資訊系統侵權鑑定研究
A study of information systems infringement analysis and identification

郭忠義、張嘉賢
國立臺北科技大學資訊工程系
Department of Computer Science and Information Engineering,
National Taipei University of Technology
Email: { jykuo, t103598014}@ntut.edu.tw

摘　要

本文研究透過程式自動化的方式來比對程式碼是否有抄襲或剽竊之嫌疑，使用者只需要將兩個專案匯入至系統中，系統會將兩個程式專案中非程式碼部分過濾並把程式碼部分擷取出來，將無意義或是多餘程式碼過濾，並且使用者可以選擇專案中要比對的程式碼群組，系統會將此群組程式碼有使用到並擴散出去的程式碼一併納入進來，並將程式碼的註解、物件結構與演算法結構擷取出來，然後透過有效如 winnowing、Cosine Distance 與 Graph 結構比對...等的演算法來對各部分進行比對，最後將各部份的比對結果相似度與最後比對的結果相似度報表一併呈現給使用者，最後由相關單位認定兩方的專案是否有抄襲或是剽竊的。

關鍵詞：物件導向專案、程式剽竊、程式抄襲、抄襲偵測、程式相似度

Keyword：OOP project、Code Plagiarism、Plagiarism Detect、Code similarity

一、緒論

現在軟體發達的時代，並且對於原始碼也是受到智慧財產權的保護，程式設計人員受雇於公司所撰寫之程式碼為公司之產品，離開公司時不應將資料攜帶至其他公司或是散布出去，如有因為此事情而在法律上有糾紛，則需要第三方公正單位檢測是否有程式碼抄襲的嫌疑。

在電腦程式侵權鑑定有法律與資訊技術兩個部分，需要法律與資訊領域相關人員合作，本文研發一個程式對於侵權鑑定程式，美國聯邦法院提出了三步測試法判斷標準 [1]，1.抽離 :將程式解構、抽離之後抽取出應受保護之表達模式部分，以便之後過濾部分能判定是否有著作權保護適格。2.過濾 : 將非受到著作權保護的部分過濾掉，例如高度抽象之思想或概念等屬於公共財產，並未有原創性、特殊性...等公共領域之結構排除。3.比較 :將過濾後應受到著作權保護的部分進行是否有實質相近的比較。

對於大型專案往往程式碼物件數量及檔案數量高達幾千或幾萬個，而透過人工檢查需要花費大量時間，且因程式至其他公司使用，不太可能程式完全沒有變動，會因應其他需求而改動原有的功能，此部分可能會在人工檢測有遺漏，本論文設計並實作系統提供檢測程式碼之間的相似鑑定比較，找出程式碼是否有剽竊另一方的嫌疑部分，之後比較結果給予法律相關單位。

二、相關文獻探討

本文參考國外有關於程式碼抄襲相關國外文獻 [2, 3, 4, 5]，不同文獻對於程式碼檢測方法各不相同，一些文獻使用程式原始碼方式檢測程式碼抄襲 [2, 3, 4]，一些文獻使用將程式碼執行起來使用動態程式偵測的方式檢查相似度[5]，動態程式會產生指令碼，再對指令進行偵測，其方法非對原始程式碼進行檢測，但其概念可以給予原是程式碼檢測當作參考。

1.1 JPlag [2]:

由 Malpohl 開發的系統，透過網路的方式上傳之後進行比較，其執行方式會先去除程式碼中沒有意義的資訊，例如空白、註解，再將去除完的程式碼轉為一連串的 token，並且再將 token 之間互相比對，token 之間會有部分長度的資料相同，而相同長度越長的 token 就表示相似度越高，可是此比對使適用於基本型態，現在的專案使用的變數型態不一定只有基本型態，都會使用大量自定義或是內建涵式庫的物件，且剽竊來的原始碼如果經過名字 refator 就會無法比較出來。

2.2 Sid 系統 (A SOFTWARE INTEGRITY DIAGNOSIS SYSTEM)[3]:

其與 JPlag 一樣將原始碼轉換為 token 進行比較，其 token 主要抓取來源語言的標準關鍵字，例如: if、else、while、for、switch、運算符號、括號…等，之後再用 Lempel-Ziv 方法把重複程式碼部分進行壓縮，最後再使用 Kolmogorov complexity 方式來比對之間的相似度，可是這樣假如是將程式進行一些修改，例如演算法之間插入一些別的程式碼，或者是程式碼的流程順序進行稍為的互換，都有可能使得比對無法抓出其之間抄襲部分。

2.3 Software Plagiarism Detection with Birthmarks[4]:

Tamada、Okamoto、Nakamura、Monden 與 Matsumoto 利用偵測程式碼的軟體胎記來檢測出

程式，軟體本身會有一些特性，即使程式被修改其特性也很難修改，則此特性被稱為軟體的胎記，軟體胎記可能會有多個種類，有些使用特定 API 呼叫順序或呼叫次數來當作程式的胎記，而有些軟體胎記則是抓出使用系統 IO 的函數當作 Key，例如 C 語言中的 scanf、printf 或 openFile 之類的函數，並且再對所組成的指紋進行比對，Tamada 等人使用指紋的串列來做比對，相同的連續子字串越長的表示相似度越高，其缺點也是因為如果有程式有不同的需求而在中間加入一些程式碼或是程式碼交換位置就會有可能無法比對出來。

2.4 Software Plagiarism Detection with Birthmarks Based on Dynamic Key Instruction Sequences[6]:

由 Zhenzhou Tian 等人所提出的方法，其參考 Tamada[4]等人使用胎記的方式並改為動態程式，利用對程式在執行過程中產生的指令碼進行收集，之後將受程式的 input 所擴散影響到的指令碼進行擷取出來，並且為了減少程式碼運算量花費大量時間，過濾掉單純的記憶體搬運的程式碼，並且參考其他文獻[7, 8, 9]使用 k-gram 演算法將這些指令碼轉換後變成這個程式的動態胎記，之後透過 Cosine distance 演算法來比對程式之間的動態胎記，最後結果便為雙方程式的相似度。

三、背景知識

本論文會先探討一些常見的抄襲程式碼混淆避開被人工檢查常有的方法，並且介紹一些對於結構的相似度比對演算法以及使用範例，這些演算法會應用於本文不同部分的結構相似度比對。

3.1 典型抄襲攻擊方式:

不論是學生的作業或是大型的專案都會有人做出避免被發現是抄襲或是剽竊其他人的程式碼的事情，常有的方法有改變變數函數名稱:1.改變變數函數的名稱是最常有的，名稱只是給予程式開發員辨識用的，只更改名稱並不會影響到程式邏輯的功能，且有許多工具能夠輕易地改變許多變數名稱 2.改變程式碼位置:有些程式碼之間可以互換位置，而其並不會影響到程式的運作功能，例如下圖 1 中第 2、3 與第 4 行的程式，變數 a 與變數 b 都只是被使用而已，可以隨意的更改位置甚至合併，功能都不會受到影響，而且人工也很難辨識出這種問題，3.改變程式的撰寫方式:程式碼撰寫方式會依據每個人的使用習慣會有所不同，可能有人單行判斷式會寫括弧，而有些人會為了程式碼簡短而省略，程式運作不會受到影響。本論文辨識程式要能夠避免這些問題有效的找出是抄襲或是剽竊來的程式。

```
1  a = 10,b = 100,c = 2;
2  c += 11;
3  c += a;
4  c += b;
```

圖 1 簡單的變數運算程式碼

3.2 def 與 use:

對於程式的執行過程，主要就是變數的資料運算並且改變，而變數假如是被使用，則其表示為 use，而假如變數因受到運算而改變，其表示為 def，如果變數只是被 use 而已，如圖 1 中程式碼第 3 行變數 c 為 def 而變數 a 則為 use，當變數為 use 其變換順序並不會影響到其演算法結果，但如果是 def 的話，順序不同則有可能最後會改變演算法結果，在混淆程式碼當中，可以透過調換變數 use 的程式碼位置，人工在檢測的話可能會因調換位置差別很遠而難以檢查出來，因此本文系統要能夠檢測出此類問題。

3.3 Euclidean Distance 與 Cosine Distance:

Euclidean Distance 用於計算節點之間的距離公式，其有時候也有被用於計算兩個之間的相似度，不同維度表示不同種類的東西，而兩點之間距離越近表示其之間越相似，首先下面舉例一個範例來加以說明 Euclidean Distance 與 Cosine Distance 之間區別，假如有 3 個人 A、B、C，他們到水果行，分別購買了不同數量的水果(如下表 1)

表 1 三位購買不同數量的水果

	蘋果	鳳梨	香蕉	柳丁
A	3	1	2	0
B	3	0	2	4
C	0	3	1	0

要知道這三位之中 A 與其他兩位的購買行為哪個比較相近，依據 Euclidean Distance 的計算法如下公式 1 其中x_{1i}與x_{2i}為兩個比對對象同一種東西的數量，例如比較對象為 A 與 B，而計算結果為 A 與 B 距離為 4.123，A 與 C 的距離為 3.742，而 A 與 C 距離較近，所以認定 A 與 C 較相似。

$$\sqrt{\sum_{i=0}^{k}(x_{1i}-x_{2i})^2} \qquad (1)$$

可是 Euclidean Distance 在計算過程中，對於沒有購買的權重會比有購買的權重大，但是依據平常人的認知都會覺得 A 跟 B 都有購買鳳梨根香蕉所以他們比較相近，而不是覺得 A 與 C 都沒有購買柳丁而覺得他們比較相近，而 Cosine Distance 則解決了這樣的問題，其來自於三角函數的公式，Cosine Distance similarity 如下公式 2，x_{1i}與x_{2i}一樣為兩個比對對象同一種東西的數量，接著計算結果 A 與 B 的相似度為 0.645，A 與 C 的相似度為 0.422，所以此公式算出的結果 A 與 B 會比較相近，

且 Cosine Distance 方法也有被國外文獻[5, 10, 11]拿來對程式相似度偵測應用。

$$\frac{\sum_{i=1}^{k} x_{1i} * x_{2i}}{\sqrt{\sum_{i=1}^{k} {x_{1i}}^2} \times \sqrt{\sum_{i=1}^{k} {x_{2i}}^2}} \qquad (2)$$

3.4 winnowing 演算法[12]:

由 Schleimer 等人所提出的演算法，以用於指紋辨識演算法 k-gram 發展而來，其常用於文字的辨識，如果在文字之間插入一些文字或是調換位置，也能夠辨識出來，在大量網頁的資料的實驗中，也得到很不錯的結果，後來也被應用在 MOSS(Measure Of Software Similarity)上比較網路的文字資料，其會將許多的一長串的文字且成多個片段 gram 再用片段用來比較。演算法如下:

首先定義一些變數如表 2:

表 2 winnowing 演算法變數定義

t	字串大小
k	gram 的大小
w	window 的大小
g	gram 與 gram 之間的距離大小

第一步驟需先將空白去除，再去除完的字串分成多個長度為 k 的片段，將其稱為 gram，例如有一個句子[where there is a will there is a way]，將這個語句中之中所有的空白先去除成為[wherethereisawillthereisaway]，去除完空白的長度為 29 所以此語句的 t 為 29，而 gram 的長度 k 為自行定義，隨機假設定 k 為 3，將原始字串以每 3 個為一個組合如下圖 2 的方式分成多個 gram，則會得到如[whe]、[her]、[ere]…[way]的 gram，之後將每個 gram 進行雜湊，雜湊的方法可以依據不同的情況使用不同方式做雜湊，此以每個文字的編碼相加後除以 100 當作雜湊法，之後得到一串的數字列(24,19,16,31,21,21,19,16,20,21,17,31,21,32,21,32,28,21,19,16,20,21,17,31,13,37)。

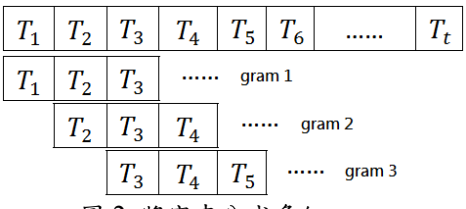

圖 2 將字串分成多組 gram

之後要選擇一個 windows 的大小 w，此值可以依據不同情況設定不同的大小，假設本論文選擇此 w 的大小為 4，將這些數字以長度為 4 的方式依照剛剛圖 2 中的方式再將資料分成多組資料(24,19,16,31)、(19,16,31,21)、(16,31,21,21)、…、(17,31,13,37)，接著將每組資料中選取出最小的值，而多組中的最小都是同一個 hash，則就將其只選擇一次，例如前三組之中都是最小的值都是同一個 hash 值為 16，就只選擇一次 16，最後選擇完後會得到數列(16,19,16,17,21,21,19,16,17,13)，此數列便是此句子的指紋，之後可以此指紋可依據不同情況使用不同的方法來進行比對。

四、鑑定方法設計

本系統為用於比對專案，專案中會有許多的檔案組成，可能有設定檔、執行檔、套件、圖片…等各式各樣檔案，不同語言又各會有許多不同種類的程式碼檔案組成，程式碼撰寫方式也會不同，程式碼撰寫的方式都有可能不同，所以本文架構在設計可以在抽換的部分可以對不同語言較容易的使用，程式架構流程如下:

4.1 程式執行測試

在雙方程式比對之前，需先對雙方的程式進行編譯並執行檢查，在雙方確認為所要比對得程式，並且檢查所提供的專案完整性，是否能夠正常的將程式執行起來，此舉避免被告因有抄襲疑慮而少提供一些關鍵的檔案資料，在雙方都確認無誤後再開始使用系統對程式進行檢測。

4.2 專案過濾與程式碼結構擷取

使用者可以選擇並匯入要比對的專案，專案中可能會有各式各樣的檔案種類，過濾掉非程式碼的檔案(例如:jpg、txt…之類檔案)，而不同程式語言所使用的副檔名也會不相同，並且使用針對此語言要先將資料撰寫方式統一化，在利用 Parser 將程式的結構轉為本系統能解讀得結構資料，並接著將資料集中管理。

4.3 設定檔案比較群組

因為專案在業界可能檔案及物件數量會非常大，兩專案之間每個檔案互相比較會花費非常大量的時間，在確定哪些檔案之行為較為相近下可以設定需要比對的檔案清單，來減少多餘的比對時間，使得整理花費時間減少，而如果使用者沒有設定就是全部直接互相比對。

4.4 物件結構與資料比對

之後物件比較子系統，依據使用者設定的比對清單進行比對，其中有比對幾個部分:1.註解比對:往往註解不同公司寫的註解應該不會有所相同，但是如果是在某公司離開直接使用原公司的程式直接修改，註解可能會完全相同，其有可能成為結果判斷的關鍵。2.變數比較:物件中會有私有物件，如物件成員變數，所以在比較物件是否相似會先比較它擁有的物件相似，擁有相似的成員變數，可以推論其使用目的可能相同。3.函數比較，程式流程主要部分就是函數，所以對於函數會互相比對，依據不同權重計算出物件之中的相似度，並將結果存於資料庫中。

4.5 結果報表呈現給使用者

專案之間的程式碼比對完成之後，過濾過相似度較低的結果，並將相似度較高的結果物件與物件之間相似度相關資料報表呈現給使用，其中包含檔案註解相似度、物件成員變數相似度與物件演算法結構相似度，最後由使用者給予相關單位進行判定是否有抄襲的行為。

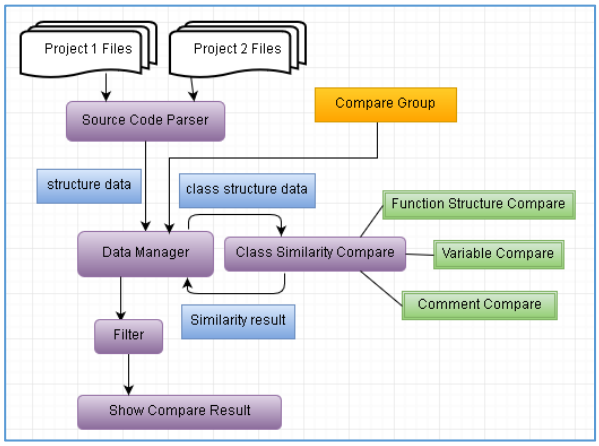

圖 3 系統流程圖

4.6 系統架構元件說明

系統在不同電腦執行時間會有所不同，而不同電腦的 CPU 核心數與記憶體也不相同，本系統設計可以分散式處理，在比對時會同時啟動多個不同的執行緒，所以可以依據不同電腦效能調整執行緒數量，而系統中的流程架構如上圖 3，各元件說明如下:

- Project Files:指檔案的專案，一個專案會有許多的程式碼檔案，將其匯入至系統中。
- Source Code Parser:因為不同語言解讀方式都會有所不同解讀方式，所以針對不同語言可以抽換此部分來使得系統可以針對不同的語言。
- Data Manager:資料集中管理，會存放物件的結構資料，其比對的相似度結果資料，因為有其他比對可能會使用到其他物件之間的相似度，也能在透過這邊進行查詢，避免重複運算耗費資源。
- Class Similarity Compare:此部分進行程式的比對，有分為物件擁有的物件或資料型態比對，物件之間函數的比對，檔案之間的註解比對，其中擁有的物件又分為內建物物件及自行設計的物件，內建的物件使用 Cosine distance 方式進行比對，而自創物件會再用 Class Similarity Compare 互相比對，再依照不同權重算出總相似度，註解比對的部分會使用 winnowing 演算法比對之間相似度；而函數結構比較方式會先將物件使用到的不同變數流程轉換為 Graph 結構，再將兩兩函數之間的 Graph 結構進行比對。
- Filter:比對結果會非常的大量，而顯示大量資料會使的使用者難以查看，所以會將相似度過低的資料過濾掉，只顯示相似度高的結果。
- Show Compare Result:將結果資料轉為使用者查看的介面，設計人性化的介面將比對結果顯示給使用者。

五、實作方法

本研究使用 Java 語言與 MySQL 資料庫實作，因執行過程有大量的對資料庫進行存取，所以針對資料庫 table 使用 index key 進行優化，當資料庫資料量龐大的時候，能夠更快速的查詢，這樣整體的執行速度加快。

5.1 程式語言 Parser 實作

系統需要先實作出針對不同語言的 Parser，來將資料結構爬成系統能夠理解的架構，而在 Parser 前置處理需要先將多餘的程式碼去除，例如:空白、註解、多餘的冒號進行去除，且程式的撰寫的 code standard 也不一樣，所以要先將程式進行正規化處理。

5.2 物件比對實作:

物件是由成員變數與函數所組成，並且軟體工程師在撰寫的時候會為了未來的維護而寫上註解，雖然註解並不會影響功能，但是每個人對於一個事情解釋方式應該並不會完全相同，且中文或英文對一個名詞也有許多的叫法，所以這部分也會進行比對。

5.2.1 物件成員變數比對

比對類別成員變數相似度:成員變數可能會有兩種狀況，其中一種為基本型態或是程式內建的物件型態，例如:int、String 或是 Java 中的 List，此部分使用 Cosine distance 的方法來比對函數，而如果成員變數為程式設計者自己定義則需再用遞迴的方式比對物件之間相似度，並且依照成員變數的數量而給予不同的權重，在整合算出物件的成員變數相似度。

5.2.2 物件成員函數比對

函數有回傳值與 input 的參數，依據不同用法回傳值與參數皆會不同，函數為程式主要運作的部分，而演算法為每個函數主要的部份，演算法是由許多變數之間的操作所組成，每一行的程式碼都是有操作不同的變數，最後給予演算法運算的結果。而變數常有為了物件導向原則，避免讓外部程式能夠只接存取物件的成員變數，所以會先將成員函數變成 private 型態，在經過 getter 與 setter 函數來存取資料，而許多的 getter 與 setter 並沒有運算的作用(如下圖 4)，而只當作資料取得的接口，所以本論文會先將空的建構元以及這類無運算的 getter 與 setter 進行排除，留下其他的函數來進行比對。

```
private String src_code;
public String getSrc_code() {
    return src_code;
}

public void setSrc_code(String src_code) {
    this.src_code = src_code;
}
```

圖 4 單純的 get 與 set 函數程式碼

首先把程式碼依據變數被賦予值或是被取其值賦予其他變數，如果只是被取值，則其可以交換位置也不會影響結果，所以一個節點會分支出多個節點，表示其之間可以同時被執行，不受順序的影響，然後產生 Graph 結構(如下圖 5)，之後再將函數結構中的每變數抽離出來，而一個函數中會有多個變數，所以對應多個 Graph 結構，如下圖 6 左邊為 x 變數的 Graph 結構，右邊為 y 變數的 Graph 結構。

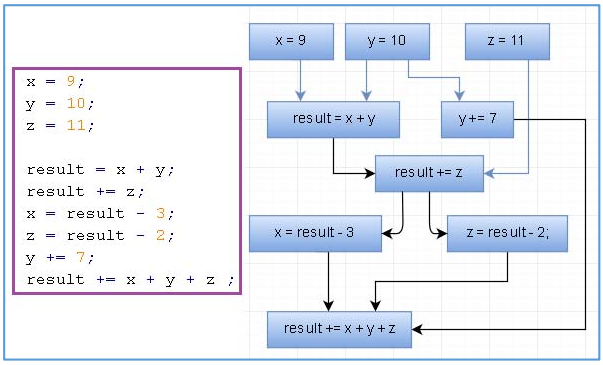

圖 5 程式碼轉為變數流程圖

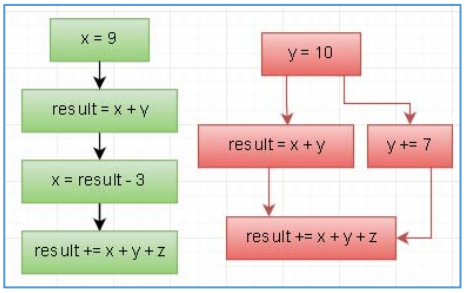

圖 6 變數流程圖抽出的 x 與 y 流程圖

而物件導向的程式碼並不只有此類單純的程式碼，還會有物件呼叫函數的部分，本論文將此動作以 call 代表，呼叫完的函數也有可能再被當參數引用，所以一個節點可能會有多個動作，如下圖 7，圖中 builder 變數先呼叫 ToStrnig 函數會取得一個字串，而這字串會提供給 database 這的變數使用，所以單一個節點可以再拆成兩個行為 call 與 use，將所有的節點依據其行為皆拆解 def、use 與 call，之後將相同階層或相鄰的節點的相同動作結合成一個節點，如下圖 8。

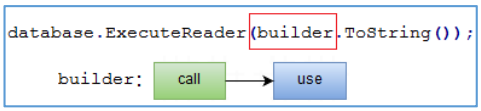

圖 7 變數 builder 的動作

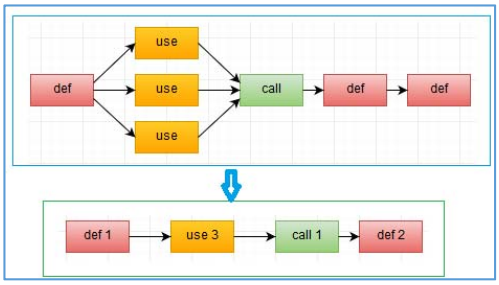

圖 8 節點的相同動作結合

接著將不同變數之間結合完的動作進行交叉比對，例如下圖兩個物件之間的變數，找出交叉比對中最大符合的值當作此兩個變數的結構相似度如下圖 9，variable 1 與 variable 2 之間動作節點交叉比較，分別為節點 1 對節點 1、節點 2 對節點 2 與節點 4 對節點 3，總和為(1/1+2/3+2/2)/4=0.667，此兩個變數的相似度便是 0.667。

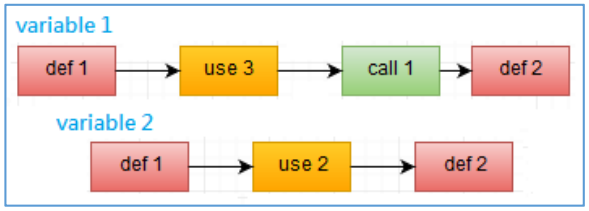

圖 9 兩變數的動作節點

算出演算法參數變數結構相似度、函數內宣告使用之變數結構相似度及回傳值相似度，並且給予回傳值、參數與函數內變數結構相似度不同的權重來算出物件函數成員函數的相似度，總相似度公式如下公式(3)，其中w_v、w_f、w_f分別為物件成員變數、物件函數、檔案註解之相似度權重(weight)，c_v、c_f、c_c分別為物件成員變數、物件函數、檔案註解之過濾後之數量(count)，s_v、s_f、s_c分別為物件成員變數、物件函數、檔案註解之過濾後之相似度(similarity)。

$$\frac{w_v*c_v*s_v+w_f*c_f*s_f+w_c*c_c*s_c}{w_v*c_v+w_f*c_f+w_c*c_c} \quad (3)$$

5.2.3 註解比對

註解比對部分，相似於對於語文的比對，而語文並沒有特別規定的寫法，只要程式人員能夠看得懂便可，並沒有規定要有動詞名詞或是主詞，所以本研究對於註解部分使用 winnowing 演算法來比對程式檔案之間的相似度。

5.3 實驗與結果

本論文目標需檢測出程式是否有剽竊或是抄襲的嫌疑，而與之比對為人工比對的方式做比較，

首先本論文使用已知有剽竊與抄襲嫌疑的兩個專案，兩個專案均為中大型專案，都是 C#網頁應用程式專案，專案的功能帳務系統，可以查看自己的帳務資料，而管理員可以管理許多功能，並且有公告訊息之功能，還有可以查看與運算帳務報表，會使用到資料庫進行存取，其中一個專案約 1000 個程式碼檔案，而另外一個專案約 850 個專案檔案，並先從中塞選並分類出相似行為的程式碼檔案，例如資料庫溝通程式碼檔案、UI 處理程式碼檔案…等，分成約 70 組的比對組別，每個組別有 2~15 個不等的數量，先透過人工比對的方式將有抄襲或剽竊的部分挑選出來，並且過濾出相似度高的檔案結果；而另外一邊使用本論文程式一樣對這些比對組別做檢驗，之後運算完產生結果，將此結果與人工比對的結果做比較，最後將分類出的人工檢測與本論文檢測檢測出來的結果之間的準確度比對，用以來驗證本論文方法是否能夠有效檢驗出剽竊或抄襲的嫌疑關鍵。

本論文實驗分為四個部分來做比較，分別為物件成員變數、物件函數、檔案註解比較以及結合以上三個結果之整合結果，其中每種比對皆會過濾掉無意義的比對，物件成員變數無意義的比對為兩方之物件其中一方無函數，物件函數無意義的比較為其中一方過濾後之函數數量為 0 之物件，而註解無意義之比對為兩方之比對檔案其中一方無註解，去除無意義比對部分，再針對剩下部分利用本論文程式與人工比對程式進行比較，來驗證本論文程式之精確度。

實驗中比較之雙方有瞟竊嫌疑情形分別有情況 1:直接複製使用並無任何變，情況 2:複製對方程式碼並且修改物件或變數名物件內之函數稱並沒有修改函數功能(如下圖 9)，情況 3:物件內之成員變數型態替換為其他資料型態或自創型態(如下圖 10)，情況 4:物件內之成員變數或函數位置互換，但功能並無修改，情況 5:物件函數因不同需求而編修原本函數部分內容(如下圖 11)。

```
class WebGridBinder{
        public WebGridBinder(){
        }
                private string _bindWebGridN
        public string BindWebGridName{
                get{
                        return _bindWebGridName;
                }
                set{
                        _bindWebGridName = value;
                }
        }
```

```
class WebDataViewBinder{
        public WebDataViewBinder(){
        }
                private string _bindWebDataVi
        public string BindWebDataViewName{
                get{
                        return _bindWebDataViewName;
                }
                set{
                        _bindWebDataViewName = value;
                }
        }
```

圖 9 只有修改物件與變數名稱

```
class ParameterDeclareAttribute:Attribute{
        private string _inName;
        private string _outName;
        private BizParameterDirection _direction;
        private Type _type;
        private bool _required;
```

```
class ParameterDeclareAttribute:Attribute{
        private string _inName;
        private string _outName;
        private ParameterDirection _direction;
        private Type _parameterType;
        private bool _required;
```

圖 10 改變為相似之資料型態

```
public virtual bool ValidateExpress(XmlNode node ){
        bool flag = false;
        if(node.Attributes["name"] != null )
                flag = true;
        return flag;
}
```

```
public bool Validate(XmlNode node ){
        bool flag = false;
        if(node.Attributes[ "name" ] != null && node.Attributes[ "circlePart" ]!= null )
                flag = true;
        return flag;
}
```

圖 11 依照需求而改變函數內容

在不同環境所需要執行的時間也互有所不同，以下提供本論文的執行環境做參考，本論文系統執行環境如下表 3，因本系統有使用資料庫，資料庫的資料數量大時在傳統硬碟與 SSD 所花費的時間也有所不同。

表 3 本論文系統執行環境

作業系統	Windows 8.1 64 位元
MySQL 版本	5.6.19 MySQL Community Server
硬碟	120GB SSD
CPU	Intel i7 4770
RAM	12GB

經過本論文之程式系統總花費時間如下表 4，雖然 CPU 只有 8 執行緒，但是當有執行緒需要等待 IO 的時候，可以先執行其他執行緒，所以實驗設定為 16 執行緒，並且也實驗了只使用單一執行緒所需要花費的時間，抄襲比對結果如下表 5，其中的總數量為已經過濾掉無意義之比對數量，正確數量為與人工比對之相同結果，如人工比對找出有瞟竊抄襲嫌疑本論文也檢測出有瞟竊抄襲嫌疑，或人工比對並無瞟竊抄襲嫌疑本論文程式檢測後也無瞟竊抄襲嫌疑，準確率為正確數量除以總數量。

表 4 實驗所花費時間

執行緒數量	1
A 專案 Parser 時間	46.064 秒
B 專案 Parser 時間	54.737 秒
比對時間	300.403 秒
總時間	401.204 秒
執行緒數量	16
A 專案 Parser 時間	15.07 秒
B 專案 Parser 時間	16.354 秒
比對時間	158.492 秒
總時間	189.916 秒

表 5 本論文程式比對實驗結果

比對部分	總數量	正確數量	準確度
物件成員變數	1036	968	93.43%
物件成員函數	1068	1022	95.69%
檔案註解	1264	1224	96.83%
完整比對	1397	1295	92.69%

由以上實驗結果可得到本論文方法中，註解比對演算法的成功率較高，且本研究發現此演算法因為編碼的關係對於中文註解的效果比英文註解的效果更好，而成員函數比對方法的準確度也比想像中高，函數中演算法有所更動也能夠有效的找出，即使替換成相似的物件也因延伸比對所以能夠有效的確認此替換物件是否相似，且使用執行緒執行速度雖然不會跟著執行緒數量成等比例的成長，但是在使用多執行緒進行可以加快比對的時間，在大量的比對時也會有明顯的幫助。

六、結論

本研究提出的方法，能夠協助程式碼智慧財產鑑定找出程式碼檔案間有程式碼剽竊嫌疑的目標，藉此幫助比對人員節省比對大量程式碼的時間，過濾掉比對結果差距過大的部分，並且也能找出人工檢測可能有遺漏的部分，使得侵權最後的審判結果能夠更有依據性。

致謝

本研究由科技部計畫 104-2221-E-027-116 所補助，特此感謝。

參考文獻

[1] 美國聯邦第二巡迴法院 928F. 2d, 693, 694 號，Computer Associates International, Inc. vs. Altai,Inc.案，1992。

[2] [Online]. Available: http://www.jplag.de/ 2016

[3] Andrei Zary Broder, "On the resemblance and containment of documents" Proceedings of Compression and Complexity of Sequences, pp. 21-29, 1997.

[4] Haruaki Tamada, Keiji Okamoto, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto, "Dynamic software birthmarks to detect the theft of windows applications," Proceedings of International Symposium on Future Software Technology, pp. 1-6, 2004.

[5] Haruaki Tamada, Keiji Okamoto, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto, "Design and evaluation of dynamic software birthmarks based on api calls," Nara Institute of Science and Technology, Technical Report, 2007.

[6] Zhenzhou Tian, Qinghua Zheng, Ting Liu, Ming Fan, Eryue Zhuang, Zijiang Yang, Senior Member, "Software Plagiarism Detection with Birthmarks Based on Dynamic Key Instruction Sequences," IEEE Transactions on Software Engineering, vol. 41, no. 12, pp. 1217-1235, 2015.

[7] Ginger Myles and Christian Collberg, "K-gram based software birthmarks," Proceedings of the ACM symposium on Applied computing, pp. 314–318, 2005.

[8] David Schuler, Valentin Dallmeier, and Christian Lindig, "A dynamic birthmark for Java," Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 274-283, 2007.

[9] Xinran Wang, Yoon-Chan Jhi, Sencun Zhu, and Peng Liu, "Detecting software theft via system call based birthmarks," Annual Computer Security Applications Conference, pp. 149-158, 2009.

[10] Zhenzhou Tian, Qinghua Zheng, Ming Fan, Eryue Zhuang, Haijun Wang and Ting Liu, "DKISB: Dynamic key instruction sequence birthmark for software plagiarism detection," IEEE International Conference on High Performance Computing and Communications, pp. 619-627, 2013.

[11] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk, "Detecting similar software applications," Proceedings of the 34th International Conference on Software Engineering, pp. 364-374, 2012.

[12] Saul Schleimer, Daniel Shawcross Wilkerson and Alex Aiken, "Winnowing: Local Algorithms for Document Fingerprinting", Proceedings of the ACM SIGMOD international conference on Management of Data, pp. 76-85, 2003.