poco tempo per svolgere

autonomamente i compiti ad essi affidati; perché sono poco motivati e vogliono ottenere buoni voti senza troppa fatica.

Non esiste una tecnica infallibile per l'individuazione di plagio, né fra i tool automatici, né fra le tecniche manuali. Una recente ricerca sviluppata dalla dr.ssa Minh Ngoc Ngo del SIT (Singapore Institute of Technology) ha proposto un metodo alternativo per evitare il plagio fra codici sorgente: proporre agli studenti uno scheletro di codice sorgente che li costringa a seguire una certa struttura per realizzare il loro programma. In questo modo risulta praticamente impossibile copiare il codice da un altro poiché il codice deve essere comunque modificato e adattato allo scheletro proposto.

Questa nuova strategia di assegnazione degli esercizi si prefigge diversi obiettivi: chiarire i risultati da ottenere; aiutare gli studenti ad apprendere; promuovere un apprendimento omogeneo fra tutti gli studenti.

Il primo step per realizzarli è la creazione del codice scheletro, sul quale tutti gli studenti si baseranno per realizzare il loro lavoro. Con questo strumento gli studenti svilupperanno la loro abilità nel valutare il codice esistente e saranno incoraggiati a sviluppare le proprie idee piuttosto che a copiare il codice direttamente da Internet.

Definito lo scheletro, è necessaria una modalità di insegnamento che aiuti gli studenti a lavorare con costanza e favorisca il lavoro individuale: delle scadenze settimanali nella consegna dei lavori e un test in classe ogni due esercizi di programmazione assegnati potrebbero essere strategie efficaci per motivare gli studenti e minimizzare la loro tendenza a procrastinare lo svolgimento del lavoro.

In questa modalità, non ci si preoccupa di verificare se lo studente ha copiato o meno ma si testa la reale comprensione della materia. Gli studenti sono liberi di studiare e imparare codice sorgente scaricato da Internet o realizzato dai propri compagni; poiché gli studenti verranno valutati sulla base dei test svolti in classe, per poterli superare ogni studente dovrà avere imparato realmente, quindi copiare senza studiare non risulterà una strategia vincente.

La strategia descritta è stata applicata in un corso di programmazione a settembre 2014 e a gennaio 2015 la percentuale di fallimento per gli studenti di questo corso è stata del 6,3% mentre per altri corsi di programmazione la percentuale oscillava tra il 25% e il 50%. Anche nei mesi successivi, quando

la difficoltà degli argomenti trattati è cresciuta, gli studenti hanno mostrato di gradire questa modalità di insegnamento e quasi il 60% di loro ha affermato di aver incrementato le proprie competenze seguendo il corso di programmazione.

Questa indagine dimostra che è necessario accertarsi che gli studenti abbiano compreso gli esempi presentati e non copiato il codice sorgente solo per poter completare l'esercizio assegnato. Definire codici scheletro e strategie di insegnamento e testare la comprensione sono operazioni lunghe e complesse, ma i primi risultati raggiunti suggeriscono che il metodo proposto può significativamente migliorare il livello di comprensione e di competenza sviluppato dagli studenti.

Una struttura semi-automatica per il rilevamento del plagio

Come individuare il plagio all'interno del codice sorgente scritto da uno studente di informatica? La ricerca del plagio richiede tempo poiché richiede di confrontare fra loro coppie di documenti che contengono centinaia o anche migliaia di righe. B. Lesner, R.Brixtel, C.Bazini e G.Bagan hanno proposto una struttura semi-automatica che evidenzia le coppie di documenti più sospette leggendone il codice affinché successivamente sia possibile un controllo manuale. Tale strumento non è vincolato a nessun linguaggio di programmazione specifico, l'unica *conditio sine qua non* è che i documenti confrontati siano scritti nel medesimo linguaggio.

Nel codice sorgente definiamo il plagio come l'applicazione di trasformazioni successive applicate a un documento originale. Una trasformazione preserva il funzionamento del programma ma non il suo aspetto. L'obiettivo degli autori dell'indagine è di riuscire a gestire 4 tipi di trasformazioni: ridenominazione, riordinamento del codice, aggiunta/rimozione di commenti, sostituzione di sezioni di codice o istruzioni con altre analoghe. Più trasformazioni vengono rilevate, meno i documenti vengono considerati plagio l'uno dell'altro.

Una struttura di questo tipo può risultare molto utile per aiutare i docenti ad individuare codici plagiati, soprattutto perché consente di ridurre il numero di coppie da confrontare manualmente.

La struttura si basa su un approccio *bottom-up*, costituito da sei tappe principali: 1) *pre-filtering*; 2) *segmentazione e misurazione della somiglianza*; 3) *matching* dei segmenti; 4) *post-filtering*; 5) *valutazione della distanza*; 6) *presentazione dell'analisi* del codice. I primi step operano a livello dei caratteri, quelli successivi operano sulle stringhe, sul documento e infine arrivano a livello di codice sorgente.

L'algoritmo 1 rappresenta il modo in cui le diverse fasi interagiscono fra loro.

Algorithm 1: Main algorithm

```
Input: 𝒟: a set of documents
begin
    /* Prefilter the documents                          */
    𝒟′ ← {PreFilter(d) | d ∈ 𝒟}
    /* Segment filtered documents                       */
    foreach d′_i ∈ 𝒟′ do S_i ← Seg(d′_i)
    /* Compute the distance between each pair of
       documents                                        */
    foreach d′_i, d′_j ∈ 𝒟′, i > j do
        ℳ_(i,j) ← DocumentDistance (S_i, S_j)
    /* Return a human-readable result                   */
    return Display (ℳ)
end
```

L'operazione di *pre-filtering* sostituisce ogni stringa alfanumerica (ad esempio nomi di variabili o parole chiave) con un simbolo singolo.

Durante la fase di *segmentazione e misurazione della somiglianza* ogni documento viene diviso in segmenti, dove con *segmento* si intende un sottinsieme contiguo di codice e la funzione di segmentazione *Seg* divide il documento *d* in una sequenza di segmenti $Seg(d) = (s_{1, ..., } s_m)$. La funzione di misurazione della distanza tra due segmenti, $Dist(s_{1,} s_2)$, è un numero reale incluso fra 0 e 1 che soddisfa le proprietà della distanza. Dati due segmenti $S_1 = (s_1^1, ..., s_m^1)$ e $S_2 = (s_1^2, ..., s_n^2)$ e la funzione *Dist*, otteniamo una matrice di distanza *M,* di dimensione $m \times n$, in cui $M_{(i,j)} = Dist(s_i^1, s_j^2)$.

Quando si è arrivati ad una segmentazione sufficientemente buona, le sezioni del documento contigue e simili generano una diagonale che appare all'interno della matrice rilevando la possibile presenza di plagio; spesso tale diagonale è difficile da vedere a causa del rumore, come in figura 46, ma in ogni caso è importante sottolineare che documenti molto diversi non presentano alcuna diagonale nella loro matrice di distanza.

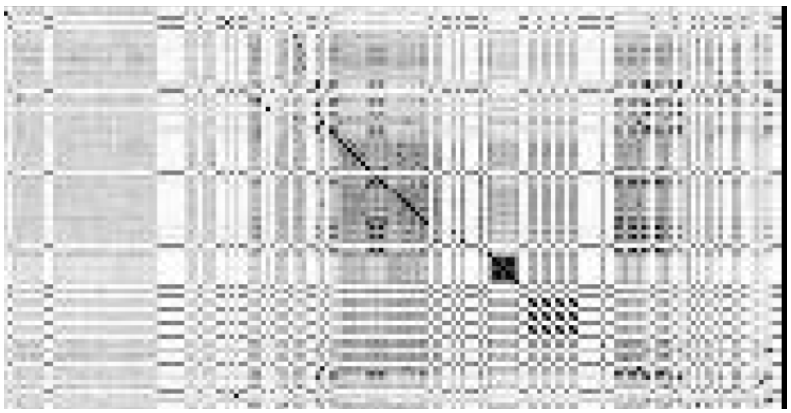

Figura 46 – Matrice della distanza fra due documenti plagiati. I punti luminosi indicano una distanza piccola

Per la misurazione della distanza fra segmenti si possono utilizzare diverse funzioni che contano il numero di operazioni (inserimenti, cancellazioni e sostituzioni) che occorrono per trasformare un segmento in un altro, oppure si può utilizzare la distanza di informazioni, che valuta la distanza segmentando il testo riga per riga.

Il *matching dei segmenti* e il *post-filtering* utilizzano la matrice di distanza M ottenuta nelle fasi precedenti per calcolare la distanza tra i due documenti; a tal fine si cerca il matching massimo della distanza minima rappresentata in M.

Il matching è definito come un gruppo di coppie $C \subset S_1 \times S_2$ tale che ogni segmento di S_1 e S_2 appaia in al massimo una coppia di C. Il matching C è massimo se e solo se tutti i segmenti della segmentazione più piccola sono contenuti in C e inoltre $|C| = min(|S_1|, |S_2|)$. La distanza del matching C è definita come:

$$\Sigma_{(s_i^1, s_j^2) \in C} M_{(i,j)}$$

Si tratta di un metodo robusto per le modifiche dei commenti: se vengono aggiunti dei commenti, la dimensione del documento aumenterà rispetto all'originale perciò difficilmente risulterà un matching fra i due documenti.

Si utilizza l'algoritmo di Munkres per eseguire il matching e ottenere la matrice di matching H, di dimensione $m \times n$, tale che $H_{(i,j)} = M_{(i,j)}$ se $(s_i^1, s_j^2) \in C$ e $H_{(i,j)} = 1$ altrimenti. Questo algoritmo ha una complessità computazionale di $O(max(m, n)^3)$.

La matrice risultante viene poi filtrata con una matrice identità di convoluzione per evidenziare le diagonali e rimuovere i match isolati.

Per eseguire la *valutazione della distanza* è sufficiente sommare e normalizzare gli elementi della matrice *M* con valore minore di 1, dando una distanza compresa tra 0 e 1, come rappresentato nell'ultima riga dell'algoritmo 2.

Algorithm 2: DOCUMENTDISTANCE

Input: S_1, S_2: two sets of segments (one per document)
Data: $Dist$: a segment distance function
begin
  /* Build the segments distance matrix $\mathcal{M}$ */
  foreach $(s_i, s_j) \in S_1 \times S_2$ **do**
    $\mathcal{M}_{(i,j)} \leftarrow Dist(s_i, s_j)$
  /* Find max matching with min distance */
  $\mathcal{M}' \leftarrow Matcher(\mathcal{M})$
  /* Post-filter the matching matrix */
  $\mathcal{M}'' \leftarrow PostFilter(\mathcal{M}')$
  /* Return the document-wise distance */
  return $1 - \frac{1}{\min(|S_1|,|S_2|)} \sum_{i,j} 1 - \mathcal{M}''_{(i,j)}$
end

A questo punto ci occupiamo della *presentazione dell'analisi del codice*. Assumiamo di avere una distanza per ogni coppia di documenti del codice e di riportare tali distanze in un foglio elettronico in cui ogni cella contenga una distanza e sia evidenziata con un colore. Il colore della cella rappresenta la somiglianza tra i documenti in termini di somiglianza media del codice: le coppie possono essere classificate in 8 o 16 classi, ciascuna associata ad un colore che va dal verde (documenti legittimi) al rosso (documenti con alta probabilità di plagio). Si usa poi un algoritmo di classificazione gerarchica per generare un albero binario del quale i documenti rappresentano le foglie; l'ordine delle foglie seguito nell'attraversamento in profondità di questo albero viene applicato alle righe e alle colonne della matrice di distanza finale per raggruppare i documenti simili in celle vicine.

L'applicazione del procedimento appena descritto ha dato esiti molto diversi in presenza di linguaggi di programmazione diversi come C, Python, Haskell, Bash, PHP e Java. In diversi casi è stato rilevato plagio in documenti sospetti ma non è possibile stabilire se tutti i documenti che non sono risultati sospetti fossero effettivamente lavori originali. Tuttavia i risultati di questo procedimento sono promettenti poiché consente di individuare frodi che precedentemente sfuggivano, anche quando gli studenti utilizzavano lo stesso codice di base. Il lavoro prosegue con

l'intenzione di facilitare il lavoro di correzione che permette di analizzare il codice sia a livello di documenti che di segmenti.

Un editor anti plagio: prevenzione e rilevamento

È importante parlare anche di prevenzione del plagio oltre che del suo rilevamento.

La pensano così Peter Vamplew e Julian Dermoudy, docenti dell'Università della Tasmania, che, in occasione di una conferenza sulle tecnologie informatiche, hanno descritto un approccio anti-plagio basato sulla considerazione dell'intero processo di creazione dei codici sorgente, piuttosto che sull'analisi del solo codice sorgente.

Parlando di approccio educativo, gli studiosi sottolineano come il primo passo da compiere sia quello di informare gli studenti circa la natura del plagio, le ragioni per le quali si considera inaccettabile, i rischi che corre chi decide di compiere azioni di plagio. Nelle classi in cui si utilizzano strumenti per il rilevamento automatico del plagio e gli studenti ne sono informati, di solito i casi di plagio sono meno numerosi.

Per scoraggiare il plagio è buona prassi modificare ogni anno le esercitazioni che si assegnano agli studenti e fornire loro uno scheletro sul quale sia complicato copiare e incollare un codice generico scaricato dalla rete.

I tool automatici permettono di verificare i file presentati dagli studenti e identificare quelli sospetti. Alcuni sono disponibili come web service (come **Turnitin**) e la loro forza sta nella continua espandibilità dei database di documenti da utilizzare per i confronti futuri.

Per quanto riguarda la ricerca di plagio fra codici sorgente, lo sviluppo di software risale almeno al 1976 quando comparvero i primi tool basati su varie metriche come il conteggio del numero di occorrenze di una particolare operazione.

Sistemi più evoluti sono stati sviluppati in epoca recente e si basano sull'analisi della struttura sottostante al codice. I due tool più conosciuti di quest'ultima generazione sono JPlag e MOSS che analizzano il codice, lo suddividono in token e quindi applicano ai token degli algoritmi di confronto per calcolare la similarità.

Caratteristiche del sistema: *APE* e *Gorilla*

Non esiste una bacchetta magica per combattere il plagio perciò è realistico pensare che utilizzare diverse tecniche fra loro complementari possa dare risultati migliori. L'approccio di Vamplew e Dermoudy prevede due livelli: da un lato controllare le azioni degli studenti durante lo sviluppo dei propri lavori, dall'altro rivedere il codice una volta che il lavoro è stato presentato.
Questo tipo di monitoraggio è molto simile a quello realizzato con PowerResearcher, un software commerciale che fornisce un ambiente di sviluppo che integra diverse caratteristiche richieste per la ricerca o la didattica.
La componente chiave del sistema qui presentato è un editor di testo specializzato, l'editor anti plagio o *APE*, che include caratteristiche che scoraggiano azioni di plagio; una seconda componente, chiamata *Gorilla*, è utilizzata per classificare i lavori consegnati dagli studenti sulla base di informazioni addizionali.
APE memorizza delle informazioni aggiuntive per ogni file che viene salvato. In particolare, oltre al testo scritto dallo studente, per ogni file *APE* annota i tempi di clock di sistema relativi ad ogni modifica e li memorizza insieme al codice sorgente.
Ovviamente l'inclusione di queste informazioni aggiuntive può interferire con l'attività del compilatore o dell'interprete; per ovviare a questo inconveniente è possibile salvare simultaneamente due differenti versioni del file (soluzione questa che però potrebbe confondere lo studente) oppure memorizzare i dati di identificazione sotto forma di commento all'inizio o alla fine del codice, così le informazioni saranno a disposizione ma non contrasteranno con l'attività di compilatori o interpreti.
Per evitare che uno studente possa modificare le informazioni aggiuntive utilizzando un qualunque editor, per poi presentare una nuova versione del file, nel momento in cui un file viene caricato in *APE* o in *Gorilla* il contenuto del file viene usato per generare un codice di autenticazione che sarà confrontato con il codice già memorizzato; se il codice è stato modificato con un altro editor diverso da *APE*, il sistema lo segnalerà.
APE impedisce il copia & incolla da altre applicazioni: copiare e incollare testo è possibile solo attraverso la sezione appunti che *APE* mette a disposizione localmente. È basato sull'ambiente di sviluppo Eclipse, dal

quale eredita funzionalità come l'evidenziazione del testo, l'indentazione automatica, le operazioni di trova e sostituisci e il supporto al debug.

APE e *Gorilla* non rappresentano che uno degli strumenti a disposizione per la lotta contro il plagio; alla Scuola di Informatica dell'Università della Tasmania questa lotta si combatte su tre piani: educare gli studenti a non compiere operazioni di plagio (e informarli sull'utilizzo di strumenti come *APE* e *Gorilla* è già un buon deterrente); rendere il rapporto costi/benefici insito nel plagio meno attraente per gli studenti (ovvero ridurre il peso delle esercitazioni sul voto finale affinché copiare non apporti sostanziali benefici); rilevare e punire i casi di plagio, in accordo con le regole dell'Università (la punizione più comune consiste nell'acquisizione di un voto negativo ma sono previste pene più severe nel caso di plagio ripetuto).

Il rilevamento del plagio nell'e-learning

Il plagio è un problema globale, che interessa diversi ambiti delle nostre vite. Nei sistemi e-learning gli studenti caricano i loro lavori, come i codici sorgente.

Esistono diversi metodi e tool per l'individuazione automatica del plagio, molti dei quali condividono diversi svantaggi:

- hanno un'interfaccia grafica molto povera;
- ignorano commenti e stringhe di testo;
- sono poco robusti ai cambiamenti lievi;
- non riconoscono il riordinamento degli operandi nelle espressioni.

Un approccio proposto da un'università slovacca abbina un metodo di tokenizzazione sofisticato per il preprocessing all'algoritmo di Greedy-String-Tiling per il confronto dei token.

L'approccio funziona in questo modo:

- se il programma consiste in diversi file, questi file saranno integrati in uno solo di grandezza proporzionale;
- le variabili vengono rinominate;
- tutto il testo viene trasformato in minuscolo;
- si estraggono stringhe e commenti;
- si suddivide il codice in token;
- si confrontano i token;
- si presentano i risultati all'utente.

Rilevare la similarità tra due programmi è un'operazione complessa che richiede regole molto più sofisticate di quelle usate per analizzare testi scritti in linguaggio naturale. Il successo di un tool dipende drammaticamente dai dati disponibili per il confronto, ad esempio dal database dei lavori precedenti.

Va ricordato che un tool da solo non può essere la soluzione poiché il problema del plagio dipende dalle persone; ci deve essere qualcuno che si assume la responsabilità di rilevare il plagio, raccogliere le prove per dimostrarlo e dare un giudizio su quanto ha scoperto.

I problemi etici non saranno mai risolti da un software.

L'information retrieval: un supporto per l'individuazione del plagio

La difficoltà nell'individuazione del plagio aumenta quando abbiamo una vasta collezione di documenti da verificare. Uno studio presentato al FIRE (Forum for Information Retrieval Evaluation) che si è svolto a Bangalore nel 2014, propone una tecnica di recupero delle informazioni (*information retrieval*, IR) in base alla quale ogni documento è trattato come una pseudo-query e richiama una lista dei documenti più simili rispetto a quella query.

In pratica si accumulano documenti nella lista finché il valore di similarità dell'i-esimo documento rispetto all'(i-1)esimo è più alto di una soglia predefinita ε, secondo la seguente equazione:

$$Plag(Q) = \{ D_i : \frac{sim(Q, D_i) - sim(Q, D_{i-1})}{sim(Q, D_{i-1})} \leq \varepsilon \}$$

Ogni codice sorgente viene rappresentato attraverso un AST e poi si estraggono i termini da ciascun nodo dell'AST.

L'idea è poi quella di rappresentare il codice solo attraverso i suoi termini più rappresentativi, scelti in base alla frequenza con cui compaiono nella collezione dei documenti, secondo il linguaggio di modellazione (*LM*) descritto dalla seguente formula:

$$LM(t, f, d) = \lambda \frac{tf(t, f, d)}{len(f, d)} + (1 - \lambda)\frac{cf(t)}{cs}$$

dove *t, f, d* rappresentano rispettivamente il termine, la frequenza e il documento, mentre il parametro λ controlla la relativa importanza della frequenza del termine.

L'industria del software e la pirateria informatica

In un articolo dei ricercatori giapponesi Monden, Okahara, Manabe e Matsumoto, si affronta il problema della violazione delle licenze software.

Spesso le aziende, per incrementare la produttività, riutilizzano software *open source* (OSS) come parte di un proprio prodotto. Il problema della violazione è così ampio che le organizzazioni che sviluppano OSS, come il Software Freedom Law Center, hanno iniziato ad aiutare gli sviluppatori ad individuare le violazioni di licenze nei prodotti commerciali.

Diversi servizi sono disponibili per l'identificazione di codice OSS. La Black Duck Software's Protex analizza il codice sorgente usando la tecnologia di stampa del codice e confrontando il codice sorgente con una collezione di oltre 200.000 prodotti software. Palamida (industria di software californiana) rileva il riutilizzo di codice sorgente OSS attraverso una ricerca multipattern che comprende matching di frammenti di codice e classifica i matching in base alla rilevanza.

Quale numero di cloni o quale dimensione per singolo clone è la soglia per determinare se il programma sospetto è colpevole o no di violazione della licenza?

Per decidere come affrontare il problema si deve prima rispondere a due domande:

- Qual è la metrica più appropriata per valutare il numero e la dimensione di codice clonato fra due programmi?
- Qual è il limite minimo nella misura del codice clonato per concludere che il programma sospetto è colpevole e qual è il limite massimo per concludere che non lo è?

Per rispondere alla prima domanda, gli autori analizzano le diverse metriche con le quali si può misurare la clonazione in un codice e fra queste scelgono tre misure relative ai cloni: la lunghezza massima dei cloni (*MLC*); il numero di coppie di cloni (*NCP*); la similarità locale basata su cloni (*LSim*), che rileva la percentuale di duplicazioni in una coppia sospetta.

Per rispondere alla seconda domanda, prima di tutto si crea la struttura rappresentata in figura 47 per definire la colpevolezza o la non colpevolezza e quindi si usano le metriche prescelte per determinare i limiti massimo e minimo.

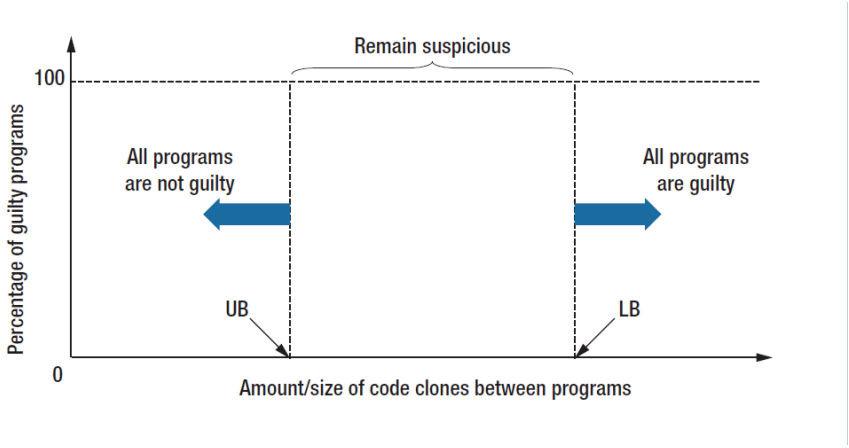

Figura 47 – Definizione di colpevolezza e non colpevolezza. Se due programmi hanno un numero o una dimensione abbastanza alti di codice clonato al loro interno, possono essere considerati colpevoli, ma abbiamo bisogno di un limite minimo (*LB*) per giungere a questa conclusione. Allo stesso modo, un programma che ha solo un numero o una dimensione bassi di codice clonato non è colpevole ma abbiamo bisogno di un limite massimo (*UB*) per affermarlo. Tra questi confini risiedono i programmi sospetti che richiedono un controllo umano per determinare il riutilizzo del clone

Clone Detection. I codici clonati sono esattamente o quasi linee copiate di codice in una coppia di programmi sorgente. I programmatori possono introdurre accidentalmente codice clonato usando frammenti di codice. Come descritto nel capitolo 2 della presente tesi, con clone di Tipo 2 si intende una copia sintatticamente identica tranne che per il nome delle variabili, dei tipi di dato o delle funzioni; per rilevare questo tipo di cloni viene utilizzato **CCFinderX**, versione aggiornata di **CCFinder** che confronta sequenze di token costituite da linee di codice sorgente e può individuare codice clonato confrontando oltre un milione di linee in un tempo computazionale e con requisiti di memoria ragionevoli.

Definizione delle metriche. Applicando *MLC*, *NCP* e *LSim* si possono distinguere cloni da altro software riutilizzato accidentalmente. Definiamo queste metriche come:

- *MLC* – numero di token della coppia di cloni più grande rilevata tra i due programmi;

- *NCP* – somma di tutte le coppie di cloni che contengono più di 30 token;
- *LSim* – percentuale di duplicazione all'interno della coppia di codici sorgente più sospetta.

LSim si può utilizzare per rilevare codice clonato quando un programmatore ha copiato uno o più file da un altro prodotto. Dati i due programmi da confrontare *A* e *B*, *LSim* si calcola come segue:

$$LSim\ (A,B) = \frac{2 \times MLC(A,B)}{|a| + |b|} \times 100$$

in cui *a* e *b* indicano la coppia di file con il clone più grande tra *A* e *B*, *MLC(A,B)* è la lunghezza del clone più grande e |*a*| e |*b*| indicano le lunghezze di *a* e *b* in numero di token. Se *a* e *b* sono identici, *LSim* restituirà il valore 100.

L'esperimento. Per testare le metriche, gli autori del paper hanno selezionato 50 prodotti OSS dalla Free Software Directory disponibile online; per questi 50 prodotti hanno controllato 1225 coppie per determinare se ogni coppia includeva del codice clonato. È stata applicata ricorsivamente questa procedura:

- per ogni coppia, un programmatore esperto ha individuato il clone più grande e ha giudicato in base alla sua esperienza se si trattava di un clone prodotto accidentalmente. Se no, il clone è stato considerato basato su software riutilizzato e quindi colpevole;
- se il clone è stato considerato colpevole, il programmatore è passato alla coppia successiva;
- se il clone non è stato considerato colpevole, il programmatore ha individuato il successivo clone più grande e ha continuato la sua analisi fino ad arrivare ai cloni di lunghezza 30 token.

Per calcolare il limite minimo di ogni metrica, ogni coppia di prodotti con il valore di metrica più alto è stata etichettata come potenzialmente colpevole. Poi sono stati calcolati precisione e recupero.

La precisione è il rapporto fra numero di coppie correttamente etichettate e il numero totale di coppie potenzialmente colpevoli. Il recupero è il rapporto fra il numero di coppie correttamente etichettate e il numero totale di coppie

già classificate come colpevoli (vedi anche definizioni riportate al capitolo 2 della presente tesi).
Allo stesso modo per calcolare il limite massimo di ogni metrica, ogni coppia con il valore di metrica più basso è stata etichettata come potenzialmente non colpevole ed è stato identificato il valore più alto che otteneva il 100% di precisione.

Risultati. Tutte le coppie di prodotti erano colpevoli se contenevano cloni di dimensione più grande o uguale a 305 token. I risultati mostrano che *NCP* non è una metrica utile con cui identificare i cloni basati su riutilizzo di codice, mentre *MLC* e *LSim* sono metriche efficaci per l'individuazione di questo tipo di cloni.

Il plagio: una questione morale

Uno studio sull'etica degli studenti universitari

Gli scandali morali nel mondo del lavoro sono in crescente aumento, segno evidente che comportamenti immorali sono assunti da aziende e manager di tutto il mondo e che, quindi, è importante per la società che i valori etici e l'integrità siano insegnati a tutti fin dall'infanzia. Numerosi studi del passato rilevano che, tra gli studenti, le femmine hanno un'etica più profonda rispetto ai maschi; indagini effettuate nel mondo del lavoro evidenziano anche che non ci sono sostanziali differenze fra uomini e donne rispetto all'etica in azienda.
Rodzalan e Saat, rispettivamente ricercatrice e docente all'Università di Tecnologia della Malesia, hanno realizzato una ricerca che mira da un lato a rilevare l'etica degli studenti universitari nelle università pubbliche malesi (con particolare riferimento a quattro aspetti del processo morale: consapevolezza, giudizio, intenzione e comportamento) dall'altro ad identificare le differenze tra studenti di discipline accademiche e genere diversi.
L'interesse primario di questa ricerca è la comunità, che si aspetta dei futuri professionisti che lavorino rispettando i valori etici. La ricerca prende in considerazione quattro aspetti del procedimento morale. Il primo è la consapevolezza, ovvero l'abilità dell'individuo di riconoscere e interpretare

l'esistenza di un problema morale. Una volta individuato il problema, l'individuo dovrà giudicare qual è l'azione moralmente giusta, quindi il giudizio è la capacità di scegliere un'azione da compiere, dopo aver valutato la situazione e le possibili conseguenze. Il passo successivo nel processo morale è l'intenzione, ovvero il grado di impegno profuso per realizzare una particolare azione, scegliendo un valore morale piuttosto che un altro e assumendosi la responsabilità delle proprie azioni. Dopo aver deciso quale risultato si desidera ottenere, non resta che tradurre l'intenzione nel comportamento che meglio può consentire il raggiungimento di questo risultato. Per favorire comportamenti eticamente corretti nella società, è molto importante nutrire i più giovani con alti valori etici.

Hanno partecipato al questionario 2.000 studenti dell'ultimo anno di sei università pubbliche malesi; il questionario prevedeva una prima parte, nella quale si chiedevano all'intervistato dati demografici, e una seconda parte, nella quale si presentavano delle situazioni che coinvolgevano i valori etici e si chiedeva di esprimere il proprio gradimento rispetto alle reazioni proposte di fronte a queste situazioni.

I soggetti coinvolti, studenti di Scienze Sociali, Scienze e Ingegneria, erano per il 34,1% maschi e per il 65,9% femmine, principalmente di età inferiore ai 25 anni (87,7%) e di nazionalità malese (69,8%). Dalle risposte fornite è emerso che gli studenti non sono inclini ad agire in modo immorale e molti di essi hanno dichiarato che agire moralmente è più importante che ottenere buoni voti. Gli intervistati concordano che regolamenti e codici etici possono essere validi strumenti per portarli a rispettare i valori etici.

Complessivamente le femmine sono apparse più sensibili alla questione morale rispetto ai colleghi maschi ed è risultato anche che gli studenti di Ingegneria posseggono un livello etico più basso rispetto a quelli di Scienze Sociali e Scienze. Quest'ultima differenza dipende probabilmente dalla natura del piano didattico di Ingegneria, riconosciuto come uno dei più critici e duri che induce spesso gli studenti a sacrificare i propri principi etici per non rallentare la carriera universitaria.

Pur avendo un alto livello etico, quando nelle situazioni sono coinvolti i loro amici, gli studenti possono assumere atteggiamenti diversi; generalmente le femmine sono più solidali con i compagni e ci tengono a preservare le proprie relazioni amicali mentre i maschi sono più aggressivi e competitivi.

È importante che l'università insegni agli studenti a reagire sempre considerando l'etica, anche quando entreranno nel mondo del lavoro.

L'analisi del plagio dalla prospettiva dei social network

La competenza nel campo della programmazione è riconosciuta anche come un importante meccanismo per migliorare il pensiero astratto e algoritmico che potrà essere utile agli studenti durante l'università e anche nella loro vita professionale. Alcuni autori sostengono che ci siano studenti che si sono laureati in informatica senza aver acquisito competenze di programmazione.

L'inefficacia nell'insegnamento della programmazione è un problema conosciuto a livello internazionale, che riguarda diverse università in giro per il mondo. La programmazione non si può apprendere veramente se gli studenti non fanno abbastanza esperienza pratica.

Per risolvere questo problema e allo stesso tempo prevenire frodi, plagio e collusioni fra gli studenti, alcuni autori hanno proposto di adottare la pratica di esami controllati ovvero offrire agli studenti la possibilità di svolgere i loro esami usando un apposito tool di programmazione. Questa strategia ha riportato buoni risultati ma alcuni problemi persistono:

- non è possibile richiedere agli studenti di risolvere un problema significativo, come sarebbe dovuto data la natura dell'esame;
- questo strumento potrebbe far diminuire la motivazione tra gli studenti;
- lo strumento può non rispecchiare fedelmente l'ambiente di lavoro.

I compiti assegnati agli studenti non hanno grande importanza nella valutazione in programmazione perché:

- assegnare un grande numero di esercitazioni agli studenti crea un flusso di lavoro intenso per i docenti e il personale e fornisce un feedback;
- il contesto libero nel quale gli studenti realizzano le esercitazioni consente loro di incorrere in frode, plagio o collusione.

All'università ogni azione di plagio commessa dagli studenti è la più grande minaccia per il processo di apprendimento, con serie implicazioni per la comunità informatica in generale e universitaria in particolare. Ci sono diverse ragioni per le quali gli studenti compiono azioni di plagio, come le

scarse competenze, la mancanza di punizioni, il desiderio di ottenere un riconoscimento più alto senza doversi sforzare. Una volta che lo studente decide di copiare e trova una fonte, lo step successivo sarà cercare di nascondere l'azione compiuta. La forma più semplice per nascondere il plagio è cambiare la formattazione del codice sorgente. Sebbene molti docenti possano rilevare questi espedienti leggendo il codice, il rilevamento manuale di plagio è praticabile solo quando si gestisce un numero limitato di studenti. È per questo che sono stati studiati dei tool per il rilevamento automatico del plagio.

Gli algoritmi per il rilevamento del plagio possono essere classificati in due gruppi principali:

- *corpus-oriented:* i codici sospetti sono confrontati con altri memorizzati in un luogo noto (ad esempio un database) attraverso una funzione di similarità;
- *style-oriented:* il modo in cui ognuno utilizza il linguaggio è unico. Queste caratteristiche possono essere usate per identificare testi plagiati in due diversi modi: usando solamente il codice sospetto e ricercando al suo interno cambiamenti anomali di stile, oppure usando esempi precedentemente classificati per determinare la paternità di un codice.

L'approccio *corpus-oriented* è sempre stato criticato perché è limitato dalle dimensioni del deposito e non è adatto a gestire i casi in cui lo studente abbia copiato da un'azienda, da Internet o da qualunque altra risorsa esterna. Per minimizzare il plagio attraverso Internet diversi autori hanno proposto che le attività e gli strumenti di valutazione siano realizzati con caratteristiche che impediscono il plagio dall'inizio. Altri autori hanno proposto che i tool di rilevamento del plagio siano costruiti su estensioni dei servizi di ricerca disponibili su Internet, come le API di Google; questi tool sono in grado di estendere la ricerca di plagio di software oltre il limite del gruppo degli studenti di una stessa università.

Nessuna delle soluzioni proposte assicura la totale protezione contro tutte le tattiche di plagio del codice che gli studenti possono realizzare.

Anche se le università hanno a disposizione diversi tool per l'individuazione automatica del plagio, i professori spesso non sono disposti ad usarli, poiché, al di là degli automatismi, la decisione di accusare lo studente che

ha copiato e prendere dei provvedimenti contro di lui spetta al docente e richiede spesso uno sforzo ulteriore che il docente non vuole sostenere.

Tuttavia il numero straordinario di fallimenti nelle discipline informatiche, il numero crescente di studenti per ogni docente e la diminuzione drastica del numero di studenti che perseguono una laurea in discipline informatiche suggerisce che c'è più pressione sui docenti quanto sono meno severi nei confronti del plagio di software.

Luquini e Omar, docenti universitari in Brasile sostengono che è possibile analizzare il plagio dalla prospettiva dei social network. Questa prospettiva può portare a nuovi approfondimenti circa l'apprendimento e l'insegnamento usando i social network e trasformando il rilevamento del plagio in uno strumento più utile ed educativo.

Il modello del social network e le sue metriche possono rappresentare la struttura dei gruppi sociali. Si possono usare le metriche dei social network per produrre il comportamento desiderato nel lavoro di squadra. La disciplina dei social network riconosce che la topologia della rete offre opportunità e impone restrizioni agli individui all'interno del gruppo.

Si ipotizzi che, quando un esercizio assegnato è personalizzato per la classe e non può essere copiato direttamente da una risorsa esterna, gli studenti che vogliono copiare faranno affidamento sui compagni di classe per ottenere la copia del codice da copiare. La decisione del compagno di classe non è casuale ma riflette le abitudini sociali dello studente. L'ipotesi sostenuta in questo paper è che se un codice dello studente *A* è collegato da plagio al codice dello studente *B*, allora lo studente *A* è in relazione con lo studente *B*. Se definiamo la funzione di plagio *P* e la funzione di relazione *S*, allora ipotizziamo che $P(a,b) \rightarrow S(a,b)$.

Il tipo di questione proposta nell'ipotesi è uno standard per l'analisi dei social network in cui, dato un certo numero di attori, è necessario conoscere se due distinte relazioni sono collegate con alcune altre.

Per dimostrare questa ipotesi, prima di tutto è stato fatto un questionario sociometrico tra gli studenti per capire quali social network utilizzassero. Il questionario sociometrico è uno strumento per modellare i social network coinvolgendo le informazioni raccolte dalle discussioni all'interno di un gruppo.

Durante il corso sono state assegnate 16 esercitazioni; per ogni assegnazione è stata creata una matrice, nella quale ogni colonna riportava il nome dello studente e le righe riportavano la frequenza di token usata in ogni programma. Ad ogni matrice è stata applicata la procedura *LSI* (Latent Semantic Indexing) come funzione di similarità.

Il risultato è stato una matrice in cui ogni esercitazione di uno studente è stata confrontata con ogni altra per determinarne la similarità. Ciò ha prodotto la funzione *P* per ogni studente in ogni gruppo di esercitazioni.

Prima del test di correlazione la matrice che rappresentava i dati sociometrici è stata trasformata usando tre regole:

1. solo le relazioni reciproche sono rimaste nella matrice;
2. per valutare l'importanza della relazione per lo studente, la metrica del social network chiamata connettività è stata applicata alla matrice;
3. la matrice finale è stata normalizzata usando il valore maggiore tra gli elementi di riferimento.

I risultati dell'esperimento sono rappresentati nella figura 48. Ogni valore di correlazione *r* è presentato con la propria significatività o valore *p* per ogni esercitazione. Ulteriori relazioni sociali che non sono state catturate con il sociogramma potrebbero apportare dei cambiamenti alle informazioni ricavate.

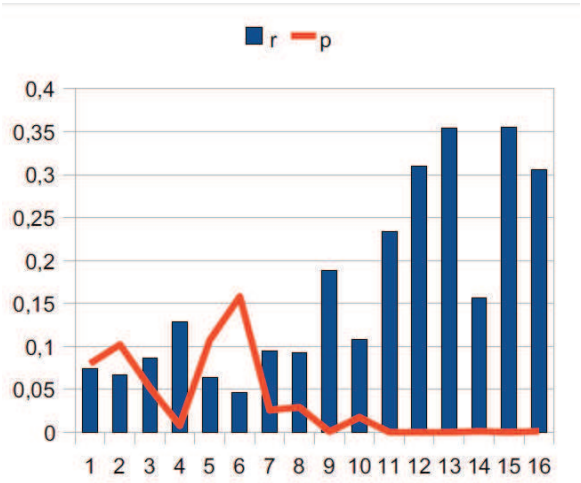

Figura 48 – Evoluzione della correlazione tra le esercitazioni. *r* è il fattore di correlazione e *p* è il valore di significatività o valore *p*

La correlazione attinge all'intera rete ma non può catturare le forti connessioni che gli studenti hanno. Il plagio può essere individuato analizzando la classe e il contesto sociale in cui si sviluppa e i codici

realizzati dagli studenti possono essere utilizzati come modello per analizzare i social network. Questa tecnica può essere ulteriormente sviluppata per creare meccanismi di intervento che evitino uno scontro diretto docente-studente e usino i social network per promuovere miglioramenti nell'apprendimento.

I giovani programmatori e le azioni di plagio

L'apprendimento della programmazione è un compito complesso e così i programmatori alle prime armi spesso tendono a copiare il codice sorgente mentre imparano la programmazione.

Molte istituzioni prevedono punizioni severe per i casi di plagio. Sono state sviluppati diversi ambienti di sviluppo per assistere i neo programmatori nell'apprendimento della materia. È stata condotta un'indagine alla Nelson Mandela Metropolitan University del Sud Africa; il prof. Dieter Vogts ne parla in un interessante articolo pubblicato nel 2009. Sono stati coinvolti gli studenti del primo anno del corso di programmazione, ai quali è stato proposto un questionario in forma anonima sul loro comportamento rispetto al plagio durante la loro attività universitaria.

Il questionario prevedeva diverse sezioni:

- informazioni biografiche;
- comportamento nei confronti del plagio in generale;
- presentazione di una serie di scenari, che gli intervistati dovevano classificare come comportamenti *accettabili* o *non accettabili*.

La popolazione coinvolta nel sondaggio era costituita per il 72% da maschi e per il 28% da femmine; solo il 38% degli intervistati (18 studenti su 47) ha dichiarato di parlare fluentemente l'inglese. Questi 18 soggetti sono stati classificati come sotto la media; i restanti 29 come sopra la media.

Il 40% dei soggetti intervistati ammette di aver commesso plagio durante il primo anno di corso.

Nelle tabelle 8 e 9 si evidenzia come gli intervistati hanno risposto rispettivamente alle domande «Approvi che gli altri copino?» e «Quanto codice copi abitualmente?»

	All (n=47)	Below Avg (n=19)	Above Avg (n=28)
Yes	0%	0%	0%
No	30%	26%	32%
Depends on situation	70%	74%	68%

Tabella 8 – Risposte alla domanda «Approvi che gli altri copino?»

	All (n=47)	Below Avg (n=19)	Above Avg (n=28)
Single line	26%	26%	25%
Couple lines to single method	57%	53%	61%
Couple methods to single class	15%	21%	11%
Couple of classes to files	2%	0%	3%

Tabella 9 – Risposta alla domanda «Quanto codice copi abitualmente?»

Nella tabella 10 sono elencati i motivi principali che inducono gli studenti a copiare; in cima all'elenco la mancata comprensione della richiesta del docente e l'incapacità a risolvere il problema.

All (n=47)	Rank
Did not understand the problem in the assignment	1
Did not know how solve a problem	2
Program not compiling and did not understand error messages	3
When accessing online help files, the information was not useful	4
Tried to complete assignment, but did not have enough time	5
Did not attempt assignment because of lack of time	6
Program development environment was complicated	7
Others expect success of student	8
Little perceived value of completing assignments	9
Wanted to see if would get caught plagiarizing	10
Did not want to complete an assignment	11

Tabella 10 – Elenco dei motivi per i quali gli studenti plagiano

Dalla tabella 11 si evince che il 59% dei soggetti intervistati ammette di aver osservato almeno 1 volta i colleghi che plagiavano codice.

Number of times observed plagiarism	All (n=47)
0 times	41%
1 time	18%
2-5 times	25%
6-10 times	11%
11 or more times	5%

Tabella 11 – Casi di plagio osservati

Agli studenti intervistati sono stati presentati 6 scenari:

S_1 – Ricevi un messaggio di errore dal compiler e non lo comprendi;

S_2 – Non ricordi come operano i costrutti del linguaggio;

S_3 – Non comprendi cosa richiede il compito assegnato dal docente;

S_4 – Non sai come risolvere il compito assegnato;

S_5 – Non hai abbastanza tempo per completare l'esercizio assegnato;

S_6 – Non te la senti di svolgere il compito assegnato

Per la maggioranza degli scenari, la prima azione realizzata dagli intervistati è chiedere aiuto al docente e ai compagni di corso; il plagio è considerato l'ultima azione, da compiere quando nessuna delle precedenti è stata sufficiente a risolvere il problema.

Anche quando ammette il plagio, la maggioranza degli studenti dichiara di non copiare più di qualche linea di codice o qualche metodo.

Il 76% degli studenti sopra la media (cioè quelli che conoscono meno bene la lingua inglese) ammettono che potrebbero copiare se si trovassero nello scenario 6, mentre di fronte allo stesso scenario sono molti di meno gli studenti sotto la media che copierebbero.

Questo dato suggerisce che gli studenti ricorrono al plagio quando hanno meno fiducia in se stessi; potrebbe essere utile tenerne conto per assisterli in maniera più efficace nel loro percorso universitario.

Un'indagine sulla conoscenza del plagio da parte degli studenti di informatica

Quanto è chiaro agli studenti il concetto di plagio del codice sorgente?

Gli studenti sono confusi su questo argomento soprattutto di fronte a particolari circostanze:

- quando devono scrivere codice *object-oriented*. Qui c'è un conflitto tra il riutilizzo legittimo di conoscenze e il desiderio di scrivere qualcosa di innovativo;
- quando si trovano a riutilizzare parti di codice scritto da loro stessi in precedenti esercitazioni e già presentato al docente.

Gli studenti hanno diverse fonti alle quali attingere per ottenere codice sorgente, come Internet, banche dati di codice, libri di testo; esiste una tale quantità di risorse, soprattutto online, che copiare diventa molto facile per gli studenti.

Nei corsi di informatica gli studenti fanno confusione con il concetto di plagio del codice sorgente; per testare il loro livello di comprensione è stata effettuata un'indagine via web che comprendeva una serie di quesiti, relativi a questi argomenti:

1. auto-plagio e riutilizzo del codice sorgente;
2. copia di testo da libri e risorse online;
3. furto o pagamento di altre persone per realizzare un lavoro;
4. collaborazione inappropriata;
5. conversione di codice in un altro linguaggio di programmazione;
6. falsificazione.

In base alle risposte fornite, ogni intervistato ha totalizzato un punteggio, indicativo di quanto bene il soggetto conoscesse il problema del plagio. Il questionario è stato presentato agli studenti dei dipartimenti di Informatica del Regno Unito e anche se non era richiesto agli intervistati di identificare la propria istituzione, il 77% di essi lo ha fatto ed è emerso che gli studenti appartenevano a 18 università del Regno Unito e a 3 con sede in altri paesi europei.

Oltre la metà degli studenti che hanno risposto, precisamente il 53% (410 soggetti), era costituita da studenti universitari iscritti a un corso di laurea in informatica di primo livello; 702 studenti su 770 intervistati provenivano dal Regno Unito o dall'Unione Europea.

L'indagine ha evidenziato che gli studenti di corsi di laurea connessi con le scienze economiche sono più sensibili al problema del plagio rispetto agli studenti di informatica.

Le risposte fornite sono state suddivise in due gruppi: il gruppo *A*, che include gli studenti che hanno totalizzato almeno 10 punti, e il gruppo *B*. Il gruppo *A* comprendeva 178 studenti che, con le loro risposte, hanno dimostrato di aver compreso abbastanza bene cosa si intende per plagio di codice sorgente, mentre gli studenti del gruppo *B* apparivano poco preparati sull'argomento.

Agli studenti è stato chiesto: a) se erano stati correttamente informati sul plagio dalla loro università (il 96,7% ha risposto sì); b) se sentivano di aver capito il problema (il 98,6% ha risposto sì); c) se avevano capito quali penalizzazioni comportava il plagio (il 92% ha risposto di aver compreso).

La consapevolezza dei rischi che si corrono compiendo azioni di plagio può costituire una motivazione in più per gli studenti a capire meglio il problema del plagio.

Gli scenari alla base dell'indagine sono stati raggruppati in base ai sei argomenti citati sopra; elenchiamo di seguito i risultati ottenuti.

1. Auto-plagio e riutilizzo del codice sorgente. La maggioranza degli studenti ha dichiarato di ritenere accettabile il comportamento di uno studente che riutilizza una parte di codice già presentata in precedenza per includerla in un nuovo lavoro; gli studenti non considerano questo scenario un caso di plagio.

2. Copia di testo da libri e risorse online. In questo caso sono emersi pareri discordanti in merito alla differenza tra «utilizzo del codice» e «furto di idee», perché non è sempre chiara la differenza fra copia e consultazione.

3. Furto o pagamento di altre persone per realizzare un lavoro. In questo caso la maggioranza degli studenti non ha avuto dubbi nel definire plagio i comportamenti presentati.

4. Collaborazione inappropriata. Con le loro risposte, gli studenti hanno dimostrato di non avere chiaro il confine tra discussione formativa e collusione inappropriata tra studenti e ciò suggerisce che gli studenti credono che lavorare insieme ai loro compagni sia comunque accettabile anche se le indicazioni del docente erano state altre.

5. Conversione di codice in un altro linguaggio di programmazione. Gli esempi presentati hanno evidenziato che gli studenti sanno che se il codice riutilizzato è opportunamente segnalato non si tratta di plagio.

6. Falsificazione. Di fronte al caso di uno studente che non riuscendo a far funzionare il proprio programma ne falsificava i risultati modificando l'output, gli studenti non hanno avuto dubbi nell'affermare che non si tratta di plagio.

Gli studenti hanno complessivamente abbastanza chiaro il problema del plagio ma ci sono zone grigie nel modo in cui i docenti informano gli studenti.

L'area di maggior confusione è quella che riguarda il riutilizzo di codice proprio già presentato in una precedente assegnazione. La seconda area grigia riguarda la definizione di riferimenti non corretti come caso di plagio; su questo aspetto occorre istruire meglio gli studenti. C'è parecchia confusione anche nel concetto di collaborazione, poiché spesso gli studenti ritengono in buona fede di poter collaborare coi colleghi pertanto non sono sempre concordi nel definire questo un caso di plagio.

Una buona formazione dovrebbe essere parte attiva dell'esperienza universitaria; il plagio è una questione complessa e il confine tra comportamenti utili allo sviluppo dello studente e comportamenti che possono essere puniti come frode non è sempre chiaro.

Il codice di condotta: uno strumento efficace per la prevenzione del plagio

Le linee guida e i regolamenti delle università raramente parlano di plagio e se lo fanno non affrontano la questione in modo sufficientemente dettagliato. Il plagio di software può avere un impatto importante sui risultati degli studenti. Ma quali sono le forme di riutilizzo consentite?

Nell'educazione e nell'apprendimento è giusto incoraggiare gli studenti a riutilizzare il lavoro e le idee di altri; ciò che non si deve incoraggiare è la presentazione del lavoro di altri come proprio. Questo è plagio e non è tollerato.

Generalmente le università definiscono delle regole proprie in relazione al plagio, perciò è necessario che gli studenti siano correttamente informati dei rischi che corrono quando compiono un atto di plagio. Il riutilizzo è un aspetto importante nella realizzazione di un software, tanto che se non si segue un rigoroso approccio per il riutilizzo di materiale esistente, la qualità del software che stiamo realizzando potrebbe essere compromessa.

In un articolo pubblicato nel 2009, Gibson ha tentato di definire un codice di condotta per il riutilizzo corretto del software, in modo da offrire agli studenti diversi vantaggi: una definizione chiara di plagio; uno strumento di tutela per lo studente che venga accusato ingiustamente; una struttura che supporti efficacemente i docenti nell'individuazione del plagio all'interno dei lavori presentati dagli studenti; un miglioramento nella qualità del software prodotto dagli studenti.

Si tratta di un codice di condotta: semplice da capire e applicare; consistente rispetto ai casi di plagio trattati; noto a tutti gli studenti.

Pratiche di riutilizzo non solo accettate ma anche molto utili allo studente per acquisire maggiori competenze sono le seguenti: composizione e aggregazione; eredità; uso dei template e delle caratteristiche generali; riutilizzo di patterns e architettura; riutilizzo di interfacce e specifica dei requisiti, test inclusi.

Lo studente dovrà ricordare che:

- tutto il software riutilizzato non dovrà trovarsi nello stesso file;
- per tutto il software riutilizzato dovranno essere riportati gli opportuni riconoscimenti e per tutto il software presentato lo studente dovrà evidenziare quali parti del codice sono riutilizzate e quali lui ha scritto autonomamente;
- tutto il software riutilizzato deve essere opportunamente testato;
- tutti gli studenti che vengono scoperti a plagiare software saranno puniti, come stabilito dal codice di condotta.

È sufficiente un codice di condotta per debellare il problema del plagio?

Le ricerche effettuate dagli autori dimostrano che, nel caso in cui gli studenti non erano stati chiaramente informati circa i rischi che correvano plagiando materiale, circa la metà degli studenti ha copiato materiale scaricandolo dal web.

Anche dopo che agli studenti è stato illustrato il codice di condotta, un numero significativo di essi ha continuato a copiare, anche se in percentuale più bassa. C'è di buono che molti degli studenti che avevano plagiato del codice sono stati scoperti a causa di errori presenti nei sorgenti che avevano trovato sul web; una lezione preziosa per loro.

L'etica informatica

Per quanto si parli di etica e di informatica ormai dalla meta degli anni '80, nessuno studio è riuscito a dare una definizione sintetica di etica dell'informatica.

Nel 2014 sulla rivista SIGCAS Computer & Society è comparsa un'indagine del Champlain College di Burlington che tentava di dare una definizione il più possibile sintetica e condivisa. Come la comunità informatica definisce l'etica informatica? Dal 1978 al 2013 sono state date almeno 30 definizioni diverse di etica informatica, la maggioranza delle quali è stata divulgata dopo il 2000. L'autore dell'articolo, Brian R. Hall, effettua un'analisi puntuale e precisa della letteratura esistente sull'argomento e ne ricava la seguente definizione:

> *L'etica informatica è l'insieme degli sforzi interdisciplinari e collaborativi di studiosi e professionisti compiuti per studiare metodicamente e influenzare praticamente i contributi e i costi di ogni opera informatica nella società globale.*

Quando si parla di etica nell'informatica ci si riferisce alla memorizzazione di dati personali, alla tutela o alla violazione del *copyright*, all'accesso alle informazioni, al plagio, alla pirateria.

L'etica informatica riguarda il mondo accademico, ma anche quello professionale e la società tutta.

Negli anni '80 si affermava che la tecnologia informatica, con la sua malleabilità logica, sarebbe stata rivoluzionaria; l'informatica rivoluzionaria lo è stata davvero e, poiché questa rivoluzione continua a trasformare il mondo, l'etica informatica rimane un concetto fondamentale per uno sviluppo equilibrato della società.

Riflessioni personali

L'argomento di questa tesi mi ha profondamente appassionato, poiché sono convinta che le tecnologie siano un mezzo straordinario per aiutarci a risolvere problemi in innumerevoli ambiti delle nostre vite e mi ha fatto piacere sapere che in tutto il mondo, sempre di più, si cerca di punire il plagio e di premiare la competenza in ambito informatico.

Nella mia (lunga) carriera universitaria mi sono purtroppo imbattuta spesso in casi di plagio, nel momento in cui, per realizzare elaborati da presentare

per poter sostenere un esame, molti colleghi studenti sceglievano la scorciatoia di condividere lo stesso elaborato, realizzato da uno di loro o da un esterno, per poter sostenere tutti gli esami nei tempi previsti dal piano didattico.

Personalmente mi fa piacere non aver mai seguito questa prassi: mi sono iscritta che già lavoravo ed ero più grande dei miei colleghi e la motivazione che mi ha spinta a farlo è stato il desiderio di imparare e difficilmente avrei potuto imparare senza capire ciò che facevo. È per questo che sono soddisfatta di non aver acquisito solo crediti ma anche un ricco bagaglio di conoscenze.

Percorso universitario... a ostacoli

Anche alla luce della letteratura che ho consultato per la stesura di questa tesi, non posso che condividere l'opinione predominante che giudica il plagio come una piaga del nostro tempo.

Nel mondo accademico il plagio determina frustrazione negli studenti che scelgono di capire piuttosto che di copiare, poiché, nel caso in cui l'inganno non venga scoperto, spesso chi ha copiato ottiene voti migliori in tempi più brevi, rispetto a chi lavora sodo.

Devo ammettere che in diverse occasioni ho assistito a interrogazioni da parte di docenti del mio corso di laurea che «smascheravano» colleghi colpevoli di aver copiato; mi piace pensare che nella prosecuzione delle loro carriere, universitarie e poi anche professionali, quei momenti di mortificazione pubblica abbiano rappresentato per quei colleghi un incentivo ad evitare di assumere nuovamente atteggiamenti scorretti.

Nella mia condizione di studentessa ho avuto modo di vedere coi miei occhi l'evoluzione del mondo universitario, un tempo destinato a coloro i quali potevano permettersi di investire diversi anni in formazione prima di accedere al mondo del lavoro, oggi divenuto formalmente più accessibile, grazie alla riforma cosiddetta del «3+2» e a quelle successive.

Tale accessibilità, purtroppo, ha determinato anche, a mio avviso, l'iscrizione di persone meno motivate di un tempo, che spesso scelgono comunque l'università incoraggiate dalla brevità dei percorsi, anche perché l'alternativa è l'ingresso in un mondo del lavoro che, soprattutto in Italia, non è né facile né entusiasmante.

Per favorire la conclusione dei percorsi universitari, negli ultimi anni le modalità di verifica dell'apprendimento sono diventate meno rigide e, se da un lato questa semplificazione ha alleggerito il lavoro degli studenti a tempo pieno (si pensi all'introduzione degli esami parziali, che permettono allo studente di organizzare meglio le proprie scadenze), dall'altro il percorso universitario rimane difficile per chi non può frequentare tutte le lezioni, soprattutto se studia materie scientifiche, che prevedono ore di laboratorio spesso indispensabili per comprendere fino in fondo quanto si studia.

In questa condizione di frustrazione, non stupisce che qualcuno scelga la via più breve, copiando laddove può, pur di non andare fuori corso.

Altro dato che è emerso dalla letteratura consultata è il fatto che molti studenti di ambiti scientifici si laureano fuori corso o addirittura abbandonano il percorso senza concluderlo.

Anche per corsi di laurea che formalmente non prevedono l'obbligo di frequenza, spesso è complicato preparare gli esami senza frequentare, inoltre non sempre i docenti sono disponibili nei confronti degli studenti non frequentanti, anzi, spesso c'è una sorta di diffidenza verso chi non frequenta, come se chi non frequenta lo facesse sempre e soltanto perché non è abbastanza motivato.

Il fatto molto concreto e banale è che non tutti possono permettersi di studiare senza lavorare, quindi a volte, quando più che un «percorso universitario» ci si trova di fronte un «percorso a ostacoli», rinunciare, seppur a malincuore, resta l'unica scelta possibile.

Dispiace constatare che viene meno, di fatto, la condizione richiesta dall'articolo 34 della nostra Costituzione, laddove si dichiara che «i capaci e meritevoli, anche se privi di mezzi, hanno diritto di raggiungere i gradi più alti degli studi». Questa realtà va di pari passo con la semplificazione della carriera universitaria di chi ha tempo da investire e denaro da spendere: chi non è motivato e volenteroso ma è figlio di genitori danarosi, può permettersi anche un percorso più lungo pur di ottenere l'ambito «pezzo di carta», che poi esibirà nel mondo del lavoro, corredandolo spesso con una robusta dose di arroganza, come se avere una laurea rendesse chi la possiede una persona necessariamente più preparata.

Molti studi dimostrano invece che spesso non è così: chi si laurea in informatica a volte non sa nulla o quasi di programmazione, come ho letto

in uno dei paper selezionati per la mia tesi. Dichiarazione che preoccupa, se si considera che l'abitudine al plagio è radicalmente diffusa anche nel mondo del lavoro, dove l'incompetenza può generare gravissimi danni non solo etici ed economici (si pensi ad esempio ad un software per elaborare immagini diagnostiche realizzato da programmatori incompetenti: potrebbe generare risultati distorti che nuocerebbero irrimediabilmente alla salute dei cittadini).

C'è da dire purtroppo che a volte anche frequentare non è sufficiente per capire: nella mia esperienza da studentessa non sono mancati i casi in cui a difettare di competenza sono stati i docenti, dimostrando di non riuscire a spiegare alcuni concetti senza invocare l'aiuto dei loro assistenti o addirittura degli stessi studenti.

Per fortuna ho incontrato anche più di un professore che ha svolto la sua attività con grande professionalità e con l'obiettivo principale di non lasciare indietro nessuno dei suoi studenti. Ho trovato spesso docenti disponibili a fornirmi tutte le informazioni che mi mancavano e solidali con la mia condizione di studente-lavoratore; anzi, uno di loro mi ha esplicitamente manifestato la sua ammirazione per la mia tenacia nel portare avanti anche la carriera universitaria nonostante un lavoro a tempo pieno (e, nel mio caso personale, anche nonostante una serie di problematiche familiari tutt'altro che facili da gestire; d'altronde, nell'arco di oltre un decennio, succedono inevitabilmente parecchie cose nella vita di una persona).

Buone regole per una buona società

Come molti degli studiosi che ho letto preparando il mio elaborato, sono anch'io convinta che il plagio si possa combattere efficacemente solo con l'introduzione e il rispetto di buone regole, sia nel mondo accademico che professionale.

Non credo di aver fatto nulla di speciale nel laurearmi, era importante per me concludere questo percorso e ora mi auguro che la laurea possa rappresentare un'ulteriore opportunità da spendere nel mondo del lavoro (anche perché, dato il momento storico tutt'altro che roseo, credo che in futuro serviranno tutte le «armi» possibili a chi, come me, è destinato a frequentare ancora per molti anni il mondo del lavoro).

Sono però molto contenta di aver avuto, preparando la mia tesi, l'occasione di scoprire che molti studiosi nel mondo lavorano alacremente per favorire nella società la diffusione di valori etici come il rispetto per il lavoro degli altri. Sono contenta anche di aver resistito tutti questi anni, perché l'esperienza universitaria, con tutte le sue contraddizioni e difficoltà, resta una palestra di vita importantissima, che mi ha permesso di coltivare il desiderio sano di imparare da chi ne sa più di me, costringendomi a riconoscere i miei limiti e allo stesso tempo i pregi degli altri.

Se non mi fossi sudata ogni minuto dei crediti acquisiti, tutto questo oggi non sarebbe mio. E di questo sono felice e orgogliosa.

Bibliografia

[1] Abd El-Wahed S. M., Elfatatry A., Abougabal M. S. – Alexandria University, Egypt (2009), "Detection of Plagiarism in Database Schemas Using Structural Fingerprints", *atti della ACS International Conference on Computer Systems and Applications (AICCSA) 2009*, pp. 787-790;

[2] Acampora G., Cosma G. – School of Science and Technology, Nottingham Trent University, UK (2015), "A Fuzzy-based Approach to Programming Language Independent Source-Code Plagiarism Detection", *atti della* 2015 *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Istanbul, 2-5 agosto 2015*, pp. 1-8;

[3] Ahtiainen A., Surakka S., Rahikainen M. – Helsinki University of Technology (2006), "Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises", *atti del Koli Calling 2006*, pp. 141-142;

[4] Ajmal O., Missen M. M. S., Hashmat T., Moosa M., Ali T. – Dept. of Computer Science & IT, The Islamia University of Bahawalpur (2013), "EPlag: A Two Layer Source Code Plagiarism Detection System", *atti della 2013 Eighth International Conference on Digital Information Management (ICDIM), 10-12 settembre 2013, Islamabad*, pp. 256-261;

[5] Aliprandi S. (2012), "CAPIRE IL COPYRIGHT – Percorso guidato nel diritto d'autore", *www.aliprandi.org/capire-copyright,* pp. 1-164;

[6] Baby J., Kannan T, Vinod P, Gopal V. – Department of Computer Science & Engineering, SCMS School of Engineering and Technology, India (2014), "Distance Indices for the Detection of Similarity in C programs", *atti della 2014 INTERNATIONAL CONFERENCE ON COMPUTATION OF POWER, ENERGY, INFORMATION AND COMMUNICATION (ICCPEIC)*, pp. 462-467;

[7] Bowyer K. W., Hall L. O. – Department of Computer Science and Engineering, University of South Florida (1999), "Experience Using "MOSS" to Detect Cheating On Programming Assignments", *atti della 29th ASEE/IEEE Frontiers in Education Conference, 10-13 novembre 1999 San Juan, Puerto Rico, Sezione 13b3*, pp. 18-22;

[8] Burrows S., Tahaghoghi S. M. M., Zobel J. – School of Computer Science and Information Technology, RMIT University, Melbourne (2006), "Efficient plagiarism detection for large code

repositories", *SOFTWARE-PRACTICE AND EXPERIENCE 2007, N. 37*, pp. 151-175;

[9] Buzzi M., Iglesias M., Rossi R. (2005), "Aspetti del Diritto d'Autore nella Società dell'Informazione: Licenze Open Source e Brevettabilità del SW", *IIT-TR 05/2005*, pp. 1-46;

[10] Carbonaro A. – Università di Bologna (a cura di) (2003-2004), *materiale didattico Corso di Programmazione A.A. 2003-2004, Corso di Laurea in Scienze dell'Informazione, Facoltà di Scienze MM.FF.NN.*;

[11] Caso R. (2008), "Forme di controllo delle informazioni digitali: il Digital Rights Management", *Quaderni del Dipartimento di Scienze Giuridiche, 70, Università degli Studi di Trento*, pp. 6-67;

[12] Chuda D., Navrat P. – IISE FIIT Slovak University of Technology, Bratislava, Slovakia (2010), "Support for checking plagiarism in e-learning", *Procedia Social and Behavioral Sciences, VOL. 2 2010,* pp. 3140-3144;

[13] Ciesielski V., Wu N., Tahaghoghi S. – RMIT University, Melbourne (2008), "Evolving Similarity Functions for Code Plagiarism Detection", *atti del GECCO '08 12-16 luglio 2008, Atlanta, Georgia USA*, pp. 1453-1460;

[14] Cosma G., Joy M. (2008), "Towards a Definition of Source-Code Plagiarism", *IEEE TRANSACTIONS ON EDUCATION, VOL. 51, N. 2, maggio 2008,* pp. 195-200;

[15] Cosma G. – P.A. College; Joy M. – University of Warwick, UK (2012), "An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis", *IEEE TRANSACTIONS ON COMPUTERS, VOL. 61, N. 3,* pp. 379-394;

[16] Cui B., Li J. – Beijing University of Posts and Telecommunications, Beijing, China; Guo T. – China Information Technology Security Evaluation Center, Beijing, China; Wang J. – Beijing Forestry University, Beijing, China; Ma D. – Beijing University of Posts and Telecommunications, Beijing, China (2010), "Code comparison system based on Abstract Syntax Tree", *atti del IC-BNMT2010,* pp. 668-673;

[17] Cunegatti B., Scorza G. (2001), "La tutela del software" *in Multimedialità e diritto d'autore, Edizioni Simone*;

[18] Dames, K. M. (2006), "Plagiarism: The New 'Piracy'", *Information Today – ProQuest Telecommunications, novembre 2006*, pp. 21-22;

[19] Dye J. (2007), "TO CATCH A Thief: Tools and Tips to Combat Digital Content Plagiarism", *EContent-ProQuest Telecommunications, settembre 2007*, pp. 32-37;

[20] Ganguly D., Jones G. J. F. – Centre for Global Intelligent Computing (CNGL), School of Computing, Dublin City University (2014), "DCU@FIRE-2014: An Information Retrieval Approach for Source Code Plagiarism Detection", *atti del Forum for*

Information Retrieval Evaluation (FIRE 2014), Bangalore, India, pp. 1-4;

[21] Gibson J. P. – Telecom & Management SudParis (2009), "Software Reuse and Plagiarism: A Code of Practice", *atti del ITiCSE '09, 6-9 luglio 2009, Parigi*, pp. 55-59;

[22] Gitchell D., Tran N. – Department of Computer Science, Wichita State University (1999) "Sim: A Utility For Detecting Similarity in Computer Programs", *atti del simposio SIGCSE '99, New Orleans, LA USA*, pp. 266-270;

[23] Hage J., Rademaker P. – Dept. of Information and Computing Sciences, Utrecht University, The Netherlands; van Vugt N. – School of Computer Science, Open Universiteit Nederland (2011), "Plagiarism detection for Java: a tool comparison", *atti della Conference CSERC 2011, 7-8 aprile 2011, Heerlen, The Netherlands*, pp. 33-46;

[24] Haider K. Z., Nawaz T., ud Din S., Javed A. – University of Engineering and Technology, Taxila, Pakistan (2010), "Efficient Source Code Plagiarism Identification Based on Greedy String Tilling", *IJCSNS International Journal of Computer Science and Network 204 Security, VOL.10, N.12*, pp. 204-210;

[25] Hall B. R. – Department of Software Technology, Champlain College, Burlington (2014), "A Synthesized Definition of Computer Ethics", *SIGCAS Computers & Society, VOL. 44, N. 3,* pp. 21-35;

[26] Imran N. – University of Central Florida (2010), "Electronic Media, Creativity and Plagiarism", *SIGCAS Computers & Society, VOL. 40, N. 4,* pp. 25-44;

[27] Jhi Y.-C. – Samsung SDS R&D Center, Seoul, Korea; Jia X. – Institute of Information Engineering, Chinese Academy of Sciences; Wang X. – Shape Security, Mountain View, CA; Zhu S., Liu P., Wu D. – Pennsylvania State University (2015), "Program Characterization Using Runtime Values and Its Application to Software Plagiarism Detection", *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 41, N. 9, settembre 2015*, pp. 925-943;

[28] Ji J.-H., Woo G., Cho H.-G. – Dept. of Computer Engineering, Pusan National University (2008), "A Plagiarism Detection Technique for Java Program Using Bytecode Analysis", *atti della Third 2008 International Conference on Convergence and Hybrid Information Technology,* pp. 1092-1098;

[29] Ji J., Park S., Woo G., Cho H. – Dept. of Computer Engineering, Pusan National University (2007), "Understanding the Evolution Process of Program Source for Investigating Software Authorship and Plagiarism", *ICDIM '07. 2nd International Conference, VOL. 1*, pp. 98-103;

[30] Joy M. – University of Warwick, Coventry, UK; Cosma G. – P.A. College, Larnaca, Cyprus; Yau J. Y.-K., Sinclair J. – University of Warwick, Coventry, UK (2011), "Source Code Plagiarism – A

Student Perspective", *IEEE TRANSACTIONS ON EDUCATION, VOL. 54, N. 1, febbraio 2011,* pp. 125-132;

[31] Juergens E., Deissenboeck F., Hummel B. – Institut für Informatik, Technische Universität München (2009), "CloneDetective – A Workbench for Clone Detection Research", *atti del ICSE'09, 16-24 maggio 2009, Vancouver, Canada*, pp. 603-606;

[32] Kikuchi H., Goto T., Wakatsuki M., Nishino T. – Graduate School of Informatics and Engineering, The University of Electro-Communications, Tokyo (2014), "A Source Code Plagiarism Detecting Method Using Alignment with Abstract Syntax Tree Elements", *atti della SNPD 2014, 30 giugno-2 luglio 2014, Las Vegas, USA,* pp. 1-6;

[33] Kodhai E. – Department of IT, SMVEC, Puducherry, India; Kanmani S., Kamatchi A., Radhika R., Vijaya Saranya B. – Department of IT, PEC, Puducherry, India (2010), "Detection of Type-1 and Type-2 Code Clones Using Textual Analysis and Metrics", *atti della 2010 International Conference on Recent Trends in Information, Telecommunication and Computing,* pp. 241-243;

[34] Kustanto C., Liem I. – School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia (2009), "Automatic source code plagiarism detection", *atti della 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing 2009,* pp. 481-486;

[35] Lazar F.-M., Banias O. – Politehnica University of Timisoara (2014), "Clone detection algorithm based on the Abstract Syntax Tree approach", *atti del 9th IEEE International Symposium on Applied Computational Intelligence and Informatics, 15-17 maggio 2014, Timişoara, Romania,* pp. 73-78;

[36] Le Nguyen T. T., Carbone A., Sheard J., Schuhmacher M. – Monash University, Caulfield East, Victoria (2013), "Integrating Source Code Plagiarism into a Virtual Learning Environment: Benefits for Students and Staff", *Australasian Computing Education (ACE) Conference 2013, Adelaide, Australia. Conferences in Research and Practice in Information Technology, Vol. 136,* pp. 155-164;

[37] Lesner B., Brixtel R., Bazin C., Bagan G. – GREYC (Groupe de recherche en informatique, image, automatique et instrumentation de Caen) (2010), "A Novel Framework to Detect Source Code Plagiarism: Now, Students Have to Work for Real!", *atti del SAC '10 (Symposium on Applied Computing) 22-26 marzo 2010, Sierre, Svizzera*, pp. 57-58;

[38] Luquini E. – Information System Dep. Faculdade Módulo, São Paulo, Brazil; Omar N. – Universidade Presbiteriana Mackenzie – UPM, São Paulo, Brazil (2011), "Programming Plagiarism as a Social Phenomenon – Rethinking Plagiarism Detection tools in the Programming Assessment Process", *atti della IEEE Global Engineering Education Conference (EDUCON) – "Learning*

Environments and Ecosystems in Engineering Education", 4-6 aprile 2010, Amman, Jordan, pp. 895-902;

[39] Maurer H., Kappe F., Zaka B. – Graz University of Technology, Austria (2006), "Plagiarism – A Survey", *Journal of Universal Computer Science, VOL. 12, N. 8,* pp. 1050-1084;

[40] McCart J. A., Jarman J. – Information Systems and Decision Sciences, College of Business Administration, University of South Florida (2008), "A Technological Tool to Detect Plagiarized Projects in Microsoft Access", *IEEE TRANSACTIONS ON EDUCATION, VOL. 51, N. 2, maggio 2008*, pp. 166-173;

[41] Monden A., Okahara S. – Nara Institute of Science and Technology, Japan; Manabe Y. – Osaka University; Matsumoto K. – Nara Institute of Science and Technology, Japan (2011), "Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations", *IEEE SOFTWARE, marzo-aprile 2011*, pp. 42-47;

[42] Muddu B., Asadullah A., Bhat V. – Infosys Labs, Bangalore, India (2013), "CPDP: A Robust Technique for Plagiarism Detection in Source Code", *atti del IWSC 2013, San Francisco, CA, USA*, pp. 39-45;

[43] Naik R. R., Landge M. B., Mahender C. N. – Dept. of CS & IT Dr. B.A.M.U., Aurangabad (2015), "A Review on Plagiarism Detection Tools", *International Journal of Computer Applications, VOL. 125, N. 11,* pp. 16-22;

[44] Nappi N. (2015), "La tutela del software nell'Unione Europea e il caso Oracle: una forte scossa al diritto d'autore", *rivista online FiLO Diritto, http://www.filodiritto.com, 3 agosto 2015;*

[45] Ngo M. N. – Singapore Institute of Technology, Singapore (2015), "Eliminating Plagiarism in Programming Courses through Assessment Design", *International Journal of Information and Education Technology, VOL. 6, N. 11,* pp. 873-879;

[46] Noto La Diega G. – Università degli Studi di Palermo (2013), "I programmi per elaboratore e i confini del diritto d'autore. La Corte di Giustizia nega la tutela a funzionalità, linguaggio di programmazione e formato dei file di dati", *GIURETA Rivista di Diritto dell'Economia, dei Trasporti e dell'Ambiente, VOL. XI, 2013*, pp. 69-96;

[47] Ohmann T. – School of Computer Science, University of Massachusetts; Rahal I. – Department of Computer Science, College of Saint Benedict, Saint John's University (2015), "Efficient clustering-based source code plagiarism detection using PIY", *Knowledge and Information Systems, VOL. 43, maggio 2015,* pp. 445-472;

[48] Park J. – SureSoft Technologies Inc., Korea; Son D., Kang D., Choi J. – Department of Computer Science, Dankook University, Korea; Jeon G. – Department of Computer Engineering, Korea Polytechnic University, Korea (2015), "Software Similarity

Analysis based on Dynamic Stack Usage Patterns", *atti del RACS '15, 9-12 ottobre 2015, Prague, Czech Republic*, pp. 285-290;

[49] Pascuzzi G., Caso R. (2010) "Il diritto d'autore dell'era digitale" *in Il diritto dell'era digitale, Bologna, Italia: Società Editrice Il Mulino, 2010*, pp. 199-249;

[50] Poon J. Y. H., Sugiyama K., – National University of Singapore; Tan Y. F. – KAI Square, Singapore; Kan M.-Y. – National University of Singapore (2012), "Instructor-Centric Source Code Plagiarism Detection and Plagiarism Corpus", *atti del ITiCSE'12, 3-5 luglio 2012, Haifa, Israel,* pp. 122-127;

[51] Pramono Y. W. T., Suhardi – School of Electrical Engineering and Informatics, Institut Teknologi Bandung (ITB), Indonesia (2014), "Detecting Plagiarism in Cross-Platform Mobile Applications – Case Study: Game Application Similarity in Symbian Platform and Android Platform", *atti della International Conference on Information Technology Systems and Innovation (ICITSI) 2014, 24-27 novembre 2014, Bandung-Bali,* pp. 159-164;

[52] Prechelt L., Malpohl G., Philippsen M. – Universität Karlsruhe, Germany (2000), "Finding plagiarisms among a set of programs with JPlag", *Journal of Universal Computer Science, 28 marzo 2000,* pp. 1-23;

[53] Rodzalan S. A., Saat M. M. – The Faculty of Management, University of Technology Malaysia (2015), "Ethics of Undergraduate Students: A Study in Malaysian Public Universities", *International Journal of Information and Education Technology, VOL. 6, N. 9*, pp. 672-678;

[54] Sharma S. – C-DAC Mohali; Sharma C. S., Tyagi V. – C-DAC Mumbai (2015), "Plagiarism Detection Tool 'Parikshak'", *atti della 2015 International Conference on Communication, Information & Computing Technology (ICCICT), 16-17 gennaio 2015, Mumbai, India*, pp. 1-7;

[55] Tian Z., Zheng Q., Liu T., Fan M. – Xi'an Jiaotong University (2013), "DKISB: Dynamic Key Instruction Sequence Birthmark for Software Plagiarism Detection", *atti delle 2013 IEEE International Conference on High Performance Computing and Communications e 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 619-627;

[56] Tian Z., Zheng Q., Liu T., Fan M., Zhuang E., Yang Z. – Xi'an Jiaotong University (2015), "Software Plagiarism Detection with Birthmarks Based on Dynamic Key Instruction Sequences", *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 41, N. 12, dicembre 2015*, pp. 1217-1235;

[57] Università di Bologna (2014), Codice etico e di comportamento, D.R. n° 1408 del 01/10/2014, Bollettino Ufficiale n° 93 del 31/10/2014;

[58] Vamplew P., Dermoudy J. – School of Computing, University of Tasmania (2005), "An Anti-Plagiarism Editor for Software Development Courses", *Australasian Computing Education (ACE)*

Conference 2005, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 42, pp. 83-90;

[59] Vogts D. – Nelson Mandela Metropolitan University, South Africa (2009), "Plagiarising of Source Code by Novice Programmers a 'Cry for Help'?", *atti del SAICSIT'09, 12-14 ottobre 2009, Riverside, Vanderbijlpark (South Africa),* pp. 141-149;

[60] Žáková K., Pištej J., Bisták P. – Slovak University of Technology, Faculty of Electrical Engineering and Information Technology (2013), "Online Tool for Student's Source Code Plagiarism Detection", *atti della ICETA 2013, 11th IEEE International Conference on Emerging eLearning Technologies and Applications, 24-25 ottobre 2013, Stary Smokovec, The High Tatras, Slovakia,* pp. 415-419;

[61] Zangara I. – Università di Catania (a cura di) (2009-2010), "La tutela del software e delle banche dati", *materiale didattico Corso di Legislazione Informatica A.A. 2009-2010, Corso di Laurea in Informatica, Facoltà di Scienze MM.FF.NN.*;

[62] Zhang F. – Penn State University, PA, USA; Jhi Y.-C. – Samsung SDS Co., Ltd., Korea; Wu D., Liu P., Zhu S. – Penn State University, PA, USA (2012), "A First Step Towards Algorithm Plagiarism Detection", *atti del ISSTA'12, 15-20 luglio 2012, Minneapolis, MN USA,* pp. 111-121.

Ringraziamenti

Voglio ringraziare innanzitutto la Professoressa Antonella Carbonaro, per avermi supportato e accompagnato in questi lunghi mesi di preparazione della tesi. La sua fiducia in me ha alimentato anche la mia, rendendo possibile un'impresa che temevo ormai irrealizzabile.

Voglio esprimere un sentito "grazie" anche ai tanti Professori, come Marco Antonio Boschetti, Gabriele D'Angelo, Aristide Mingozzi, Mirko Ravaioli, Paola Salomoni, che in questi anni di lunga e faticosa carriera universitaria hanno messo a mia disposizione non solo la loro profonda competenza ma anche il tempo, la pazienza o semplicemente una parola incoraggiante o una stretta di mano quando ne avevo bisogno.

In questo percorso ho vacillato spesso e ogni volta che stavo per arrendermi c'era una mano tesa verso di me alla quale ho potuto aggrapparmi per ripartire: era la mano di Milena Giacomoni, preziosa compagna di studio nella mia avventura universitaria. Posso affermare che non c'è nessun altro al mondo con cui io riesca a studiare così bene come con lei; grazie, amica mia!

Le ore in un giorno sono ventiquattro perciò studiare lavorando mi ha inevitabilmente costretta a sacrificare il tempo passato con la mia famiglia.

Voglio dire grazie alla mia mamma che finché ho vissuto con lei mi serviva pranzo-colazione-cena in camera e mi risparmiava ogni lavoro domestico quando ero sotto esame e grazie soprattutto per non aver mai smesso di credere che ce l'avrei fatta anche quando la fine del percorso sembrava irraggiungibile.

Grazie al mio babbo per avermi detto di non arrendermi; ci ripenso sempre quando sono in difficoltà e tutte le volte, per una inspiegabile magia, finisce che a non arrendermi ci riesco veramente.

Se non mi arrendo è anche grazie ai meravigliosi Amici che ho, soprattutto alle mie Amiche storiche: grazie a Deborah, perché possono passare mesi senza sentirci ma quando decidiamo di scambiarci una parola è sempre quella giusta; grazie a Emanuela che c'è sempre ad ascoltarmi, coccolarmi e incoraggiarmi anche quando non me lo merito; grazie a Claudia, con la

quale divido da anni non solo l'ufficio ma anche le gioie e i dolori, tanto che ormai le voglio bene come a una mamma.

Grazie ai miei adorati nipotini, Viola, Sofia Vittoria, Leonardo, Alessandro, Diego Vitaliano, Elia Vinicio, che con le loro coccole mi fanno sempre sentire una principessa anche se non lo sono e che ormai da tempo aspettavano che la zia diventasse uno "scienziato"!

Grazie a mia sorella Martina, perché per quanto folli e incomprensibili possano essere le mie scelte, so che su di lei posso contare sempre e comunque.

Grazie anche a mia sorella Morena, con la quale invece non riesco a condividere quasi nulla senza timore di essere giudicata, ma so che non è importante solo quello che si dice ma anche quello che si sente e in tanti suoi gesti, anche piccoli, il suo amore io lo sento.

E infine grazie a Ivan, che cammina al mio fianco tenendomi per mano da diciassette anni. Grazie per avermi supportato e sopportato anche in questo tortuoso percorso universitario benché fosse più mio che nostro. Senza le tue provvidenziali tazze di caffè, gli incoraggiamenti e i rimproveri, avrei senz'altro rinunciato molto tempo fa. Se è per me che avevo cominciato questo percorso, è per noi che ho tagliato il traguardo. Ne «La Fine è il mio inizio» Tiziano Terzani per descrivere l'amore racconta del palo al quale l'elefante si fa legare con un filo di seta; se l'elefante dà uno strattone può scappare quando vuole, ma non lo tira. Ha scelto di essere legato con un filo di seta a quel palo. Ecco, io ti ringrazio per aver scelto di essere legato a me con un filo di seta; non potrei sentirmi più amata e libera di così.