繁體中文文章之剽竊偵測工具實作

費彥霖
國立政治大學資訊管理學系

唐日新
國立台北商業技術學院資訊管理學系

廖淯任
國立成功大學工程科學系

摘要

全球資訊網(World Wide Web)的普及使得各種形式的資料在網路上廣為流傳，上網搜尋各種資訊已相當便利，但便利的同時也為人類帶來新的難題與挑戰；事實上，網路資訊的剽竊(Plagiarism)已是非常嚴重的問題。針對如何預防或偵測剽竊的問題，已有許多學者提出相關文獻探討(Chen et al. 2004)，許多偵測程式剽竊的軟體如：MOSS、JPLAG、YAP、SID等也陸陸續續被開發出來，另外許多偵測剽竊的商業網站如TURNITON.COM等，也都營運了幾年，相關技術之研究與應用已相當成熟與普及，不過這些研究以及軟體工具多半適用於電腦程式(program)的相似度的比對，對於實際偵測文章剽竊的工具很少，尤其在中文文章比對上，目前並沒有相關的研究，因此本研究的目的在於提出一套以LZW演算法為基礎(Ziv & Tempel 1977)，改良式的中文文章比對演算法，而為探討該演算法之可行性，實作一套中文文章剽竊偵測的雛形系統以進行實證，實驗評估顯示，任意選擇兩篇中文文章為比對主體，即能篩選出兩篇文章內容中所有相同的中文字句，因此，本研究的實驗結果，展現作者提出的中文文章剽竊偵測演算法，在技術上及實務應上具備可行性。然而，研究所提之演算法仍有許多改善的空間，透過進一步發展後，以期將來能夠作為日後華文世界中，文章剽竊研究的基礎，並能成功地被應用於教育研究等智慧財產相關體系中。

關鍵字：剽竊偵測、中文文章剽竊、智慧財產。

A Prototype for Plagiarism Detection in Chinese Contexts

Yen-Lin Fei
Department of Management Information Systems, National ChengChi University

Jih-Hsin Tang
Department of Information Management, National Taipei College of Business

Yu-Ren Liao
Department of Engineering Science, National Cheng Kung University

Abstract

The popularity of World Wide Web makes all forms of data, information and knowledge accessible to the public; however plagiarism has also become a major concern. To overcome the problems, several scholars have long proposed program detection algorithms (Chen et al. 2004), and several famous tools have been developed and used widely such as MOSS, JPLAG, YAP and SID. However, these tools and algorithms are not applicable to the traditional Chinese contexts. To fill this gap, we propose a LZW-based algorithm and develop a prototype to examine the feasibility and usability. Initial results confirmed the prototype is applicable in the Chinese article plagiarism detection. Further discussion and limitations are also provided.

Key words: plagiarism detection, text plagiarism, property rights

壹、緒論

對現代人而言，網際網路已成為生活上不可或缺的工具，但資訊的容易取得，卻帶來了相當嚴重的剽竊問題。不論是報告或學術論文，只要剪下以及貼上（copy and paste）簡單的動作，再稍稍修改成符合主題需求的格式，就似乎輕而易舉地完成一份創意十足的個人作品。剽竊行為在學術界發生的比率相當高，根據McLafferty與Foust (2004) 的研究顯示：學生認為剽竊在校園內是普遍且可被接受的。學校教師常常會面臨的困境之一是：學生時常繳交的作業是由網路上下載的資訊加以編輯而來，並且許多學生認為如此的作業方式是理所當然的。重覆的內容造成教育工作者的負擔，例如學生的作業有抄襲的情形時，老師在面對大量的作業內容與整理比對抄襲處時，需要花費非常多的人力與時間，同時也造成評分的困難與公平性的質疑。國內雖然沒有正式的統計資料，不過根據作者的經驗，剽竊行為在台灣大專校園中也相當普遍，許多大專教師也苦無對策。因此，網際網路時代的剽竊行為，是相當重要卻又受到忽視的研究課題，相當值得關注與探討。

由上述討論可知，如何整合資訊技術輔助教育工作者或其他相關機關來評定與解決剽竊的問題，是一個值得關心的議題。本研究針對剽竊偵測的問題，首先進行相關技術的文獻探討，並從實務研究導向，根據過去字串比對以及檔案壓縮的研究成果，提出一套改良式的中文文章剽竊偵測比對演算法，同時實作出一套雛型系統進行實驗性評估，並制訂文件相似性指標，作為偵測中文文章剽竊的依據。

貳、文獻探討

過去文獻中已經有不少和文章剽竊相關的研究，不過這些研究並不直接研究文章的相似度(similarity)，而是研究電腦程式間的相似性以及文章壓縮(compression)上。這些研究主要分為兩大類，第一類是比對電腦程式之間的相似度為主，另一類則是從文件的壓縮比對著手。以下則分為四小節，分別介紹目前用來偵測剽竊的軟體、演算法以及資料壓縮的演算法和目前在中文編碼壓縮上的應用。

一、剽竊偵測軟體

關於程式的剽竊(program plagiarism)，一般的定義是：一份程式碼，是由另一份程式碼所衍生出來的，只是做一些無意義文字編排動作，並非真正的了解程式碼的含意(Parker & Hamblen 1989)。在教育界，針對學生所繳交的程式作業，檢查作業間是否有抄襲的情形，對學校老師或是助教而言，深具挑戰性。藉助有效的軟體來幫忙偵測程式作業的抄襲問題，有助教學品質的提升。許多偵測程式剽竊的軟體，其實已被開發出來了。綜觀這些系統，可大約略分為兩類：屬性計算系統(Attribute-counting systems)及結構

度量系統(Structure-metric systems)(Verco & Wise 1996)。

（一）屬性計算系統(Attribute-counting systems)

一個簡單的屬性計算系統，只粗略計算一個程式碼內含有幾種運算元、運算子，以及他們各自的總數是多少，而後對每個程式碼建立一個統計圖表用以比對彼此的異同。

（二）結構度量系統(Structure-metric systems)

這個方法則是比較程式碼之間結構的異同，比起屬性計算系統，結構度量系統對於程式的剽竊問題而言，是較為優良而有效實用的偵測技術，一些廣泛的系統如Plague、Moss、JPlag、SIM以及YAP等等，都是屬於結構度量系統。系統通常包含兩個步驟，首先以語意分析機將原始碼轉換為符號串列(token sequences)；接下來比較轉換後的符號串列(token sequences)。重點在於比較轉換後的符號串列，不適切的比較方法常使得偵測的成效不彰。

二、程式剽竊偵測演算法

本偵測程式碼剽竊的演算法有三個特性(Wise 1994)：

特性一：每個符號(token)最少要被計算一次。

特性二：轉換過的區段碼對於相似的結果值都需產生影響。

特性三：每當輸入的程式碼隨機插入(insert)、刪除(delete)一些識別字時，相似的結果值都會下降。

以下介紹兩種偵測程式抄襲的軟體之演算法則。

（一）屬性計算系統(Attribute-Counting systems)

最早的此類系統用來比較兩份程式碼的相似程度，計算程式的四個值(Ottenstein 1976)。

P_1 = 有幾種運算子

P_2 = 共有幾種運算元

N_1 = 所有運算子的數目總合

N_2 = 所有運算元的數目總合

$$V = (N_1 + N_2)\log_2(P_1 + P_2) \qquad (2\text{-}1)$$

$$E = (P_1 N_2 (N_1 + N_2)\log_2(P_1 + P_2)) \div (2P_2) \qquad (2\text{-}2)$$

（二）YAP系統

YAP系統為一種以百分比(0-100 percent-match)做為評估相似度標準的系統(Wise 1994)，公式如下：

$$Match = (same - diff)/minfile - (maxfile - minfile)/maxfile \qquad (2\text{-}3)$$

$$PercentMatch = Max(0, Match) \times 100 \qquad (2\text{-}4)$$

其中maxfile及minfile是檔案大小的長度，變數same是兩個檔案相同的識別子數目，而變數diff指的是在一個程式區塊中有幾個相異的識別字數目。目的是盡可能找出最長的

相符句子的集合，集合內的句子含蓋的識別子互相有所不同。

以上這兩種系統基本上都是以程式中的識別字(token)來做兩支程式碼比對的來源，現行程式碼識別字多以英文命名，如for、while、if…end…，所以在比對上可預設條件值為程式內訂的識別字，字串比對也是以英文字串為主。

這些程式剽竊的研究成果相當豐碩，早已應用在學生程式剽竊的偵測上，有興趣的讀者可以參考Lancaster與Culwin的文章（2004）。不過程式剽竊演算法主要的限制是：這些研究的基礎在於以羅馬拼音為基礎的研究假設上，但中文字並非拼音的文字，中文字的數量又相當龐大，無法事先知道比對的文章內容會出現哪些中文字(或關鍵字)，所以無法事先設定比對條件值，在文字的比對上也無法像英文字母一樣應用Huffman tree等演算法作字串比對(Huffman 1952)，所以無法將這些研究的成果應用在中文環境下。

由於字串比對的研究限制，於是我們找出另一種和字串比對相關的研究，也就是資料壓縮的演算法，我們的基本假設是：修改資料壓縮演算法或許可以用在中文資料的字串比對。以下則簡單介紹相當著名的資料壓縮演算法LZW Algorithms。

三、資料壓縮演算法

（一）Lempal-Ziv Compression LZ演算法

最1977年由兩位以色列電腦專家Abraham Lempel及Jacob Ziv所發表的壓縮演算法(Ziv & Lempel 1977)，它的原理是把原本資料內相同的字串以較小的代號取代，被取代的資料只需儲存一份，藉以減少資料長度，解壓縮時再將代號還原為原本取代前的資料即可。目前市面上有的壓縮檔案格式，例如LZEXE、LHA、GIF、PKZIP等都是以此方法為核心。以下即為LZW演算法介紹(Chang & Tsai, 1991；Ziv & Lempel 1977；張真誠 & 蔡文輝，1991)：

（1）使用資料源的輸入符號集合A中的所有符號字元來建立一本初始的字典，其中每一個符號被當成字典中的一個字。

（2）找出目前待編碼資料中和字典的字相同之最長子序列，將此匹配子序列編碼，送出一個欄位的編碼結果〔INDEX〕，其中INDEX指最長匹配子序列在字典中的索引位置；此匹配子序列與其後的一個符號連結成一個新的子序列，將此新序列加入字典中成為一個新字。

（3）重複步驟二直到待編碼資料都處理完畢才停止。

我們假設一待編碼序列aabcaab，初始字典為表1：

表1：LZW編碼字典初始表

字元	碼
A	1
B	2
C	3

編碼過程為如表2：

表2：編碼過程表

步驟	讀入字元	佇列內容	輸出字元	輸出編碼
1	A	A		
2	A	aa	a	1
3	B	ab	a	1
4	C	bc	b	2
5	A	ca	c	3
6	A	aa		
7	B	aab	aa	4
8		b	b	2

LZW編碼最後產生字典如表表3：

表3：LZW編碼字典結果表

字元	碼
a	1
b	2
c	3
aa	4
ab	5
bc	6
ca	7
aab	8

四、資料壓縮演算法在中文編碼壓縮上的應用

資料對於電腦而言只是一堆0與1構成的符號，為了讓0與1組成的位元串能表示出有意義的資料就必須事先加以編碼；在英文系統考慮26個英文字母以8個位元即可完成編碼，但中文字的字數若依康熙字典中就有四萬多字，於是為了在電腦上統一應用，我國資策會制定了一套名為BIG-5中文內碼，為了有效表示出中文字及符號，必須用到兩個位元組，第一個位元組稱高位元組，通常第一個位元設為1，而第二個位元組稱低位元組。由於中文編碼採用兩個位元組的編碼方式，稱之為雙位元組字集(DBCS，Double Byte Character SET)(張真誠 & 蔡文輝，1991)。

植基於LZW編碼法的純粹中文資料壓縮，採用一個整數來取代一組文字串，將資料檔案分成高位元組群(H)、低位元組群(L)、換行跳列群(S)，同時對H與L群以LZW演算法做壓縮，S群則直接輸出，再各自針對三份輸出做編碼的動作，即完成一份查詢字典，而後每當讀入新的檔案時，再與原本建立的字典做比對，即可比較出原檔案與新輸入檔案之相似度為何(張真誠 & 蔡文輝，1991)。

從過去文獻的成果看來，許多程式剽竊的演算法如MOSS、JPLAG、YAP、SID等，無法直接應用在中文的文章比對上，而資料壓縮的演算法似乎也無法直接應用在中文字串的比對，所以需要設計一個新的演算法，可以用以比對文章的相似度。本研究的主要目的就是自行設計或修改過去的演算法，以便套用在中文環境上。

參、研究方法

本研究目的在於提出一套以LZW壓縮演算法為基礎的改良式中文文章比對演算法，而後對所提出的演算法進行驗證，實作出一套中文文章剽竊偵測的雛形系統，以此中文文章剽竊偵測演算法為基礎，期望再結合技術開發出實務導向的中文剽竊偵測系統，進而做為提供教育工作者或相關機關與專業人士等可使用之剽竊偵測輔助工具。因此，研究步驟先依文獻探討結果整理出剽竊所衍生之現象與剽竊偵測目前的技術與發展，而後再以LZW壓縮技術與字串比對為基礎提出一套改良式中文文章剽竊偵測演算法，並實際開發一雛型系統，以進行演算法之實驗評估。

由於本研究須將文字檔案做壓縮處理，以供比對字串用途，我們利用Microsoft .NET平台所提供豐富的元件來開發與執行中文文章剽竊偵測程式。以下分別介紹本研究之研究流程、研究範圍與限制、以及傳統中文文章剽竊偵測演算法原理說明。

一、研究流程

由文獻探討的過程之中，瞭解剽竊在學術領域中的一些問題與爭議，於是建立了研究背景與動機；由於目前並無針對中文文章剽竊偵測的相關研究，於是就以此擬定研究目的，以期能夠設計一套演算法用以偵測中文的剽竊；接著藉由閱讀其他學者之文獻研究，構思中文資料比對演算法，然進行程式分析與實作，綜合以上步驟，形成本研究之研究流程，如圖1所示。

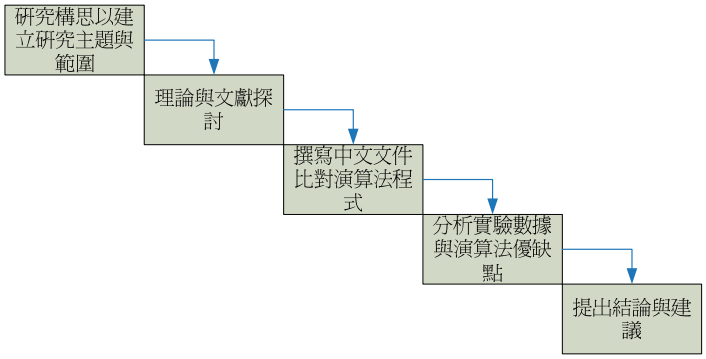

圖1：研究流程圖

二、研究範圍與限制

中文文章比對的研究雖然重要，不過目前並沒有直接針對中文文章剽竊的相關研究，本研究僅是探索性的研究，研究範圍聚焦於學生報告剽竊的偵測上，因此研究的對象便鎖定在學生族群，針對小篇幅的中文字報告來開發剽竊偵測軟體，藉著有效的字串比對演算法來找出學生所繳交報告內容的相似度為何，以期能降低學生不當抄襲、引用的風氣，進而提升教學品質。如果成效卓著，往後可能應用在中文著作權侵權的偵測上。

三、研究定位

由於本研究為針對繁體中文文章之剽竊偵測為主要應用方向，相較於既有針對英文字母為主要字根的程式碼剽竊偵測的相關應用，仍有根本上的不同。但整體而言，本研究所提出之剽竊偵測演算法仍可應用於程式碼偵測，具有較高涵蓋性；同時，由於中文字字根數量龐大，在偵測速度上與JPlag與Moss等偵測軟體相較下會犧牲一些執行效能；此外，相較於JPlag與Moss偵測軟體，本研究所開發之雛型系統為即時反應，使用者可立即得知比較文本內容的相同處為何，在教學應用面上，有助於提升教師評判剽竊速度。

表4：一些剽竊偵測軟體之屬性比較

	JPlag	Moss	本研究所提之偵測軟體
偵測主體	程式碼	程式碼	程式碼與中英文數字皆支援
演算法	Token pattern matching	未釋出	編碼比對
介面	GUI	Web interface	GUI
服務型態	Web service	Internet service	Standalone application
檔案來源	檔案或資料夾路徑	檔案或資料夾路徑	檔案(.txt檔)
速度	快	快	文章越長則需要時間越長
是否呈現比對相同處	是	是	是
即時回應	否	否	是
使用對象	老師	老師	老師
是否免費	是	是	是

四、中文文章剽竊偵測演算法原理

傳統的字串比對演算法中，大多針對英文字母做字串比對，考慮26個英文字母、大小寫、標點符號、控制符號等資料，只需用8位元即可完成編碼，英文字串無論如何排列組合都是以26個字母為字根；但是中文字數量相當龐大，光依照康熙字典就已有四萬多字，若是拿兩篇文章做直接比對，字根則為兩篇文章中文字的聯集，拿三篇文章做直接比對，字根則為三篇文章中文字的聯集，而每個中文字元在電腦系統中是以2bytes來表

示，當字根越來越龐大系統的負荷也越重。

為了達到中文文字的比對，作適度的壓縮是必要的工作。因為數量龐大的中文字，如果不作適當的壓縮與編碼，文字比對將成為系統無法負荷的工作。

（一）演算法虛擬碼

以LZW壓縮演算法做修改後，應用在中文文章比對的演算法，茲以虛擬碼說明。由於本研究之中文文章比對演算法共分為一個主程式，及四個副程式。其中我們主要使用三個陣列，先以對照表陣列(Comparison_table_array[][3])來還原結果值陣列(Result_array[])中的值，然後可以得到原始文章相同句子所對應的編碼數字。然後再將所得到的對應編碼數字依照字典庫陣列(Words_library_array[])將編碼數字還原成中文句子，最後得到兩篇文章之中相同的中文字或句子。

中文文章剽竊偵測演算法：

輸入：文章A與文章B；

輸出：結果值陣列(Result_array[])；

步驟1.先設定以下五個變數預設值：

```
Int Encode=0；
Int[][3] Comparison_table_array；
Char[] Words_library_array=Encode(A)；
Bool Stop=false；
ARTICLE= B；
```

主程式演算法虛擬碼：

```
Main function
While(!Stop)
{
   NUM_represented_article=Addcode(ARTICLE);
   For each num of NUM_represented_article
   {
        if(Check_middle_num(num))
                Result_array[].add(num);
   }
   Combine_code(ARTICLE);
   If(every num of ARTICLE==-1)
          !Stop;
   else
          switch ARTICLE;
 }
```

圖2：主程式(Main function)虛擬碼

主程式演算法虛擬碼：

```
Function Encode(an article)
{
    for each word in the article
    {
        if this word does not appear in Words_library_array[]
        {
            Words_library_array[].add(this word);
            Encode a number to represented this word;
            Encode_num++;
        }
     }
}
```

圖3：副程式(Encode function)虛擬碼

副程式虛擬碼(Addcode)：

```
1   Function Addcode(ARTICLE)
2   {
3       if(ARTICLE is represented by word)
4       {
5           for each word in ARTICLE
6           {
7               if this word appear in Words_library_array[]
8               {
9                   assign its Encode_num to represent this word;
10              }
11              else
12                  -1 represent this word;
13          }
14      }
15      else//ARTICLE has been convert to numbers
16      {
17          for each two linked_nums
18          {
19              if(first_num == Comparison_table_array[n][0] &&
20              second_num== Comparison_table_array[n][1])
21                  first_num= Comparison_table_array[n][2];
22              else
23                  first_num=-1;
24          }
25      }
26  }
```

圖4：副程式(Addcode function)虛擬碼

副程式虛擬碼(Bool Check_middle_num)：

```
1   Function Bool Check_middle_num(an number)
2   {
3       if(pre_num==-1 && next_num==-1)
4           return true;
5       else
6           return false;
7   }
```

圖5：副程式(Bool Check_middle_num function)虛擬碼

副程式虛擬碼(Combine_code)：

```
1   Function Combine_code(ARTICLE)
2   {
3       for each two linked_nums
4       {
5           if(first_num!=-1 && second_num!=-1)
6           {
7               if(first_num== Comparison_table_array[n][0] &&
8               second_num==Comparison_table_array[n][1])
9                   first_num== Comparison_table_array[n][2];
10              else
11              {
12                  Encode_num to represent first_num;
13                  Encode_num++;
14              }
15          }
16          else
17              first_num=-1;
18      }
19  }
```

圖6：副程式(Combine_code function)虛擬碼

（二）演算法流程說明

以下以一案例說明本研究提出之改良式演算法流程，假設輸入文章A、文章B。第一份文章A的內容為：資料壓縮與資料結構。

1.第一次執行

步驟一：將文章A中每個不同的字元給定一個編碼值，並記錄在字典庫陣列中。

將文章A每個中文字給定一個阿拉伯數字編碼。而後將文章A依據字典庫陣列(Words_library_array[])以數字呈現。

表5：字典庫陣列(Words_library_array[])

資	料	壓	縮	與	結	構
1	2	3	4	5	6	7

(Words_library_array[])以數字呈現。

資	料	壓	縮	與	資	料	結	構
1	2	3	4	5	1	2	6	7

我們假設文章B的內容為：資料結構與壓縮演算法。

步驟二：檢查文章是以文字還是以數字的形式代表。若為文字形式則替換成字典庫陣列中之編碼值；若已轉換為數字形式則與對照表陣列對照檢查文章連續兩兩編碼值。若文章以文字的形式時，就依字典庫陣列中的數字替換，有出現在字典庫陣列中的文字以對應的編碼數字替換，沒出現在字典庫陣列中的文字以-1替換。

原理說明：以-1替換的理由是因為既然字典庫陣列初始內容是以文章A所有出現過的字所編，文章B中若含沒有出現在字典庫陣列中的字，代表這些字元亦不可能在文章A中出現，所以也不可能會相同，於是便以-1替換。

已知字典庫陣列內字元為所有在文章A中出現字元所編，所以假設P代表字典庫陣列，可知

$$(P \cap B) \subset A \qquad (3\text{-}1)$$

若文章已經替換成數字的話，就檢查連續的兩兩編碼數字有沒有出現在對照表陣列內，有的話就將連續的兩兩編碼數字的第一個數字以對照表陣列內新編值替換；沒有的話就將連續的兩兩數字的第一個數字以-1替換。

在本例中，文章B初始狀態是文字形式，尚未被替換成數字，所以若文章B中的文字有出現在字典庫陣列內，就把文章B中的文字以字典庫陣列內的數字替換，若文章B中沒有出現在字典庫陣列內的文字，就以-1替換。

表5：字典庫陣列(Words_library_array[])

資	料	壓	縮	與	結	構
1	2	3	4	5	6	7

文章B內容照著字典庫陣列將中文替換成數字:

資	料	結	構	與	壓	縮	演	算	法
1	2	6	7	5	3	4	-1	-1	-1

步驟三：檢查非-1的編碼值有無被兩個-1夾在中間，或是被-1隔在文章開頭或結尾。

若轉換成數字後的文章有出現兩個-1中間夾著一個非-1數字的情形，或是被-1隔在字串開頭或結尾的數字，就把被-1夾著的數字及被-1隔在兩端的數字存入結果值陣列(Resule_array[])。

步驟四：轉換編碼文章主體(Switch Article)，由文章B編碼內容連續兩兩編碼，結果存入對照表陣列。

將文章B的數字兩兩編碼，若連續兩個數字中有一個以上的數字是-1，就把連續兩個數字中的第一個數字換成-1；若連續兩個數字都不是-1，且另一篇文章尚未將連續兩個數字編定新碼，就指定一個新碼放到連續兩個數字中的第一個數字，並存入對照表陣列內。

第一列存原來的文章B，第二列存替換為新值的文章B。

1	2	6	7	5	3	4	-1	-1	-1
8	9	10	11	12	13	-1	-1	-1	-1

產生一份對照表陣列表6 (Comparison_table_array[][3])，內容如下：

表6：對照表陣列(Comparison_table_array[][3])

First_num	Second_num	Comb_num
1	2	8
2	6	9
6	4	10
7	5	11
5	3	12
3	4	13

步驟五：檢查文章編碼值內容是否全為-1。

檢查被替換為新值的文章B是否每個值都是-1，是的話，程式結束；若不是每個值都是-1，則切換至文章A。重覆步驟二至步驟五至結束為止。

2. 第二次執行

以下為切換為文章A時繼續重覆步驟二至步驟五執行的動作：

步驟二：檢查文章是以文字還是以數字的形式代表。

因為文章A已為數字形式，所以就檢查連續的兩兩編碼有沒有出現在對照表陣列內，有的話就將連續的兩兩數字的第一個數字以對照表陣列內新編值替換；沒有的話就將連續的兩兩數字的第一個數字以-1替換。

本例中，第一列存原來的文章A，第二列存替換為新值的文章A。

1	2	3	4	5	1	2	6	7
8	-1	13	-1	-1	8	9	10	-1

步驟三：檢查非-1的編碼值有無被兩個-1夾在中間，或是被-1隔在文章開頭或結尾。

若轉換成數字後的文章有出現兩個-1中間夾著一個數字或是有數字被-1隔在兩端的情形，就把數字存入結果值陣列(Resule_array[])。

本例中，被兩個-1所夾的數字有13，被隔在兩端的數字有8，將13與8存入結果值陣列(Resule_array[])，產生結果值陣列：

表7：結果值陣列(Resule_array[])

8	13

步驟三：檢查非-1的編碼值有無被兩個-1夾在中間，或是被-1隔在文章開頭或結尾。

步驟四：轉換編碼文章主體(Switch Article)，由文章A編碼內容連續兩兩編碼，結果存入對照表陣列。

將文章A的數字兩兩編碼，若連續兩個數字中有一個以上的數字是-1，就把連續兩個數字中的第一個數字換成-1；若連續兩個數字都不是-1，且另一篇文章尚未將連續兩個數字編定新碼，就指定一個新碼放到連續兩個數字中的第一個數字，並存入對照表陣列內。

本例中，第一列存原來的文章A，第二列存替換為新值的文章A。

8	-1	12	-1	-1	8	9	10	-1
-1	-1	-1	-1	-1	-1	-1	14	15

將新兩兩編碼的值新增入對照表陣列(Comparison_table_array[][3])，內容如表8：

表8：對照表陣列(Comparison_table_array[][3])

First_num	Second_num	Comb_num
1	2	8
2	6	9
6	7	10
7	5	11
5	3	12
3	4	13
8	9	14
9	10	15

步驟五：檢查文章編碼值內容是否全為-1。

檢查被替換為新值的文章A是否每個值都是-1，是的話，程式結束；若不是每個值都是-1，則切換至文章B。重覆步驟二至步驟五至結束為止。

3.第三次執行

以下為切換為文章B時繼續重覆步驟二至步驟五執行的動作：

步驟二：檢查文章是以文字還是以數字的形式代表。

因為文章B已為數字形式，所以就檢查連續的兩兩編碼有沒有出現在對照表陣列內，有的話就將連續的兩兩數字的第一個數字以對照表陣列內新編值替換；沒有的話就將連續的兩兩數字的第一個數字以-1替換。

本例中，第一列存原來的文章B，第二列存替換為新值的文章B。

8	9	10	11	12	13	-1	-1	-1	-1
14	15	-1	-1	-1	-1	-1	-1	-1	-1

步驟三：檢查非-1的編碼值有無被兩個-1夾在中間，或是被-1隔在文章開頭或結尾。

若轉換成數字後的文章有出現兩個-1中間夾著一個數字的情形，或被-1隔在兩端的數字，就把中間被-1夾著的數字及被-1隔在兩端的數字存入結果值陣列(Resule_array[])。

本例中，沒有被兩個-1所夾的數字，也沒有被-1隔在兩端的數字，結果值陣列仍不變：

表7：結果值陣列(Resule_array[])

8	13

步驟四：轉換編碼文章主體(Switch Article)，由文章B編碼內容連續兩兩編碼，結果存入對照表陣列。

將文章B的數字兩兩編碼，若連續兩個數字中有一個以上的數字是-1，就把連續兩個數字中的第一個數字換成-1；若連續兩個數字都不是-1，且另一篇文章尚未將連續兩個數字編定新碼，就指定一個新碼放到連續兩個數字中的第一個數字，並存入對照表陣列內。圖示如下：

本例中，第一列存原來的文章B，第二列存替換為新值的文章B。

14	15	-1	-1	-1	-1	-1	-1	-1
16	-1	-1	-1	-1	-1	-1	-1	-1

將新兩兩編碼的值新增入對照表陣列(Comparison_table_array[][3])，內容如表9：

表9：對照表陣列(Comparison_table_array[][3])

First_num	Second_num	Comb_num
1	2	8
2	6	9
6	7	10
7	5	11
5	3	12
3	4	13
8	9	14
9	10	15
14	15	16

步驟五：檢查文章編碼值內容是否全為-1。

檢查被替換為新值的文章B是否每個值都是-1，是的話，程式結束；若不是每個值都是-1，則切換至文章A。重覆步驟二至步驟五至結束為止。

4.第四次執行

以下為切換為文章A時繼續重覆步驟二至步驟五執行的動作：

步驟二：檢查文章是以文字還是以數字的形式代表。

因為文章A已為數字形式，所以就檢查連續的兩兩編碼有沒有出現在對照表陣列內，有的話就將連續的兩兩數字的第一個數字以對照表陣列內新編值替換；沒有的話就將連續的兩兩數字的第一個數字以-1替換。

本例中，第一列存原來的文章A，第二列存替換為新值的文章A。

-1	-1	-1	-1	-1	-1	-1	14	15
-1	-1	-1	-1	-1	-1	-1	-1	16

若轉換成數字後的文章有出現兩個-1中間夾著一個數字的情形，或被-1隔在兩端的數字，就把中間被-1夾著的數字及被-1隔在兩端的數字存入結果值陣列(Resule_array[])。

本例中，被隔在兩端的數字有16，將16存入結果值陣列(Resule_array[])，產生結果值陣列如表10：

表10：結果值陣列(Resule_array[])

8	13	16

步驟四：轉換編碼文章主體(Switch Article)，由文章A編碼內容連續兩兩編碼，結果存入對照表陣列。

將文章A的數字兩兩編碼，若連續兩個數字中有一個以上的數字是-1，就把連續兩個數字中的第一個數字換成-1；若連續兩個數字都不是-1，且另一篇文章尚未將連續兩個數字編定新碼，就指定一個新碼放到連續兩個數字中的第一個數字，並存入對照表陣列內。

本例中，第一列存原來的文章A，第二列存替換為新值的文章A。

-1	-1	-1	-1	-1	-1	-1	-1	16
-1	-1	-1	-1	-1	-1	-1	-1	-1

步驟五：檢查文章編碼值內容是否全為-1。

檢查被替換為新值的文章A是否每個值都是-1，是的話，程式結束；若不是每個值都是-1，則切換至文章B。重覆步驟二至步驟五至結束為止。

在執行四次之後程式結束，我們可以看到結果值表如下：

表10：結果值陣列(Resule_array[])

8	13	16

我們得到結果值陣列的內容之後，首先將結果值陣列的內容與對照表陣列對照以還原出代表原始文章的數字：

表9：對照表陣列(Comparison_table_array[][3])

First_num	Second_num	Comb_num
1	2	8
2	6	9
6	7	10
7	5	11
5	3	12
3	4	13
8	9	14
9	10	15
14	15	16

本例中8可被還原成1和2；13被還原為3和4；16被還原為1、2、6、7。

8 還原->1、2
13還原->3、4
16還原->14、15還原->8、9、10還原->1、2、6、7

再將還原後的數字對照字典庫陣列以求出中文字句，找出兩篇文章相似的字句為何。

表5：字典庫陣列(Words_library_array[])

資	料	壓	縮	與	結	構
1	2	3	4	5	6	7

可以知道兩篇文章一樣的字句為8(還原出資料)、13(還原出壓縮)以及16(還原出資料結構)。

流程圖示如下:

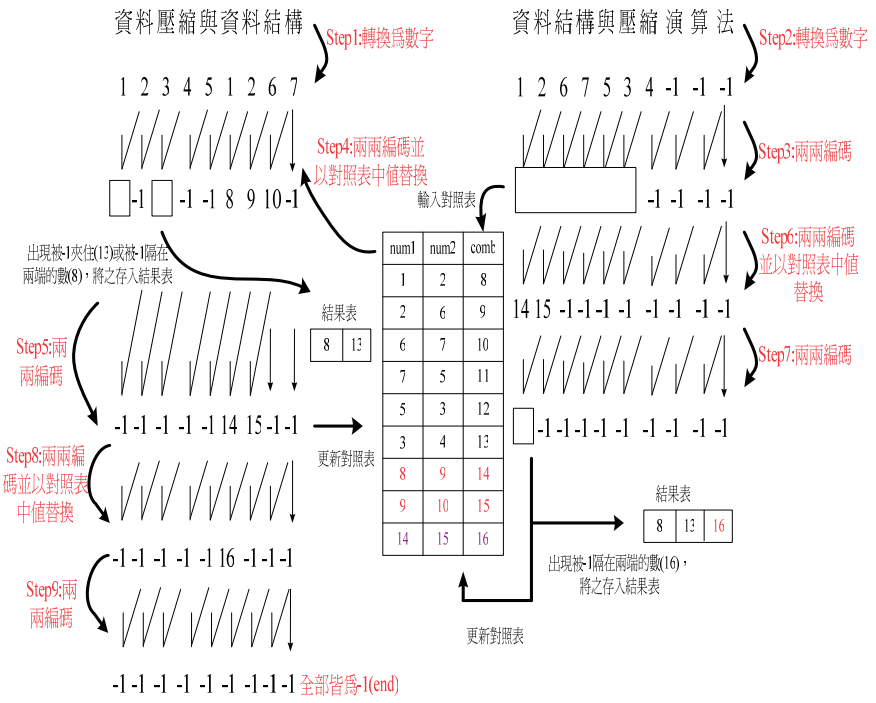

圖7：演算法案例流程圖

由圖7中我們可以看到，演算法流程主要分為兩個大步驟：一篇文章開始兩兩編碼後將編碼值存入對照表陣列；之後由另一篇文章兩兩編碼，並將有出現在對照表中num1與num2的值以comb值替換。

由文章B開始兩兩編碼並將編碼值存入對照表陣列內，再來是文章A兩兩編碼並以對照表陣列內的值替換。也就是圖中的Step3與Step4。

若文章A編碼後的值不全為-1，則由文章A開始兩兩編碼並將編碼值存入對照表陣列內，再來是文章B兩兩編碼並以對照表陣列內的值替換。也就是圖中的Step5與Step6。

若文章B編碼後的值不全為-1，便再由文章B開始做兩兩編碼並將編碼值存入對照表陣列內，再來是文章A兩兩編碼並以對照表陣列內的值替換。也就是圖中的Step7與Step8。

最後在Step9時編碼完後全為-1便結束。

肆、實驗與討論

由於繁體中文字不似英文只用26個字母為字根，字母間比對容易，所以本研究的中文文章比對演算法有助於解決中文字數量龐大造成中文字串比對不易的結果，將兩篇文章相同的字串予以一個數字做為代碼，將文字轉換為數字(2bytes->1byte)，同時也縮減數字長度，增加字串比對效能。

依據中文文章的特性，我們可以忽略長度較短的中文字串不予理會，因為中文文章的特性常有一些連結詞例如但是、我的、他的、最後、而且、如果…等等字眼經常出現，針對中文文章的剽竊研究，相似的短詞語或是常用語詞其實並無太大意義；所以在壓縮後的文章比對上找出：

1. 兩篇報告相似度高的最長句子（>6字）。
2. 兩篇報告中相似度高的段落。
3. 兩篇報告做全文比對。
4. 依以上法則而產出的相似度報告在剽竊偵測上應該較為客觀正確。

同時，由於使用者對於剽竊的定義有各自的主觀認定，在法理上而言，抄襲有狹義與廣義之分，狹義之「抄襲」為著作權法所使用，專指「表達」之抄襲，不及於思想、概念、原理之抄襲。而廣義之「抄襲」則為學術界所用，泛指所有剽竊他人的著作當作自己所創作之行為。其可包括援用他人的思想或觀念，但卻未註明出處。但此思想上之抄襲並未違法著作權法之規定，純粹是違法學術倫理。所以在學術界而言，針對論文或報告的內容是否有抄襲這個問題見人見智，除了大篇幅照抄或是明顯的段落相同之外，並無一定標準。

所以我們以一種直觀但保守的想法，針對任兩篇中文文章內容，我們不直接以數量化的角度來對文章相似度做出評斷，而是以程度上的多寡來衡量，只要一旦達到某一程度的標準，我們就認定其相似程度為何，日後可依欲偵測文章性質的不同來更動評判標準。

一、實驗測試環境

（一）硬體環境:

CPU: Intel Core 2 CPU T5500 1.66GHz。

RAM: 1G。

（二）軟體環境:

Windows Vista。

Microsoft Visual Studio .Net 2005。

二、進行程式測驗

本研究我們對對文章相似度指標制定了一張相似標準表，如表11所示：

表11：相似標準表

相同字數	說明
0～6	可忽略不計
7～30	抄襲可能程度低，大約一句話相同。
31～50	抄襲可能程度中，大約二句話相同。
51～70	抄襲可能程度高，大約三至五句話相同。
71～150	抄襲認定，大約一個小段落皆相同(六句以上)。

依此相似標準表，我們設計以下實驗，分別輸入兩篇各200字與300字的中文報告，報告共分四個段落：

（一）實驗一：全文比對。

輸入兩篇完全相同的報告，計算其執行時間與兩篇皆出現相同且連續的句子的字數的出現次數。如附件一、附件二與附件三。實驗結果見表12所示。

（二）實驗二：段落比對。

輸入兩篇二、三段內容相同，但一、四段內容不同的的報告，計算其執行時間與兩篇皆出現相同且連續的句子的字數的出現次數。如附件一與附件四、附件二與附件五以及附件三與附件十。實驗結果見表13所示。

（三）實驗三：句子比對。

輸入每個段落中都有出現相同句子的報告，計算其執行時間與兩篇皆出現相同且連續的句子的字數的出現次數。如附件一與附件五以及附件二與附件六。實驗結果見表14所示。

（四）實驗四：全文比對。

輸入兩篇完全不同的報告，計算其執行時間與兩篇皆出現相同且連續的句子的字數的出現次數。如附件一與附件七以及附件二與附件八。實驗結果見表15所示。

三、程式測驗結果分析

實驗評估的結果與分析，從客觀程式執行時間及相同句子依字數的不同在兩篇文章之中出現次數多少來進行，並依據本研究之主觀認知所訂之剽竊標準，即表7相似標準表來對兩篇文章做相似度的判斷，實驗記錄如下：

表12：實驗記錄表-實驗一

實驗一：全文相同	程式執行時間(秒)	連續相同的句子的字數出現的次數(次)			
附件一 (211字)	0:00:27:0190000s	0～6字		7～30字	
		31～50字		51～70字	
		71～150字		全文相同	1
附件二 (315字)	0:01:56:9610000s	0～6字		7～30字	
		31～50字		51～70字	
		71～150字		全文相同	1
附件三 (406字)	0:06:36:7350000s	0～6字		7～30字	
		31～50字		51～70字	
		71～150字		全文相同	1

表13：實驗記錄表-實驗二

實驗二：段落相同	程式執行時間(秒)	連續相同的句子的字數出現的次數(次)			
附件一(211字)與附件四(206字)	0:00:02:0190000s	0～6字	5	7～30字	
		31～50字	2	51～70字	
		71～150字		全文相同	
附件二(315字)與附件五(312字)	0:00:23:3060000s	0～6字	8	7～30字	
		31～50字		51～70字	
		71～150字	2	全文相同	
附件三(406字)與附件十(420字)	0:04:10:3220000s	0～6字	13	7～30字	
		31～50字		51～70字	
		71～150字	2	全文相同	

表14：實驗記錄表-實驗三

實驗三：句子相同	程式執行時間(秒)	連續相同的句子的字數出現的次數(次)			
附件一(211字)與附件六(268字)	0:00:00:9880000s	0～6字	8	7～30字	3
		31～50字		51～70字	
		71～150字		全文相同	
附件二(315字)與附件七(329字)	0:00:03:0920000s	0～6字	22	7～30字	2
		31～50字		51～70字	
		71～150字		全文相同	
附件三(406字)與附件十一(415字)	0:00:03:7900000s	0～6字	20	7～30字	1
		31～50字		51～70字	
		71～150字		全文相同	

表15：實驗記錄表-實驗四

實驗四：全文不同	程式執行時間(秒)	連續相同的句子的字數出現的次數(次)			
附件一(211字)與附件八(229字)	0:00:00:5250000s	0～6字	4	7～30字	
		31～50字		51～70字	
		71～150字		全文相同	
附件二(315字)與附件九(358字)	0:00:02:2230000s	0～6字	7	7～30字	
		31～50字		51～70字	
		71～150字		全文相同	
附件三(406字)與附件十二(401字)	0:00:02:7550000s	0～6字	11	7～30字	
		31～50字		51～70字	
		71～150字		全文相同	

由實驗記錄可以得知，任兩篇文章只要有相同的內容程式皆可發現，在執行時間方面，兩篇全文相同的文章所需執行時間最長，文章長度為211字時需27秒，文章長度約315字時需時1分56秒，文章長度約406字時需時6分36秒，亦即，兩篇文章相同的內容越多，程式執行時間就越長，成正相關。

兩篇全文皆不相同的文章執行時間最短，在文章長度為211字時比對偵測時間僅需0.525秒，在文章長度為315字時比對偵測時間僅需2.223秒，在文章長度為406字時比對偵測時間僅需2.755秒，也就是兩篇文章相同的內容越少，程式執行時間就越短，成負相關。

不論兩篇文章字數內容多少，其找出相同內容字串之精確度皆為百分之百，在程式執行時間長度方面，在本研究所使用之環境下為使用者可接受的範圍內，於是可以推論，本研究撰寫之中文文章比對程式可達實務上使用者接受的程度。

針對中文文章偵測剽竊的研究，主要的爭議點便是在於每個人對於剽竊的認知具程度上的差異，本研究最大的貢獻在於提供一種比對中文文章的演算法，能夠抓出任兩篇文章相同的中文字句，而後提供一種相似度的評估概念，依據不同的文章性質、評估者對於剽竊認定不同的標準，彈性的更改剽竊認定的標準，以符合不同使用者的偵測剽竊所需。基於上述的概念，本研究總結討論如下：

(一)由於字典庫陣列依第一篇輸入的文章，在實驗時若將先後輸入的文章互換，則實驗結果不變，但第一篇輸入的文章字數若比較少，則執行速度較快；因為建立的字典庫陣列越小，執行時間就越短。所以輸入的第一篇文章最好選擇字數少的一篇。

(二)基本上，在做兩篇全文相同的文章比對時，由於文章內容全部相同，依本研究之演算法必須不斷的建立對照表陣列來建立簡單的編碼來代表複雜的句子，所以若兩篇文章全文相同，演算法執行的結果最後會將全文縮短成一個單一的數字編碼，而後再依對照表陣列不斷還原出相同的中文，最後還原出全文相同，所以執行速度最慢，所占的記憶體與CPU資源也最多。

伍、結論與研究限制

本由於本研究僅為試探性研究，對於剽竊之認定實為見人見智，所以實驗所建立之相似度標準表也僅為個人主觀所編。透過有效的剽竊偵測軟體來協助教授或助教批改學生報告時，如對報告內容資訊有不當抄襲或引用之疑慮時，降低其比對的複雜度，節省時間之餘同時對於學生對網路資源不當引用有嚇阻的效果，進而提升學生學習品質，透過師長們的努力引導學生創意性思考，學生們彼此的互相勉勵，期能形成一股崇尚創新的校園文化。

此外，由於本研究為試探性研究，僅以兩篇文章的比較演算法為起始點，將來系統改版將以可以多篇文章交互比對，做為學校報告或論文剽竊的偵測系統為主。本研究期能對降低對於學生中文報告剽竊比對的時間與人力成本，減少校園剽竊風氣，能夠有所貢獻。

致謝

本研究承蒙國家科學委員會提供部分經費支持，計劃編號為94-2815-C-259 -026 –H，謹此致謝。

參考文獻

1. 張真誠、蔡文輝，1991，『植基於藍波-立夫編碼法的中文資料壓縮技術』，電腦學刊，第三卷．第二期：13～19頁。
2. Chang, C. and Tsai, W. "A Data Compression Scheme for Chinese and English Characters," *Computer Processing of Chinese and Oriental Languages* (5:2), 1991 pp. 154-182.
3. Chen, X., Francia, B., Li, M. and McKinnon, B., A. Seker. "Shared Information and Program Plagiarism Detection," *EEE Transactions on Information Theory*, 2004, pp. 1545-1550.
4. Huffman, D. A Method for the Construction of Minimum Redundancy Codes," in *Proceedings of IRE* (40:9), 1952, pp. 1098-1101.
5. Lancaster, T. and Culwin, F. "A Comparison of Source Code Plagiarism Detection Engines," *Computer Science Education* (14:2), 2004, pp. 101-117.
6. McLafferty, C. and Foust, K. "Electronic Plagiarism as a College Instructor' s Nightmare-Prevention and Detection," *Journal of Education for Business* (79:3), 2004, pp. 186-190.
7. Ottenstein, K. "An algorithmic approach to the detection and prevention of plagiarism," *ACM SIGCSE Bulletin* (8:4), 1976, pp. 30-41.
8. Parker A. and Hamblen. J. "Computer algorithms for plagiarism detection," *IEEE Transactions on Education* (32:2), 1989, pp. 94–99.
9. Verco, K. L. and Wise, M. J. "Plagiarism à la Mode: A Comparison of Automated Systems for Detecting Suspected Plagiarism," *the Computer Journal* (39:9), 1996, pp. 741-750.
10. Wise, M. "Running Karp–Rabin Matching and Greedy String Tiling," Department of Computer Science, Sydney University, Sydney, Australia, 1994, Technical Report 463.
11. Ziv, J. and Lempel, A. "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory* (23:3), 1977, pp. 337-343.

附錄

一、實驗短文

（一）附件一（211字）

藍芽技術是一種小範圍的無線電頻率技術，裝置間透過晶片可互相溝通，不必再透過纜線傳輸。

目前，低功率的無線傳輸仍以紅外線為主，應用的層面仍侷限在行動電話、PDA、電腦及其相關產品上。

紅外線的傳輸的距離短、且受方向限制，用過筆記型電腦的紅外線埠和印表機連線的人可能有過這樣的經驗：

努力地調整筆記型電腦和印表機的角度（幾乎是面對面）和位置（一公尺內）後，發現其慢無比，最後還是乖乖的找條線接上電腦，是不是資料的傳遞一定得要經過銅線呢？

（二）附件二（315字）

抄襲，亦稱作學術剽竊、剽竊學術研究成果、違反學術誠信，是對於原著未經或基本未經修改的抄錄，這是一種侵犯著作權的行為。

但是一些時候是否構成抄襲比較難以界定，例如模仿一個故事的情節是否屬抄襲就有很大爭議，一些人認為故事情節屬於思想範圍，而「抄襲思想」並不是犯罪的行為，因為法律只保護思想的表現方式，而不是思想本身。

不同學校對於保持學術誠信有不同的要求。簡單一點的，會要求學生在提交習作時，要附有至少5個或10個參考書目。亦有學校會要求學生在遞交功課時連同一份有法律效力的聲明一同遞交。嚴格一點的，會禁止學生把同一份功課同時交給超過一個科目。

而對於違反學術誠信的學生，輕則會發警告信，重則會開除學籍。而對於違反學術誠信的教員，則會被撤職。

（三）附件三（406字）

隨著Google在去年初宣布於台灣啟動「雲端運算學術計畫」，「雲端運算」這個聽來帶點浪漫色彩的科技名詞立時席捲各大媒體版面。眾多網路公司以及「網格運算」服務都搶搭順風車，聲稱他們的服務也屬於「雲端運算」。但是，只怕很少人能夠聽明白他們口中的這朵「雲」代表著什麼玄機，以及它究竟要做什麼「運算」。

所謂「雲端」其實就是泛指「網路」，名稱來自工程師在繪製示意圖時，常以一朵雲來代表「網路」。因此，「雲端運算」用白話文講就是「網路運算」。舉凡運用網路溝通多台電腦的運算工作，或是透過網路連線取得由遠端主機提供的服務等，都可以算是一種「雲端運算」。

所以說，「雲端運算」其實不是新技術，更嚴格的說，甚至不能算是「技術」。「雲端運算」是一種概念，代表的是利用網路使電腦能夠彼此合作或使服務更無遠弗

屆。而在實現「概念」的過程中，才會產生出相應的「技術」。

「雲端運算」的概念事實上也不算新，其本質大抵承襲自「分散式運算」(Distributed Computing)以及「網格運算」。

（四）附件四（206字）

奈米科技實際上並無統一的定義，一般說法係指物質在奈米尺寸下呈現出有別於巨觀尺度下的物理、化學或生物特性與現象。

目前，低功率的無線傳輸仍以紅外線為主，應用的層面仍侷限在行動電話、PDA、電腦及其相關產品上。

紅外線的傳輸的距離短、且受方向限制，用過筆記型電腦的紅外線埠和印表機連線的人可能有過這樣的經驗：

奈米科技的最終目標是依照需求，透過控制原子、分子在奈米尺度上表現出來的嶄新特性，加以組合並製造出具有特定功能的產品。

（五）附件五（312字）

澳洲原住民藝術在藝術市場上愈顯重要，有些非原住民的澳洲人因此剽竊原住民的原創藝術，讓外界誤以為他們的作品也是原住民藝術。

但是一些時候是否構成抄襲比較難以界定，例如模仿一個故事的情節是否屬抄襲就有很大爭議，一些人認為故事情節屬於思想範圍，而「抄襲思想」並不是犯罪的行為，因為法律只保護思想的表現方式，而不是思想本身。

不同學校對於保持學術誠信有不同的要求。簡單一點的，會要求學生在提交習作時，要附有至少5個或10個參考書目。亦有學校會要求學生在遞交功課時連同一份有法律效力的聲明一同遞交。嚴格一點的，會禁止學生把同一份功課同時交給超過一個科目。

如日本、印度或中東地區。這類題材剛問世的時候，是相當鮮明，也容易引來其他剽竊者的學習。

（六）附件六（268字）

奈米科技實際上並無統一的定義，裝置間透過晶片可互相溝通一般說法係指物質在奈米尺寸下呈現出有別於巨觀尺度下的物理、化學或生物特性與現象。

所謂奈米科技便是運用這方面的知識，在奈米尺寸等級的微小世界中操作、控制原子或分子組合成新的奈米尺度結構（電腦及其相關產品上奈米材料），以便展現新的機能與特性。

以此為基礎，設計、製作、組裝成新的材料、器具或系統，使之產生全新功能，並加以利用的技術總稱。紅外線的傳輸的距離短，

奈米科技的最終目標是依照需求，透過控制原子、分子在奈米尺度上表現出來的嶄新特性，加以組合並製造出具有特定功能的產品。幾乎是面對面。

（七）附件七（329字）

澳洲原住民藝術在藝術市場上愈顯重要，有些非原住民的澳洲人因此剽竊原住民的原創藝術，讓外界誤以為他們的作品也是原住民藝術。學術剽竊近日部分立法委員指稱

新政府若干首長擔任教職期間涉嫌抄襲、剽竊學生論文，或由學生捉刀撰寫論文後，作為提出申請國科會研究案補助之用，乃要求下台，以示負責。消息傳布後，一方面有人主張故事情節屬於思想範圍長期以來之抄襲、剽竊、捉刀、教師欺壓學生之惡風，也有人為文認為此乃學術界師生共同研究學習之當然作法。

對於記者問及助理為該立法委員擬法案及質詢稿時之情形與本次師生間論文掛名抄襲之爭議有何不同之時，會要求學生在提交習作有加，助理們均自願協助之另類答案。

如日本、印度或中東地區。會開除學籍。這類時候，是相當鮮明，也容易引來其他剽竊者的學習。

（八）附件八（229字）

奈米科技實際上並無統一的定義，一般說法係指物質在奈米尺寸下呈現出有別於巨觀尺度下的物理、化學或生物特性與現象。

所謂奈米科技便是運用這方面的知識，在奈米尺寸等級的微小世界中操作、控制原子或分子組合成新的奈米尺度結構（奈米材料），以便展現新的機能與特性。

以此為基礎，設計、製作、組裝成新的材料、器具或系統，使之產生全新功能，並加以利用的技術總稱。

奈米科技的最終目標是依照需求，透過控制原子、分子在奈米尺度上表現出來的嶄新特性，加以組合並製造出具有特定功能的產品。

（九）附件九（350字）

全球暖化指的是在一段時間中，地球的大氣和海洋因溫室效應而造成溫度上升的氣候變化現象，為公地悲劇之一，而其所造成的效應稱之為全球暖化效應。

在20世紀時，全球平均接近地面的大氣層溫度上升了攝氏0.74度。[1]普遍來說，科學界發現過去50年可觀察的氣候改變的速度是過去100年的雙倍，有理由認為該時期的氣候改變是由人類活動所推動。[1]

二氧化碳和其他溫室氣體的含量不斷增加。[2]正是全球暖化的人為因素中主要部分。燃燒化石燃料、清理林木和耕作等等都增強了溫室效應。自從1950年，太陽輻射的變化與火山活動所產生的變暖效果比人類所排放的溫室氣體的還要低。[3][4]這些結論得到30多個來自八大工業國家的研究團體所確認。[5][6]

美國加利福尼亞大學的科學家在太平洋中央夏威夷的茂納羅亞峰上設立4個7米高和一個27米高的採樣塔，每小時採樣4次，分析二氧化碳的變化情況。

（十）附件十（420字）

問題來了，若說只要是透過網路線接上「雲端」並利用遠端資源就可以稱做「雲端運算」，那麼上Gmail收發信件與利用BitTorrent之類的P2P技術取得資料，豈不都可算是「雲端運算」？但是這兩者在本質上有著明顯的不同，究竟何者才能算是「正港」的「雲端運算」呢？

所謂「雲端」其實就是泛指「網路」，名稱來自工程師在繪製示意圖時，常以一朵雲來代表「網路」。因此，「雲端運算」用白話文講就是「網路運算」。舉凡運用網路溝通多台電腦的運算工作，或是透過網路連線取得由遠端主機提供的服務等，都可以算

是一種「雲端運算」。

所以說，「雲端運算」其實不是新技術，更嚴格的說，甚至不能算是「技術」。「雲端運算」是一種概念，代表的是利用網路使電腦能夠彼此合作或使服務更無遠弗屆。而在實現「概念」的過程中，才會產生出相應的「技術」。

Gartner指出，「雲端服務」專注在於藉由網路連線從遠端取得服務。例如提供使用者安裝和使用各種不同作業系統的Amazon EC2服務。這類型的雲端計算可以視為「軟體即服務」(SaaS, Software as a Service)概念的後繼。

（十一）附件十一（415字）

太陽風是從恆星上層大氣射出的超音速電漿雲端運算學術計畫流。在不是太陽的情況下，這種帶電粒子流也常稱為「恆星風」。

在太陽的日冕層的高溫（幾百萬開氏度）下，氫、氦等原子已經被電離成帶正電的質子、氦原子核和帶負電的自由電子等。這些帶電粒子運動速度極快，以致不斷有帶電的粒子掙脫太陽的重力束縛，射向太陽的外圍，形其實就是泛指成太陽風。 一般在200-800km/s。 一般認為在太陽極小期，從太陽的磁場極地附近吹出的是高速太陽風，從太陽的磁場赤道附近吹出的是低速太陽風。太陽的磁場的活動性是會變化的，周期大約為22年。

太陽風一詞是在1950年代被Parker提出。但是直到1960年代才證實了它的存在。長期觀測發現，當太陽存在冕洞時，地球附近就能觀測到高速的太陽風。利用網路使電腦因此天文學家認為高速太陽風的產生與冕洞有密切的關係。太陽表面的磁場及電漿活動對地球有很重要的影響。

當太陽發生強烈的活動時，大量的帶電粒子隨著太陽風吹向地球的兩極，就會在兩極的電離層引發美麗的極光。分散式運算。

（十二）附件十二（401字）

太陽風是從恆星上層大氣射出的超音速電漿（帶電粒子）流。在不是太陽的情況下，這種帶電粒子流也常稱為「恆星風」。

在太陽的日冕層的高溫（幾百萬開氏度）下，氫、氦等原子已經被電離成帶正電的質子、氦原子核和帶負電的自由電子等。這些帶電粒子運動速度極快，以致不斷有帶電的粒子掙脫太陽的重力束縛，射向太陽的外圍，形成太陽風。 太陽風的速度一般在200-800km/s。 一般認為在太陽極小期，從太陽的磁場極地附近吹出的是高速太陽風，從太陽的磁場赤道附近吹出的是低速太陽風。太陽的磁場的活動性是會變化的，周期大約為22年。

太陽風一詞是在1950年代被Parker提出。但是直到1960年代才證實了它的存在。長期觀測發現，當太陽存在冕洞時，地球附近就能觀測到高速的太陽風。因此天文學家認為高速太陽風的產生與冕洞有密切的關係。太陽表面的磁場及電漿活動對地球有很重要的影響。

當太陽發生強烈的活動時，大量的帶電粒子隨著太陽風吹向地球的兩極，就會在兩極的電離層引發美麗的極光。