輔仁管理評論

中華民國 108 年 9 月，第二十六卷第三期，27-50

程式抄襲源頭偵測之研究

蔡明志・陳思豪・曾宣瑜・胡俊之*

(收稿日期：107 年 12 月 25 日；第一次修正：108 年 02 月 12 日；
接受刊登：108 年 02 月 15 日)

摘要

過去研究多著重於程式的抄襲比對，僅有少數的研究對於抄襲源頭與抄襲群組進行尋找，但這些方法均不是針對學生作業抄襲的領域而設計。

本研究使用程式抄襲與複製偵測的文獻為基礎，將相似的作業結合為群組，並以程式”重要片段”的概念；利用重要片段的參考性、重要片段的傳遞性以及重要片段位於群內相似性與群間差異性，計算抄襲群組中的源頭可能性；最後再透過權重訓練模式訓練抄襲群組的權重，提升真正源頭被偵測的可能性。

實驗結果顯示：(1) 抄襲分數計算從一至五個群組的樣本，均可具有良好的源頭偵測率。(2)使用權重訓練模式能有效提升真實源頭的權重分數，並且降低非源頭的誤判率。(3)重要片段的三階段分數計算能有效形成組內分數差異，使得真實源頭更容易被偵測。

當抄襲群組與真實源頭被分析出來後，授課老師即可進一步的藉由抄襲群組比對學生間的同儕群組競合關係，以評估同學之間是否有抄襲或被抄襲的動機。

關鍵詞彙：抄襲群組，群組與源頭分析

壹·緒論

一、研究背景與動機

對於資訊相關科系的學生而言，程式語言的學習是非常關鍵的一環，這攸關學生未來在資訊領域的發展,因此程式設計為資訊相關科系的必修課程及先修科目，同時也是良好的邏輯訓練。親手實作是學習程式語言的唯一途徑。在學習的歷程中，程式作業佔有相當大的比重，主要原因不外乎是讓學生持續的練習，讓學生經由做中學的方式，加深程式設計的能力並訓練思考的邏輯能力。

實際上，程式語言教學的過程中，作業的抄襲一直以來都是眾所皆知的問題。相互抄襲容易除了造成班上的學習風氣低落外，更失去作業本身的意

* 作者簡介：蔡明志，輔仁大學資訊管理學系副教授；陳思豪，普奇科技有限公司軟體工程師；曾宣瑜，輔仁大學商學研究所博士生；胡俊之，輔仁大學資訊管理學系副教授。

義，使得學生在抄襲後，對於更進階的作業便無法親自動手實作程式設計，也影響學生個人的未來競爭力。是以嚇阻抄襲的歪風有其必要性，更能進一步提昇課程的學習成效。

目前並無商業軟體能夠處理程式作業的抄襲，僅有少數提供學術研究用的程式比對系統 如 JPlag（Prechelt & Malpohl, 2002）、YAP3（Michael, 1996）以及 PDetect（Moussiades & Vakali, 2005）。然而這些系統的共通點是不針對抄襲的情況進行解釋，僅有簡單的數字告知作業之間的相似度，老師必須再查看原始碼以判斷是否確為抄襲。除此之外，單一的抄襲判斷演算法也相當容易誤判，且現行的比對僅止於判讀出抄襲的可能，卻未能提供額外資訊以搭配同學之間的競爭關係或合作關係，作為抄襲與否的判斷。

因此，本研究有別於現行程式僅止於提供相似度分數的作法，採用混合演算法搭配權重設計以計算樣本相似度，再利用分群方法以分析**抄襲群組與抄襲源頭**，以供授課老師配合學生之**同儕群組關係**進行交叉比對，作為是否需要對學生進行訪談的參考。

二、研究問題與目的

早期的程式抄襲比對多半是人工比對，不但需要耗費相當大的時間，當受測人數眾多時，更不是一、二位助教就有辦法達成任務。隨著科技的進步，文字比對已經具備相對成熟的工具，然而，程式比對卻有著較為缺乏的研究。因此，如何利用資訊科技達成程式比對，減少人力負荷，即為本研究的首要目的。

過去研究多著重於程式的抄襲比對上，但較少專門針對學生作業抄襲的領域進行研究。另外，雖然有抄襲比對部份方法具權重設計，然而，隨著權重的提升，可能增加比對分數，使得比對結果傾向於抄襲，卻也增加了誤判的機會；反之，調降權重則是使得實際抄襲的作業未能偵測出。

因此，利用高權重值並輔以群組關係資訊以作為抄襲與否的參考為主要目的。而描繪出群組關係的首要工作為找出源頭，因此研究問題為：**如何以有效的方法偵測出程式抄襲源頭，再進一步分析出群組關係**。

經由以上所述，本研究主要為利用或改善現有技術，進行以下步驟：

a 建立學生的抄襲權重值。

b 尋找群組中的抄襲源頭。

c 分析作業抄襲的群組關係。

d 重覆以上步驟以找出最適分群模式。

三、研究限制

在學生作業的比對上，雖然本研究提出的比對架構並無程式語言限制，然而實驗用之關鍵詞典採用 JAVA 語法為主，因此主要的作業比對限制為 JAVA 程式語言，經實驗結果在該程式語言類型的作業比對上有良好效果。

然而，不同的程式語言有不同的語法特性，並非 JAVA 程式語言比對效果明顯，即可適用於其他語言，因此，除了需重新制定關鍵詞典外，仍需進一步進行類似的實驗，以驗證其有效性，才能作為不同程式語言在作業抄襲比對上的應用。

貳·文獻探討

依據本研究的問題與目的：**如何以有效的方法偵測出程式抄襲源頭，再進一步分析出群組關係**，因此文獻探討方向即以(1)群組特性與抄襲行為：引述單一抄襲偵測的問題以及群組關係在抄襲判斷的重要性，(2)抄襲偵測技術：整理現有方法及分析優缺點，(3)抄襲群組分析與源頭演化偵測：源頭的偵測方法、群組關係之歸類方式及其可行性評估。

一、群組特性與抄襲行為

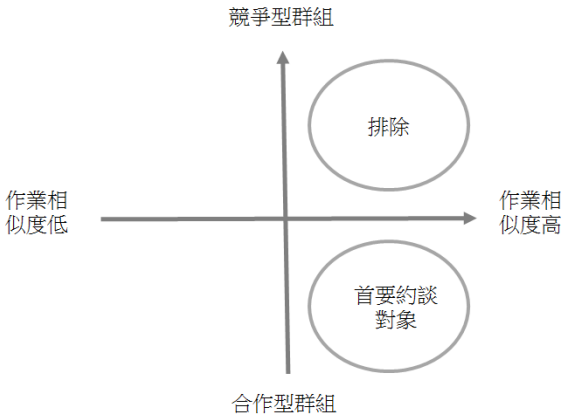
作業抄襲是各類課程普遍存在的問題，不但影響班上的學習風氣，也影響老師的教學成效，更影響學生的未來競爭力，因此抄襲的防制一直是教育作為的一環。現階段的抄襲偵測多為文字作業的抄襲偵測，對於程式抄襲應用的研究則較少，然而，如表一，抄襲偵測的準確率與誤判率是一體兩面的問題，隨著準確率的提昇，誤判率必然跟著提升。

表一 抄襲的偵測與誤判

	偵測為未抄襲	偵測為抄襲
實際為抄襲	(型一誤判) 抄襲者未被找出	(偵測正確)
實際未抄襲	(偵測正確)	(型二誤判) 未抄襲者被誤以為抄襲

資料來源：本研究整理

多數的抄襲技術單一面向分析是否有抄襲，若輔以行為科學面向解釋此一行為，群組關係對組員行為有一定的影響力。Lweicki & Bunker (1996)指出，群組認同感對於群組成員信任感的建立，形成凝聚力與互動，其中包含學習與共同解決問題。藉由群組特性的辨識，可作為取得抄襲分數後的進一步行為依據。而黃政傑、張嘉育(2010)的整理也說明了競爭模式下的教育行為將形成學生分級，也許是說，藉由群組成員與群組特性的辨識，可進一步改善誤判率。如圖一，即使是有著較高的作業相似度，也不致於被誤會為抄襲行為。例如，當群組競爭性時高，則學生為使個人分數優於其他同學，通常不願提供競爭對象抄襲自己的作業，因此在高競爭群組下即可視為排除對象。



圖一 以群組特性輔以減低型二誤判

資料來源：本研究整理

二、抄襲偵測技術

抄襲偵測技術（Plagiarism Detection）與複製偵測（Clone Detection）領域的技術多有重疊，尤其以字串、符號、樹狀結構、PDG（Program Dependency Graph）為基礎的方法都有相同的技術概念。兩者差別在複製預測致力於減少程式的原始碼重複性；抄襲偵測則是找出刻意偽裝的複製內容（Roy & Cordy, 2007）。程式抄襲指複製或修改他人的程式，產生其它功能相似但看似不同的版本（Parker & Hamblen, 1989）。Chanchal, et al.（2009）依過去文獻將程式複製歸納為以下四個種類：

類型 1：空白字元、排版與註解上的修改。

類型 2：識別字、文字、型態與類型 1 的修改。

類型 3：敘述的交換、新增或刪除，以及類型 1、2 的修改。

類型 4：程式以不同的語法進行實作，但具有相同的功能。

以上四個項目針對隱藏方式的困難程度進行區分，然而學生的作業中不考慮類型 4 的修改，因為已經達到作業寫作的目的，學生在修改的同時亦內化程式設計的邏輯與知識。程式抄襲的領域中，Whale（1990）將抄襲的類型區分為以下七個種類：

(a) 改變註解、格式、識別字、資料型態

(b) 改變運算式的運算元順序

(c) 改變相同結果的運算式

(d) 增加多餘的敘述與變數

(e) 改變無相依性敘述的順序

(f) 改變迴圈或選擇敘述的結構

(g) 以子程序的主體敘述取代子程序呼叫

上述分別對應類型 1 與類型 2 第{ 1 }項，類型 3 第{ 2, 3, 4, 5, 6, 7 }項，由此能推論抄襲偵測主要為前三種程式複製類型。

現有文獻的抄襲預測技術大約分為指紋為基礎（fingerprint-based）、字串比對為基礎（string matching-based）、樹狀比對為基礎（tree matching-based）等三種（Mozgovoy, 2006），其中字串比對能進行符號化轉換（Joy & Luck, 1999），因此符號化比對（token matching-based）可視為字串比對的子集合，它的能防止變數名稱的抄襲修改。以指紋為基礎的研究又能歸納為字串比對。本研究主要以剖析器將程式原始檔轉換為樹狀結構，經由比對程序取得程式之間的相似度，再利用相似度資訊建立抄襲群組以計算作業的分數，此分數能表示源頭的可能性。以下將透過「樹狀比對」、「符號比對」以及「字串比對」等三個技術探討程式比對的主要內容。

(一) 樹狀比對

樹狀比對在抄襲預測上有較好的效果，但它也相對難以實作（Ji, et al., 2007）。以樹狀比對為基礎的方法，一般利用剖析器將程式原始碼轉換為抽象語法樹（AST）或剖析樹（Parse Tree），接著直接運用內容或轉換為其它的特定格式（Yang, 1991; Baxter, et al., 1998）。Yang（1991）是較早提出依句法

的差異建立剖析樹方法的學者。在樹的比對方式上以動態規劃為基礎，提出最長共同子序列（Longest Common Subsequence）的概念，透過 STM（Simple Tree Match）演算法實作兩程式的比較。但不幸的，最長共同子序列的觀念無法處理無相依性結構的對調（游景翔，2007），而 GST（Greedy String Tiling）演算法最早是由 Wise（1993）提出，能排除結構的調換情況，JPlag 與 YAP3 等知名的抄襲系統皆使用此演算法（Prechelt & Malpohl, 2002; Michael, 1996）。GST 演算法的概念是將一字串進行最佳長度的選擇，從長度高於門檻值的序列集合挑出最佳解，將它視為符合的序列組合；接著再次以同樣方法，找出尚未被選中的序列，持續上述步驟直到候選序列不再超過序列長度門檻值，最後符合的序列集合即是兩字串的共同子序列。

以抄襲類型的前三項而言，因為程式之間的敘述會因變數、運算元或函數名稱的不同，而無法斷定彼此是否發生抄襲，因此多數的抄襲文獻會對程式原始碼進行前處理，將這些容易影響比對結果的敘述符號化，降低干擾的程度。

(二) 符號比對

符號比對多用在程式作業抄襲時，偵測變數名稱與迴圈型態的交換（Mozgovoy, et al., 2007）。Joy & Luck（1999）將詞彙修改的方式定義為（1）修改、加入或刪除註解（2）交換格式（3）修改識別字的名稱以及（4）改變行數等，符號比對最大的優勢是防止單純的字串改變影響偵測的結果，並且轉換的符號能夠用來表示整支程式的結構。JPlag 是典型的符號比對系統，利用剖析器將程式原始碼轉換為符號，同時忽略空白字元、註解與識別字名稱等（Prechelt & Malpohl, 2002）。JPlag 目前具有多種程式語言實作版本（C, C++, JAVA, C#, Python, …等），能將不同的程式語言轉換為通用的字串符號增加擴充的能力 。CCFinder（Kamiya, et al., 2002）利用兩階段將內容正規化，分別是語法修飾詞與識別字的代換。

第一階段的轉換包含規則化的識別字與標示符號的結構，第二階段的轉換是將參數取代（資料型態、變數與物件名稱、數值等），例如：int i = 0; ➔ $p $p = $p;。

(三) 字串比對

字串的比對能想像為程式序列之間的比對方法，多數的文獻會將原始文字進行正規化、過濾或轉換，一般常見的情況是移除註解、空白字元或對特定格式的正規化（Ducasse, et al., 2006），如表二所示。

表二 特定格式的正規化範例

語言的元素	範例	
字母文字	“Jerry”	“. . .”
字母字元	‘S’	‘.’
十進位數字	0.23	1.0
函式名稱	call（）	fuction（）

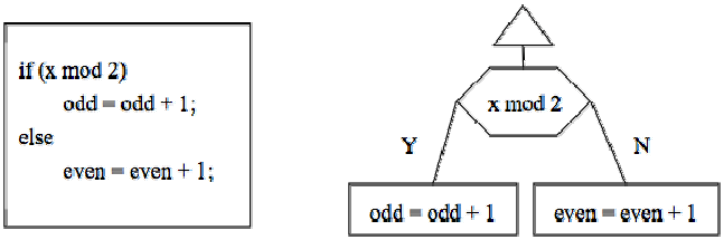
資料來源： 本研究整理

Howard（1993）利用 Karp-Rabin 指紋演算法計算原始碼中子字串的簽名，再將這些簽名以一行的方式進行計算，用來識別子字串的符合情況。Baker（1999）的 Dup 系統是以一行文字為基本單位，將註解、空白等符號移除後，再將所有內容組合為字串序列並取得每一行的雜湊值，最後再使用後綴樹（Suffix Tree）演算法進行比對。

符號比對的概念與字串比對相同，它們都以字串相似度演算法進行分析，目前多以兩個方向為主，分別是最長共同子字串（Longest Common Substring）與最長共同子序列，其中共同子字串表示連續的字元，它與共同子序列最大的分別在於是否可容忍間隙（gaps）。常用的共同子字串演算法為後綴樹演算法（Kamiya, et al., 2002），其中 Ukkonen（1995）學者所建立的後綴樹演算法具有線性的建立時間與佔用空間。然而不影響程式目的的原始碼被加入後，共同子字串會造成相當大的誤差，例如字串一的 abcabc 與字串二的 gbcabd，它們的最長共同子字串是 bcab，若在字串一中間多加一個字元形成 abcdabc，則最長共同子字串將成為 bc 或 ab；此時若使用可容許間隙的共同子序列，則結果依然保有 bcab。

(四) 混合式比對

Tairas & Gray（2006）與 Rainer, et al.（2006）兩篇文獻階在同一時期分別發表應用後綴樹技術比較抽象語法樹的研究，後者的研究將抽象語法樹的節點序列化後，再以後綴樹演算法進行比對。雖然具有良好的時間與空間複雜度，卻無法比對程式片段前後調換的情況。Greenan（2005）使用序列比對演算法找出方法（函數）層次的複製，提供結構化的複製偵測。Jiang 等學者利用向量方法，在抽象語法樹中計算結構化資訊的相似性。游景翔（2007）使用 Belkhouche , et al.（2004）提出的簡易樹狀表示法建構程式樹，如圖二所示。



圖二 Belkhouche 樹狀表示法範例

資料來源：游景翔（2007）

其中圖二（a）轉換為樹之後會成為圖二（b）的結構，好處是結構的比對成立時，才會對節點間的循序敘述比對，能避免所有敘述相互比對的時間，缺點是只針對迴圈與選擇建立結構，其它皆視為循序敘述，節點的比對上也只有採用最長共同子序列演算法，無法預測函數對調的結果。

(五) 其它方法

MOSS（Schleimer, et al., 2003）將程式切分為 k 個片段，再對每個 k 長度的片段取雜湊值，接著選取當中部份的雜湊碼當作其程式原始碼的指紋（fingerprint），目的是能快速比較兩程式，它改良傳統指紋系統的缺點，但程式的片段會因部份修改，而讓整個指紋改變，因此無法表示完整的程式內容。

各個技術分析與其優缺點比較如下表：

表三 現行比對技術的整理其優缺點比較

比對技術	方式	優點	缺點
樹狀比對	以程式碼建立成剖析樹，以利比對進行	抄襲預測上有較好的效果	需額外前處理、較難實作
符號比對	以符號相似度進行分析	實作簡易、可防止單純的字串改變影響偵測的結果	雖精準度提昇，但連帶造成較高的誤判率
字串比對	以字串相似度進行分析	實作簡易	不影響程式目的的原始碼被加入後，共同子字串會造成相當大的誤差
混合式比對	以後綴樹演算法進行比對	相較於樹狀比對，後綴樹以先結構再順序比對的方式可有效縮短時間	需額外前處理、無法預測函數對調的結果

資料來源： 本研究整理

根據表三的整理，混合式比對的方法繼承樹狀比對的預測效果，並改善了樹狀比對較為耗時的狀況，唯在函數對調的缺失部份，符號比對則能夠提供較為準確的效果。

因此，本研究即以混合式比對技術為基礎，並結合符號比對的前處理，於順序比對階段再搭配符號比對作為混合比對演算法的強化。實作上則以 Java 程式語言進行，增加 Java 支援的結構，如例外處理的 try、方法（method）與類別（class）定義等結構。

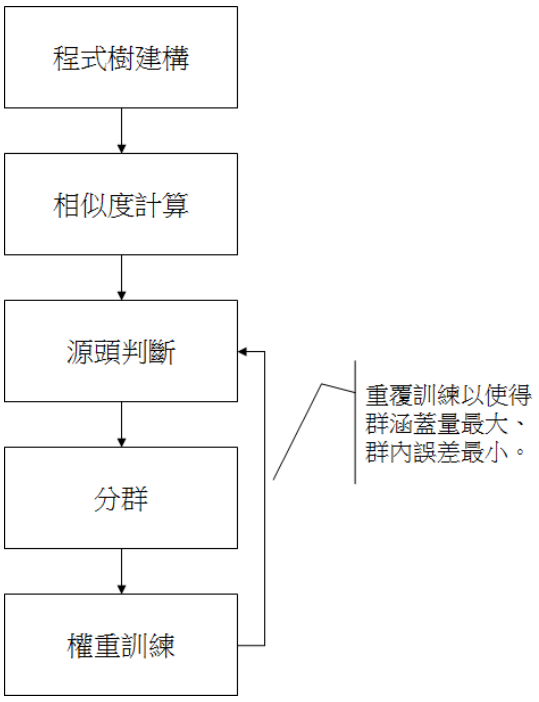
三、抄襲群組分析與源頭演化偵測

PDetect（Moussiades & Vakali, 2005）亦是針對程式作業的內容進行抄襲偵測的系統，它先取出程式原始檔中的關鍵字，接著將這些關鍵字進行相似度計算以取得相似度集合，這些相似度集合以分群演算法配合切斷策略值（Cut-off criterion value），將相似權重邊的連線頂點分為同一群以取得每個抄襲群組；另一個研究為 PDE4Java（Jadalla & Elnagar, 2008），它先利用 Jlex 與 CUP 建立 LALR 的剖析器，將程式原始碼進行符號化並轉換為 N-Gram 表示法，接著利用反轉索引建構關鍵字，再進行相似度計算以及分群。

Ji, et al.（2007）將程式抄襲源頭的追蹤加入演化，利用生物學的概念，取得程式的 DNA 序列，其意謂每個程式將具有屬於自己的關鍵字。為了判斷程式的重要性分數，他們利用調適性分數矩陣（adaptive score matrix）比對兩程式中，每個 DNA 片斷的符合情況。評分方式則是先算出所有作業中字元的佔有比率，若其佔有的比率較低，表示其具變造的可疑，因而重要程度的分數也會相對上升，經實驗結果顯示，當來源為同一作業之群組，重要程度分數較高的作業為資料集源頭的可能性越大。

然而，此方法無法完全滿足源頭偵測的需求。假設目前有兩份相同功能的程式原始碼，當中一位學生以另一份作業為副本，僅修改少部份內容，將因為多出更多判斷邏輯，使得程式的效能較源頭程式碼低落，此時會造成分數調整錯誤的問題。

因此，本研究強化現有技術，透過重要片段的參考性、重要片段的傳遞性、重要片段在群內相似性與群外差異性計算源頭的可能性。



圖三 群組化技術的改善方式

資料來源：本研究整理

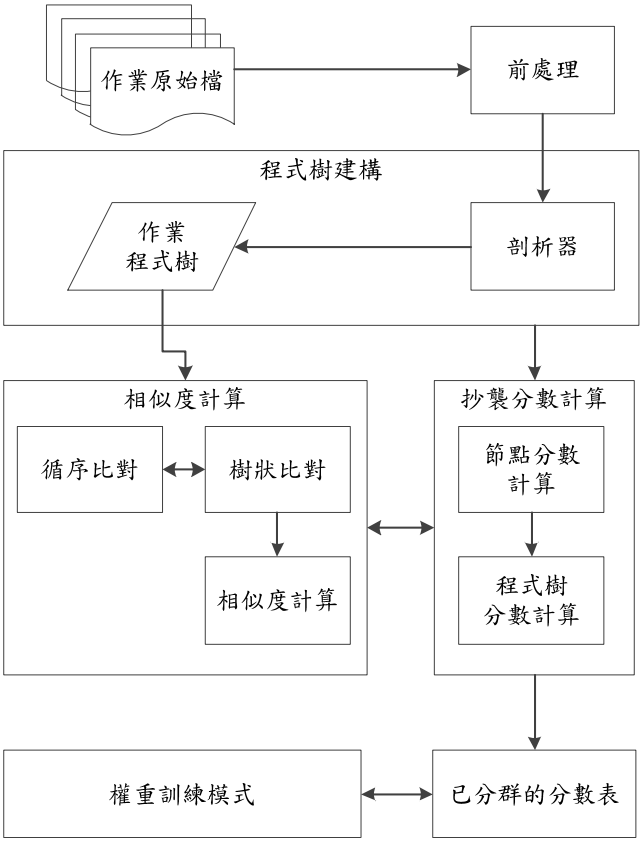
如圖三，本研究整合現有技術，以樹狀比對與符號比對為基礎，調整為以”重要片段”的概念，再以持續訓練以改善分群技術，最終達到源頭判斷與作業分群。

參·研究方法

第一節將介紹研究架構，第二節說程式樹的建構以及樹節點的設計，第三節說明程式相似度的計算，第四節談到本研究最主要的貢獻-源頭抄襲分數的計算，第五節說明抄襲分數進行權重訓練的方法。

一、研究架構

本研究主要將程式作業轉換為混合式樹狀結構，利用樹狀結構比對與樹節點內循序的比對方式計算相似度，接著將超過相似度門檻值的抄襲作業連結，形成作業抄襲群組。接著透過抄襲分數計算模組，計算每個抄襲群組內的作業，判斷它們成為源頭的可能性，最後將抄襲分數進行權重訓練，提升源頭偵測率，圖四為本研究的系統流程圖。



圖四 系統流程圖

資料來源：本研究整理

二、前處理與程式樹建構

前處理與程式樹建構，以混合式樹狀結構建構前，需先以詞彙分析作為前處理。ANTLR（ANother Tool for Language Recognition ）是開放原始碼的知名剖析器產生系統，它能進行詞彙分析以及語法剖析，本研究以 EBNF（Extended Backus-Naur Form）語法撰寫詞彙分析與語法剖析的邏輯，再由 ANTLR 轉換為目標程式語言的剖析器。

EBNF 是 Backus-Naur Form（BNF）的延伸，已被國際標準組織（International Organization for Standard）採用為國際標準。其運算符號如表四所示。

表四 符號示意表

目的	符號
定義	:
連接	,
結束	;
任一個	|
可選擇的	[…]
循環	{…}
群組	（…）
字串	"…" 或 '…'
註解	（*…*）

資料來源：本研究整理

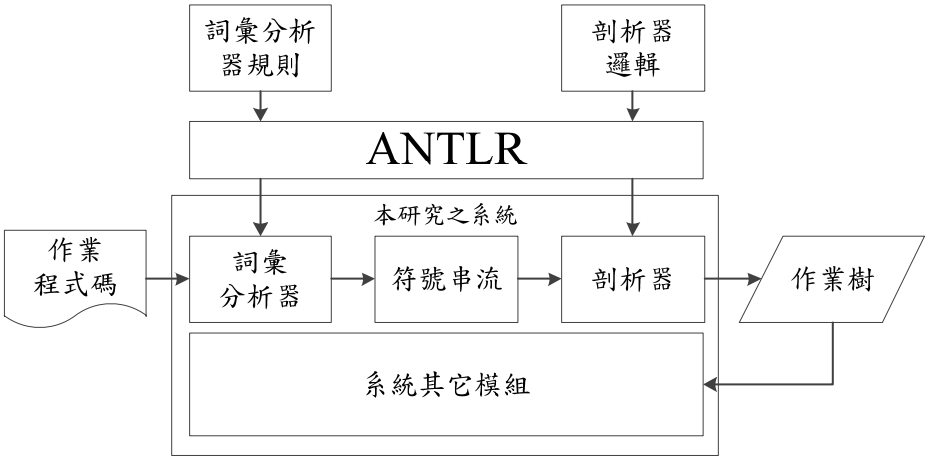
假設目標定義整數相加，可以表示為：

```
expr ：INT PLUS INT;
PLUS ：'+';
INT ：（'0'…'9'）+;
```

三行文字的定義是由下往上宣告，第三行宣告 INT 至少具有一個以上 0 至 9 之間的任一數字；第二行宣告 PLUS 為符號”+”，凡事遇上+的字元，就表示 PLUS，也因此第三行宣告 expr 表示整數與整數的相加。

如圖五，使用 ANTLR 以 EBNF 撰寫的規則進行分析，特色是能分別定義詞彙分析、語法剖析以及抽象語法樹的邏輯內容，經由 ANTLR 編譯後產出的內容，是程式語言的原始程式碼，它具有特定的邏輯剖析規則，並且因為它支援多種語言（JAVA, C#, C, Python, Objective-C 等），因此被廣泛應用於多個知名的系統內，如 Google App Engine, Adobe Flex Builder, Mac OS ProjectBuilder IDE 等。這類經由 ANTLR 輸出的內容，將是程式語言的函數或方法，因此它能處理讀入的串流，將它們依照 EBNF 撰寫的邏輯進行處理。

先建立詞彙分析器規則與剖析器邏輯，詞彙分析器是以 Open JDK 1.6 為基礎，將所有 JDK 定義的符號規則納入剖析器的規則，而剖析器邏輯是將這些定義的符號依其分類建立為樹狀結構。



圖五 作業程式碼解析技術區塊圖

資料來源：本研究整理

三、相似度計算與抄襲分數計算

建立完畢樹狀結構後，即可利用樹狀比對技術進行相似度計算及抄襲分數計算。目前大部份文字比對的文獻都以最長共同子字串偵測程式原始檔的抄襲，其中又以後綴樹運用最多，但無法避免多餘敘述的插入問題。由於本研究使用結構與敘述分離的方式偵測，且循序敘述因為有前後相依性，所以循序比對只需考慮多餘敘述插入的干擾，以最長共同子序列（LCS）演算法進行敘述的比對。

結構比對與符號比對的程序完成後，能取得相似度的節點集合與相似的循序集合，內容相似度計算公式 3.1 所示。

$$ContentSim(n_i, n_j) = \frac{2\times[LongCommonSequence(n_i n_j)]}{|n_i|+|n_j|} \qquad (3.1)$$

ContentSim() 函數輸出 0 與 1 之間的數值，LongCommonSequence() 表示兩節點 ni 與其對應之相似點 nj 的最長共同子序列符號長度，將其長度乘二後，再除以兩節點內的符號數量以讓數值正規化，因此取得的值將落於 0 與 1 之間。

結構相似度計算方式如公式 3.2，TreeSim() 函數輸出作業樹 i 與作業樹 j 的相似度，其相似度將介於 0 與 1 之間，其中 P 為作業樹 i 與作業樹 j 的對應節點集合，ni 節點必定對應至 nj 節點，分子部份先將 ContentSim() 的所有節點組合的相似值加總，再以正規化將加總的數值乘二，除去 i 與 j 的所有節點數。

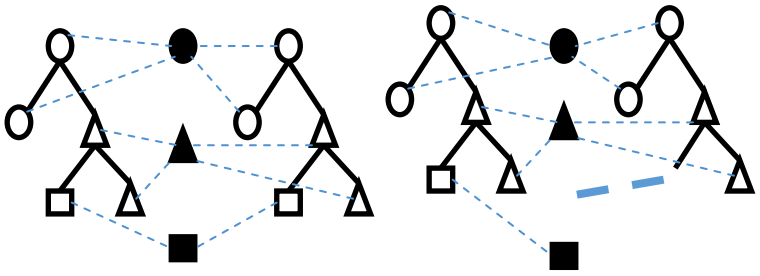
$$TreeSim(t,f) = \frac{2\times\left[\sum_{n_i,n_j \in p} contentSim(n_i,n_j)\right]}{|t|+|f|} \quad (3.2)$$

當計算出內容相似度 ContentSim() 與結構相似度 TreeSim()後，即可進一步計算出抄襲分數。抄襲分數是針對作業抄襲的特性設計的方法，當中包含三個重要概念，分別為重要片段的參考性、重要片段的傳遞性以及重要片段位於群內的相似性與群外的差異性等，以下分別針對三種特性進行說明。

(一) 重要片段的參考性

學生抄襲程式的主因之一在於他們的程度較差，無法靠本身的能力完成全部的內容（Joy & Luck, 1999）。抄襲者的動機可能是因為無法完成題目所交待的要求，透過複製他人的作業內容、甚至修改以完成作業。此時可以推測抄襲者因為本身能力不足，只對尚有能力修改的片段進行更動，本研究稱這樣的行為是有抄襲但具有部份修改。

在有抄襲但具有部份修改的情況中，抄襲者多半不會修改重要（具有難度的）片段，因此樹與樹之間從剛複製後到修改完成時具有兩個時間點，分別從兩棵節點完全相同的樹，演變為部份節點相同，如圖六所示。



圖六 重要片段的參考性示意

資料來源：本研究整理

圖六左邊是修改前的作業樹，顯示兩棵樹的每一個節點完全相同；右圖新增的節點是經過修改後的內容，為了計算每一個節點在群組中的重要程度，本研究必須計算它被每個程式樹參考（使用）的次數，如公式 3.3 所示。

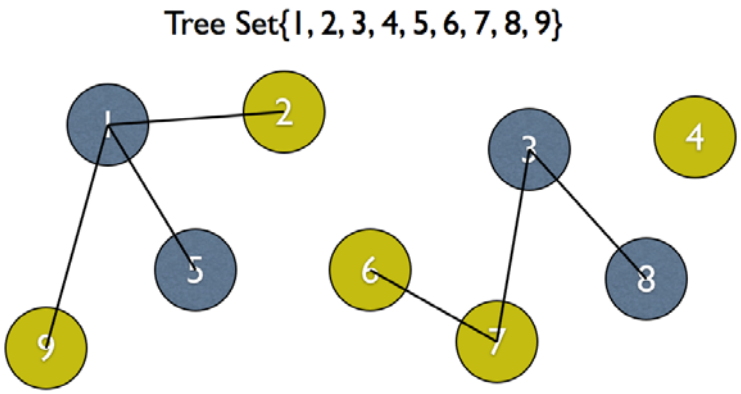
$$d(n) = \text{reference count of } T \quad (3.3)$$

其中 T={t1, t2, ..., ti}表示程式樹 ti 的集合，d（n）表示節點 n 被所有程式樹參考的次數。抄襲的學生對於抄襲的隱藏會經由修改部份字眼得來，這些字眼是他們有能力修改的部份，本研究視它為較簡單的部份。若結構複雜的內容或是相依性高而無從改起的循序敘述，那必然只能原封不動的使用，此時能突顯重要片斷會被更多抄襲作業參考的情況。

現在再考慮更多可能發生的情況，以 JAVA 程式語言為例，main 程式語言是程序的進入點，但對於模組化的程式而言，main 並不是重要的片段，因為它僅負責呼叫函式，由函式執行需要的演算法，若同一個抄襲群組的每個程式 main 都相同的情況下，被參考最多的節點就不一定是最重要的關鍵因子。

(二) 重要片段的傳遞性

能想像重要的片段困難度較高，最容易被大家抄襲，因此大部份抄襲的學生無法修改這此內容，所以在作業抄襲的群組中，包含這些重要片段的作業樹必然是相鄰的；若非如此，則視為簡單片段，因為同一個群組中若節點不相連，則表示不同的學生之間，都有能力撰寫出這樣的結構或循序敘述，因此無法成為表示源頭出處的關鍵因子。



圖七 重要片段傳遞性示意圖

資料來源：本研究整理

如圖七，假設程式樹 2、4、6、7、9 都具有同一個節點 n，圖中明顯分出三群{1, 2, 5, 9}、{3, 6, 7, 8}、{4}。本研究定義的三種狀況將依序說明：

i. 就第一群而言，因為程式樹 2 與 9 之間相鄰樹沒有參考節點 n，因此無論作業抄襲的傳遞可能方向為 2➔1➔9、9➔1➔2、2➔1➔9 或 2➔1➔9，無論上述任一種方向都無法改變節點 n 被中斷傳遞，或中途創造的可能性，因此出現在抄襲群組中的節點 n，只要具有一個以上的參考並且沒有被傳遞，就必須進行懲罰。

ii. 就第二群而言，6 與 7 之間可能的傳遞方向只有 6➔7、7➔6，因此它們之間無論何種情況，節點 n 都會被相鄰的下一個作業樹參考，所以能說它有較高的可能性為重要片段，這個情況節點 n 不需被懲罰。

iii. 最後一個情況是圖形的頂點沒有任何邊，本研究視它為正常而非抄襲的作業，因此不做任何懲罰。

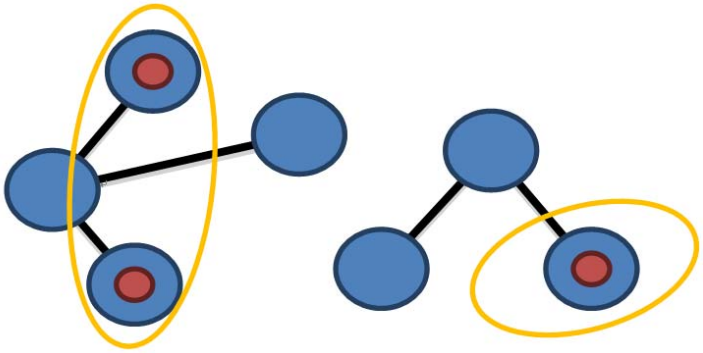
在取得懲罰集合 reduce 後，接下來以此集合計算最後的節點分數，如公式 3.4。

$$score(n) = \begin{cases} d(n) - \frac{|R|}{2} & ,if\ R \neq \emptyset \\ d(n) & ,otherwise \end{cases} \quad (3.4)$$

其中 R 表示懲罰集合，若 R 非空集合，則將節點 n 的分支度減去 R 總數的一半，反之則直接取用節點 n 的分數。

(三) 重要片段的群內相似性與群外差異性

以一般分群的原則而言，群組中的個體必須愈相近，群組間則是差異性愈遠，這個特色能延伸出對於抄襲群組的另一項分數懲罰方式。



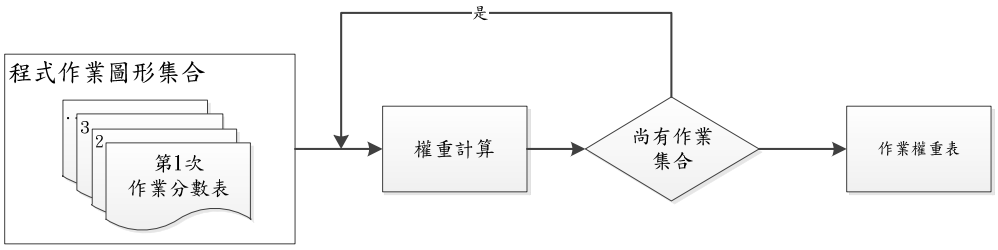
圖八 重要片段在群組間的示意圖

資料來源：本研究整理

如圖八，圈內節點表示兩群程式樹一共出現三次相同的節點，依照群內相似性與群外差異性而言，群組與群組間具有相同的節點必然無法視為群組內的重要片段，這個情況必須加入懲罰值。

(四) 權重訓練模式

若將掃描與分析一份作業示為單次的行為，則作業間必然具有循序漸進的關係。完整的權重訓練流程表示如圖九，首先以每次的作業分數作為訓練資料，再將訓練的權重回饋給原來的單次資料，因此抄襲者與被抄襲者之間的關係必然更加緊密。權重的訓練是將每次掃描的作業分數當作訓練資料以取得權重，若尚有其它未訓練的作業集合則持續進行訓練的程序。



圖九 權重訓練模式流程圖

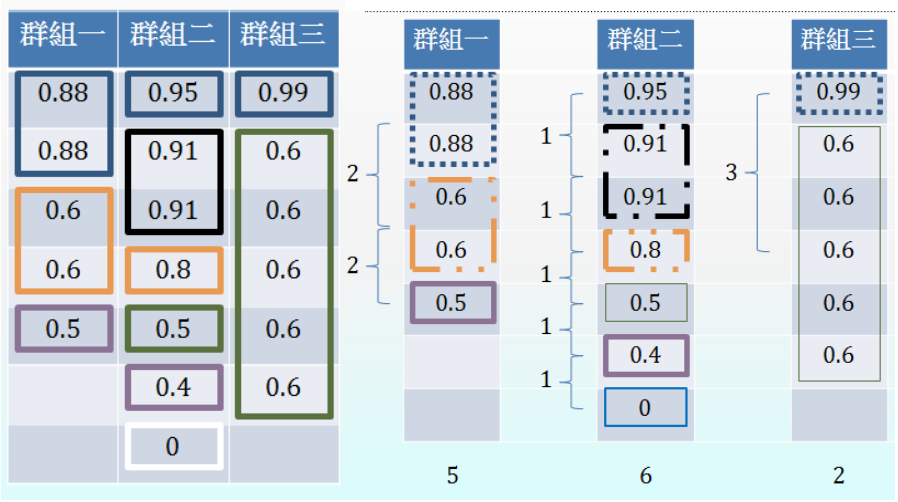
資料來源：本研究整理

除了僅有一次作業集合的情況無法回饋權重外，其它取得的權重值能再回饋給當次取得的分數列表。因此必須具有下列兩個限制條件為基礎：

1. 抄襲源頭或抄襲者可能會轉移，因此需要將第五節產生的群內相對權重值轉換為群間的全域權重值。

2. 若群內所有的分數相等，則表示本次的作業掃描僅能偵測它們具有抄襲行為，無法識別源頭與抄襲者。

權重的計算能讓區域相對值轉換為全域相對值，讓所有群組在分散間隔時間有較正確的對應方式，因此計算移動間隔能將較少級距的群組合併至較多級距的群組中。



圖十 移動間距示意圖

資料來源：本研究整理

如圖十，假設有三個群組，分別擁有五份、七份與六份作業，若將同分數視為相同等級，則它們將分別具有三種、六種與兩種級距。試著將這三個群

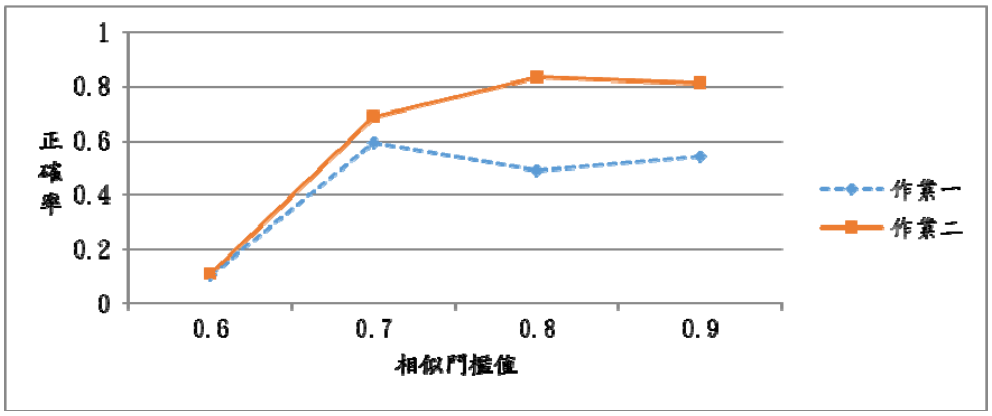
組的不同級距進行對應，讓它們能處於較為公平的分級過程，移動間距能計算出群內級距與群間最大級距的差異。

肆·實驗與分析

實驗樣本取自輔仁大學資訊管理學系，必修課程 Java 的修課大學生，樣本個數為 89，刪除錯誤樣本之後為 85，採集時間從 100 年 3 月 1 日至 5 月 31 日止，共 3 個月，每次實驗資料的收集時間為 1 小時，實際資料的收集時間約為 2-3 天。

為了確保實驗樣本的自由度，因此參與實驗資料收集的學生可任選一份作業進行抄襲或是自行撰寫，只需登記是否有抄襲或抄襲源頭即可。實驗將限制所有學生遵守抄襲源頭的原則，若單一樣本擔任五次作業中的任一源頭，則不可在其它非源頭的作業中具有任何的抄襲行為。此實驗以兩次實際資料集進行測試，並對班上的學生進行訪談，確定學生的抄襲由來，最後以此資料集評估本方法在實務上的適用性。

實驗將分別以 0.6、0.7、0.8 與 0.9 四種相似門檻值進行，詳細資料數據與作業分群資料請見附件二。



圖十一 門檻值對於分群的正確率趨勢

資料來源：本研究整理

兩次作業在四種門檻值的分群正確率如圖十一，其顯示門檻值對於分群正確率的變化情形，從趨勢圖能發現圖形的趨勢是先上升再下降的情況，但作業 2 在門檻值 0.7 時會呈現最高的數值，0.8 與 0.9 再呈現一次起伏，深入探討後發現此作業的源頭數量高達 9，換言之抄襲的群組分為 9 群之多。

以一般常理而言，門檻拉高後會讓群組減少造成群與群合併的情況，但0.9 的門檻值讓誤判為抄襲的人數提升之比率高於群組結合率，以致於正確率出現第二次成長。目前的兩份實際資料集都具有上課的參考範例以及書中的範例，所以參考範例的學生不認定為抄襲，因此正確率會因為範例群組的誤判率而較低。

實際資料集在抄襲的作業中有較多完全複製的情況，完全複製的情況下即使是以人為進行判斷也必須依賴額外的參考資料，例如作業成績、考試成績等，源頭的正確率必須扣除群組中發生全部同分或只有兩份作業的情況，才會是此研究判斷的實際正確率。

表五是觀察源頭後所得到的整理與分析。作業一之 49 與 78 為爭議源頭，說明如表五之情況分析一欄。經源頭修正後，實驗的結果顯示作業 1 為 7 群，與實際狀況相符。而作業二之 78 與 36 為爭議源頭，雖經源頭仍有三群誤差，包含參考範例群 2 群與教學群 1 群。

表五 實際作業之分析

<table>
<tr><td colspan="4">作業一</td></tr>
<tr><td>源頭</td><td>排名</td><td>門檻值</td><td>情況分析</td></tr>
<tr><td rowspan="4">49</td><td>8/54</td><td>0.6</td><td rowspan="4">從 0.7 至 0.9 的排名未改變，表示群內的結構較為獨立。</td></tr>
<tr><td>1/5</td><td>0.7</td></tr>
<tr><td>1/5</td><td>0.8</td></tr>
<tr><td>1/5</td><td>0.9</td></tr>
<tr><td rowspan="4">78</td><td>3/54</td><td>0.6</td><td rowspan="4">群中的第一高分共兩位同學，其中一位抄襲源頭將自己的版本傳遞給另一位，另一位完全複製而沒有做任何更改，讓兩者的節點多擁有一個參考數。</td></tr>
<tr><td>2/9</td><td>0.7</td></tr>
<tr><td>2/8</td><td>0.8</td></tr>
<tr><td>2/8</td><td>0.9</td></tr>
<tr><td colspan="4">備註：最多群數為門檻值 0.7 的 7 群：群中僅有 2 份作業的為 4 群，參考範例群為 1 群，共 7 群。</td></tr>
<tr><td colspan="4">作業二</td></tr>
<tr><td>源頭代號</td><td>排名</td><td>門檻值</td><td>情況分析</td></tr>
<tr><td rowspan="4">78</td><td>24/43</td><td>0.6</td><td rowspan="4">群組間又分為兩小群，一群為學生 36 的抄襲，並交給其它兩位學生；另一群則具有較大幅度的修改，其中幾位同學不僅修改變數，更在一百行左右的原始碼中增加三至五行的敘述。0.8 與 0.9 的門檻值中，78 不在抄襲群組內。</td></tr>
<tr><td>8/8</td><td>0.7</td></tr>
<tr><td>無</td><td>0.8</td></tr>
<tr><td>無</td><td>0.9</td></tr>
<tr><td rowspan="4">36</td><td>8/43</td><td>0.6</td><td rowspan="4">由於 36 抄襲 78 並有修改的情況，且其中兩位同學只進行微幅的修改，因此讓 36 的分數加總高於源頭 78 的分數。</td></tr>
<tr><td>1/8</td><td>0.7</td></tr>
<tr><td>1/4</td><td>0.8</td></tr>
<tr><td>1/3</td><td>0.9</td></tr>
<tr><td colspan="4">備註：最多群數為門檻值 0.7 的 9 群：群中僅 2 份作業者有 4 群、群中僅有一種分數者為 1 群、參考範例群（誤判群）有 2 群、教學群（誤判群）有 1 群，共 9 群。</td></tr>
</table>

資料來源：本研究整理

依照實驗結果，計算正確分群成功率 13 (正確分群) / 7(作業一實際群數) + 9(作業二實際群數) 為 81.25%。但相較於現有抄襲比對系統僅有分數未進一步分群，本研究提供了額外的抄襲群組關係資訊，以供授課老師作為實際抄襲與否的評估。

伍·結論與建議

本研究以現有之程式抄襲比對為基礎，結合分群方法以作為尋找抄襲源頭與抄襲群組的新方法。有別於以往僅以二作業的相似性作為抄襲判斷，且作業份數多時也必須逐一的兩兩比對，不但耗時且因無從辨識群組關係而使得實際價值不高。

在實際資料的訪談中發現，作業範例或是教師提供的授課資料，會影響本研究方法判斷抄襲的結果，因此分析目標必須移除範例作業，包含範例作業以及教學用作業，即可確保分群的正確性。

綜合而言，本研究透過源頭抄襲分數，只需比對分數較高與分數較低者的內容，即可得知它們的相似內容是否如預期，因此能在較短的時間中評判群組的分組是否正確。另外，分群的結果，也有利於授課老師藉由群組分析結果配合實際同儕關係的比較，進一步判斷是否有抄襲的可能。因此，本研究最大特色為：藉由群組資訊的揭露，使抄襲分析工程得以從單純的技術面向，進化為更多元的行為科學面向。

以系統完成分群後，在實際應用上仍需要與實際同儕群組關係的比較，才能有比較適合的行動依據。然而，實際同儕群組關係為授課老師依照課程分組狀況、作業指派狀況、或是學生之間的平時互動關係自行評斷，因此不屬於本研究範圍。唯在後續研究上，可再增加此一變項，可更完整呈現群組競合關係，以提供老師在對學生的品德輔導參考。

參考文獻

張火燦、劉淑寧，從社會網絡理論探討員工知識分享，*人力資源管理學報*，第 2 卷第 2 期，2002，頁 101-113。

游景翔，混合式電腦程式抄襲偵測，國立台灣科技大學資訊工程系碩士論文，2007。

黃政傑、張嘉育(2010)，讓學生成功學習：適性課程與教學之理念與策略，*課程與教學季刊*，第 3 卷第 13 期，頁 1-22。

Baker, B. S., Parameterized diff, In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99), USA, January 1999, pp. 854-855.

Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M. & Bier, L., Clone Detection Using Abstract Syntax Trees, 14th IEEE International Conference on Software Maintenance (ICSM'98), March 1998, pp. 368-377.

Belkhouche, B., Nix, A. & Hassell, J., Plagiarism detection in software designs, Proceedings of the 42nd annual Southeast regional conference, 2004, pp. 207-211.

Chanchal, K. R., James. R. C. & Rainer, K., Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, *Science of Computer Programming*, 74(7), 2009, pp. 470-495.

Ducasse. S., Nierstrasz, O. & Rieger, M., On the Effectiveness of CloneDetection by String Matching, *International Journal on Software Maintenance and Evolution: Research and Practice*, 18(1), 2006, pp. 37-58.

Greenan, K., Method-Level Code Clone Detection on Transformed Abstract Syntax Trees using Sequence Matching Algorithms, Student Report, University of California - Santa Cruz, Winter 2005.

Hirschberg, D. S., A linear space algorithm for computing maximal common subsequences, *Communications of the ACM*, 18(6), 1975, pp. 341-343.

Howard, J. J., Identifying Redundancy in Source Code Using Fingerprints. In Proceeding of the 1993 Conference of the Centre for Advanced Studies Conference (CASCON'93), October 1993, pp. 171-183.

Jadalla, A. & Elnagar, A., Pde4java: plagiarism detection engine for java source code: a clustering approach, *International Journal of Business Intelligence and Data Mining*, 3(2), 2008, pp. 121-135.

Ji, J. H., Woo, G., Park, S. H. & Cho, H. G., Evolution analysis of homogenous source code and its application to plagiarism detection, Frontiers in the Convergence of Bioscience and Information

Technologies, October 2007, pp. 813-818.

Joy M., & Luck. M., Plagiarism in programming assignments, *IEEE Transactions on Education*, 42(2), May 1999, pp.129-133.

Kamiya, T., Kusumoto, S., & Inoue, K., CCfinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Transactions on Software*, 28(7), 2002, pp. 654-670.

Lweicki, R.J. & B.B.Bunker: "Developing andMaintaining Trustin Work Relationships," in Book, R.M. Kramer and T.R. Tyler(Eds.) Trust in Organization: Frontiers of Theory and Research, 1996, pp.114-165. Sage Publications, Inc.

Michael, J. W., YAP3: improved detection of similarities in computer program and other texts, Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, 1996, pp. 130-134.

Moussiades, L. & Vakali, A., PDetect: a clustering approach for detecting plagiarism in source code datasets, *The Computer Journal*, 48(6), 2005, pp. 651-661.

Mozgovoy, M., Desktop Tools for Offline Plagiarism Detection in Computer Programs, Informatics in Education, 5, 2006, pp. 97-112.

Mozgovoy, M., Karakovskiy, S. & Klyuev, V., Fast and reliable plagiarism detection system, Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007, pp. 11-14.

Parker, A. & Hamblen, J. O., Computer algorithms for plagiarism detection. *IEEE Transactions on Education*, 32(2), 1989, pp. 94-99.

Prechelt, G. & Malpohl, M., Finding Plagiarisms among a Set of Programs with JPlag, *Journal of Universal Computer Science*, 8(11), November. 2002, pp. 1016-1038.

Rainer, K., Raimar, F. & Pierre, F., Clone Detection Using Abstract Syntax Suffix Trees, In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06) , October 2006, pp. 253-262.

Roy, C. K. & Cordy, J. R., A survey on software clone detection research, Technical Report 541, Queen's University at Kingston, 2007.

Schleimer, S., Wikerson, D. S. & Aiken, A., Winnowing: local algorithms for document fingerprinting. Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 2003, pp. 76-85.

Tairas, R. & Gray, J., Phoenix-Based Clone Detection Using Suffix Trees. In Proceedings of the 44th annual Southeast regional conference (ACM-SE'06), March 2006, pp. 679-684.

Ukkonen, E., On-line construction of suffix trees, Algorithmica, 14(3), 1995, pp. 249-260.

Whale, G., Identification of program similarity in large populations, *The Computer Journal*, 33(2),

1990, pp. 140-146.

Wise, M. J., String similarity via greedy string tiling and running Karp-Rabin matching. Retrieved October 20, 2010, from the World Wide Web: http://www.pam1.bcs.uwa.edu.au/~michaelw/ftp/doc/RKR_GST.ps.

Yang, W., Identifying syntactic differences between two programs, *Software: Practice and Experience*, 21(7), July 1991, pp. 739-755.

The Research of Source Code Detecting for Plagiarism Program

MING-JYH TSAI, SZU-HAO CHEN, HSUAN-YU TSENG, JYUN-JY HU *

ABSTRACT

In the past, studies focused on the plagiarism of programs. Only a few studies looked for plagiarism sources and plagiarism groups, but these methods were not designed for the field of student plagiarism.

This study is based on the papers of plagiarism and copy detection, combining similar assignments into groups, and using the concept of "important fragments" of programs; using the reference of important fragments, the transitivity of important fragments, and the internal similarity and inter-group difference of important fragments located in the group, to calculate the source possibility in the plagiarism group; finally, the weight training mode is used to train the weight of the plagiarized group, and the possibility that the true source is detected is improved.

The experimental results show that: (1) plagiarism scores can be sampled from one to five groups, all with good source detection rate. (2) The use of weight training mode can effectively improve the weight score of the real source and reduce the false positive rate of non-source. (3) The three-stage score calculation of important segments can effectively form the difference in scores within the group, making the real source more easily detected.

After the plagiarism group and the real source are analyzed, the instructor can further evaluate the ambiguity of plagiarism or plagiarism between the students by plagiarizing the group to match the competing group relationship between the students.

Keywords: plagiarism group, group and source analysis

* Ming-Jyh Tsai, Associate Professor, Department of Information Management, Fu Jen Catholic University. Szu-Hao Chen, Software Engineer, Potix Corporation. Hsuan-Yu Tseng, Doctoral Student, Graduate Institute of Business Administration, Fu Jen Catholic University. Jyun-Jy Hu, Associate Professor, Department of Information Management, Fu Jen Catholic University.