國立臺灣海洋大學

資訊工程學系

碩士學位論文

指導教授：張雅惠博士

根據道路挖掘開放資料之路徑規劃研究

The Study on Route Planning According to the Open Data of Road Excavation

研究生：葉詩盈 撰

中華民國 106 年 06 月

根據道路挖掘開放資料之路徑規劃研究

The Study on Route Planning According to the Open Data of Road Excavation

研究生：葉詩盈　　Student：Shi-Ying Ye

指導教授：張雅惠　　Advisor：Ya-Hui Chang

國立臺灣海洋大學

資訊工程學系

碩士論文

A Thesis

Submitted to Department of Computer Science and Engineering

College of Electrical Engineering and Computer Scienece

National Taiwan Ocean University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science and Engineering

June 2017

Keelung, Taiwan, Republic of China

中華民國 106 年 06 月

摘要

近年來台灣政府積極推動開放政府之開放資料政策，整合各局處的開放資料於單一入口網站，供市民瀏覽及程式開發者加值應用。本論文特別針對桃園市政府所提供的道路挖掘開放資料，探討如何規劃一條避開道路封閉區塊且最短的路線。先前，研究[洪 16]曾經建立一個基於雲端服務且能避開淹水區塊的路徑規劃系統。本論文主要根據該研究所提出的架構，但改為結合道路挖掘開放資料，主要差異在於必須分析開放資料，並採用關聯式資料庫系統以管理地理資料。除此之外，在[洪 16]架構中的雲端路徑處理模組，曾經提出 Join_Point 演算法刪減雲端服務所傳回的資料中太過密集的點，以提升整體效率，但有時因此無法正確判斷某條路段是否可以行走。在本論文中，我們也提出 Join_Pro 演算法來改善該演算法的準確率問題。我們實作整個系統並進行一系列的實驗。實驗結果顯示 Join_Pro 演算法的確比 Join_Point 演算法有更高的準確率，且達 90%以上。另外，針對地理資料的處理，所採用的幾何型態也比字串型態有更佳的效率。

關鍵詞：開放資料、地理資料處理、路徑規劃

Abstract

In recent years, the Taiwanese government has actively promoted the open data policy, and integrated the open data provided by different government departments into a single portal site for public viewing and developing value-added applications. This thesis specifically processes the road excavation information provided by the Taoyuan city government, and discusses how to plan a short route which detours the closed blocks. A previous study had constructed a path planning system based on cloud services that could avoid flooded blocks. This paper is based on that system, but deals with the open road excavation data instead. The main difference is that we need to analyze open data and use a relational database system to manage geographic data. In addition, the Cloud Path Processing module in the previous system consists of the Join_Point algorithm to remove unnecessary dense points in the data returned by the cloud service to improve overall efficiency, but sometimes it cannot correctly determine whether a road can pass or not. In this paper, we additionally propose the Join_pro algorithm to improve the accuracy. We have implemented the whole system and carried out a series of experiments. Experimental results show that the Join_pro algorithm obtains a higher accuracy rate than the Join_Point algorithm does, and the rate is more than 90%. In addition, the geometric data type used for representing geographic data in the relational database can provide more efficient querying facilities than the string type does.

Keywords: open data, geographic data processing, path planning

誌謝

首先，感謝指導教授張雅惠博士，對於本論文給予許多幫助，且在研究論文期間不時地共同討論，解決論文諸多的問題，使學生能順利的完成論文。

除此之外，感謝臺北科技大學劉傳銘博士和林川傑博士百忙之中抽空參與論文審查工作，也感謝蔚齊學長給予本論文意見與幫助，使論文更趨近完善。

最後，感謝實驗室學長，帶領著我適應實驗室環境以及提供各種不同方法解決困境。感謝我的同學文瀚與學弟妹們，一起陪伴著我渡過研究所生涯歷練與成長。更要感謝我最愛的家人們，在學習階段給予無限的支持和精神上的鼓勵，讓我能夠無憂的專注於研究，在此一併致上謝意，謝謝你們。

目錄

圖目錄

表目錄

第 1 章 序論與技術背景

在此章，我們先說明本論文的研究動機與目的，以及提出本論文的研究方法與貢獻，並介紹相關的研究，最後說明各章節的內容及本論文的架構。

1.1 研究動機與目的

近年來台灣政府積極推動開放政府(Open Government)之資料開放政策，整合各局處的開放資料(Open Data)於單一入口網站，供市民瀏覽及程式開發者加值應用。除此之外，台灣地區經常因為頻繁的天然災害(如淹水、土石流...等)以及公共工程(如搶修管線、道路養護、建設需求...等)而造成道路因為預警性、災害性及施工而封閉。因此本論文探討如何將政府所提供的道路挖掘開放資料結合路徑規劃系統，提前規劃一條避開道路封閉且最短的路線。

在之前的研究中[洪 16]探討如何利用 Google Maps API 和建立索引的方法規劃迴避淹水區域的路徑，整個路徑規劃系統架構，如圖 1-1 所示。首先，當使用者輸入起迄點，路徑規劃系統的主程式會呼叫『雲端路徑處理模組』查詢最短路徑。接著呼叫『淹水區域查詢模組』，判斷這條路徑是否有跟任何一個淹水區域重疊，若有則輸出該條道路路口點，然後呼叫『鄰近點篩選模組』，查詢附近未碰到淹水區域的路口點。該系統將起點、路口點的鄰近點以及終點分類。最後，執行 GSP 演算法選出每個種類代表的點當作 waypoint，waypoint 表示從起點到終點途中必須經過的點集合，並再次呼叫『雲端路徑處理模組』查詢起點經過 waypoint 到目的地的最短路徑，若還是不能通行，就須修正此路徑，也就是從種類表中刪除 waypoint，再重新呼叫 GSP 演算法以及『雲端路徑處理模組』，直到路徑不經過淹水區域或修正次數為 n。而為了能夠讓使用者便於觀看最後輸出的路徑，查詢結果以網頁方式呈現，如圖 1-2。

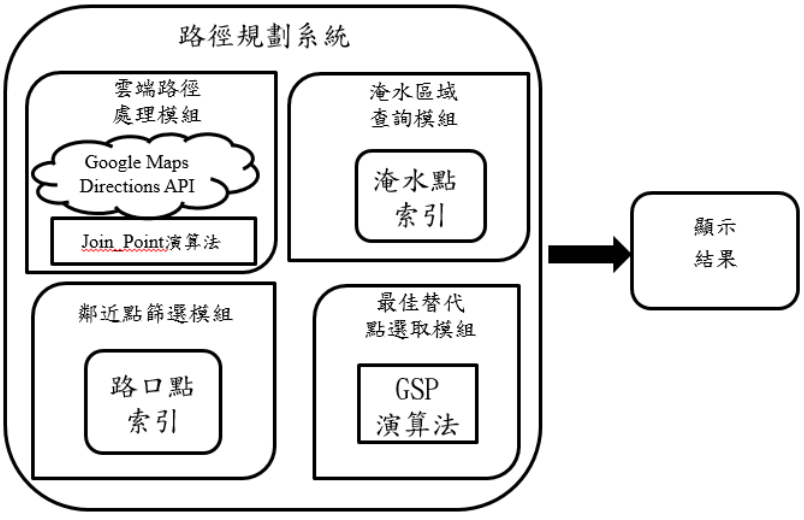

圖 1-1　迴避淹水區域的路徑規劃系統[洪 16]

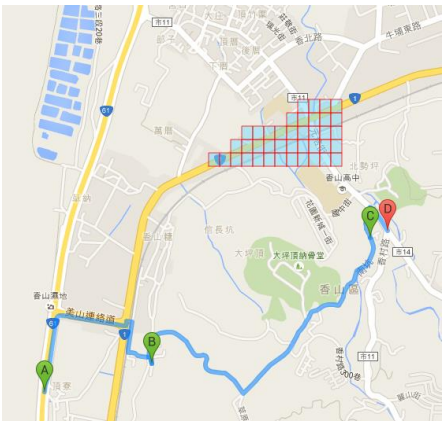

圖 1-2　迴避淹水的路徑

而我們的研究目的是，將政府所提供的道路挖掘開放資料結合圖 1-1 的路徑規劃系統，規劃一條繞過道路挖掘區塊且最短的路線。如圖 1-3 所示，粗黑線區域為一筆道路挖掘區域，左子圖為碰到道路挖掘區域的路徑，右子圖為修正後的路徑。

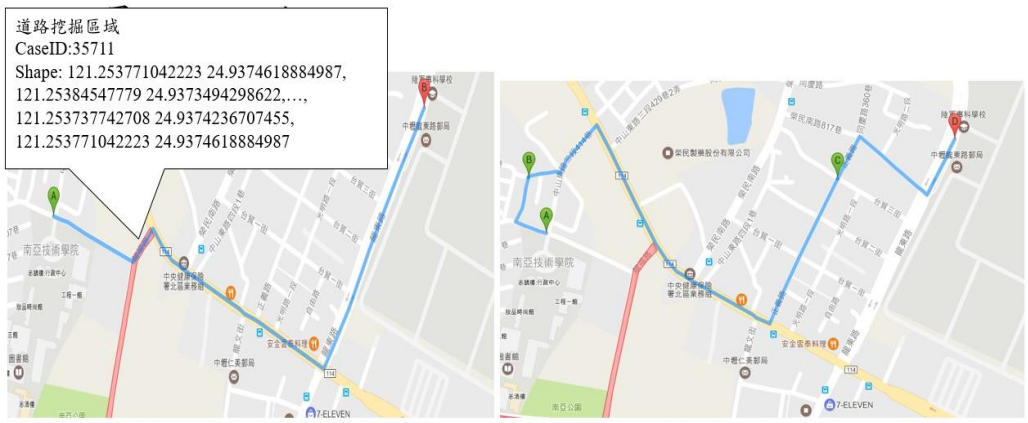

圖 1-3 研究目的示意圖

1.2 研究方法與貢獻

此研究的主要議題，在於如何整合政府所提供之道路挖掘開放資料並結合路徑規劃系統，快速的規劃出一條避開道路挖掘區塊的最短路徑。我們所提出的方法是將介接取得的開放資料經過資料分析後建立於資料庫中，再針對預計結束時間進行資料庫維護，將小於系統時間的資料從資料庫中刪除，藉此達到即時且快速的路徑規劃。

另外我們提出 Join_Pro 演算法，以改善圖 1-1 中 Join_Point 演算法的準確率問題。該作法是只要兩點小於合併距離值就刪除左邊點，但這個作法並未考慮到所有路徑情況，因此本論文針對 Google Maps API 所回傳的點，整理出了 4 種性質，並提出 Join_Pro 演算法來解決這 4 種性質。

1.3 相關研究

我們首先討論示警資料處理之相關議題。[陳 15]以 5W1H 分析法歸納示警使用者期望獲取之示警資訊內容及滿足有效管理所必須滿足之條件的示警之核心項目，並以核心資訊分析不同示警標準項目或內容之設計成果，各項目有其記錄之規定，若項目與內容皆可成功對應，即表示兩者有結合運作的可能性，最後提出整合多來源示警之開放架構，除了可確保其提供必須之示警內容，且明定示警資訊上架及下架之條件，如此，便可進一步規劃不同展示介面之示警展示或運作模式。[蘇 15]基於標準劃描述之觀點，分析現象之時間、空間及屬性之變化，提出標準化時空記錄架構，提供資料建置者一套描述時空現象之工具，並藉由分析資料時間之異質性，全面考量了因描述現象時間不同、現象合適之描述方式不同、記錄之時間解析度不同等情況，提出一套時間異質性判斷機制，強化地理資訊系統於時間方面的之處理能力。

其次，我們討論有關開放資料的相關議題。隨著政府最近提出向公眾增加開放資料的政策，使得開放資料穩定的增長，但由於不同政府部門製定的資料具有不同的時間和空間覆蓋率，因此整合這些開放資料資源成為了重要課題，在[ZY16]中，提出了一種基於空間網格的方法來存儲和管理巨量的開放資料，其作法是將每種類型的政府資料都作為每個網格的屬性儲存，而資料確切性程度和應用程序的目的決定每個網格的大小，接著使用陣列資料庫來儲存並管理存有政府開放資料的空間網格。[MH16]則是提出資料轉換和聚合平台（Data-TAP）來整合現有資料庫和已存在的開放資料平台的資料，它提供了資料標準化和結構化的方法，並透過易於理解的界面來定義解析規則進行解析資料，以符合使用者的需求。而這個平台使用了兩種資料庫，分別為負責提供應用程序資料的關聯式資料庫 Postgres SQL 和用於儲存用戶定義的資料集模型結構和解析規則的 NoSQL MongoDB 資料庫。

接著，在處理道路壅塞的問題時，研究者也常利用路徑規劃的技術，以下我們討論幾篇相關論文。論文[CJZG16] 提出一種基於訊息(pheromone)的交通管理框架。其中，車輛重新路徑規劃的策略是先利用每個車輛所存放的意圖訊息，即為未來會走的道路，選出需要重新規劃路徑的車輛，接著利用這些車輛的總行程距離來計算出k個較短的路徑，然後根據一定的概率隨機選擇一個路徑以迴避道路壅塞。論文[WDZ16]則是為了迴避事故引起的不可預測的交通壅塞提出了一種車輛重新路徑規劃的策略，稱為下段路重新路徑規劃（NRR，Next Road Rerouting）。其作法是當突發事件發生時，透過智能交通燈(iTL)，向進入道路中的所有車輛發出重新規劃路徑的警報。接著，對於每個重新規劃路徑請求，iTL收集當前的位置訊息與地圖（即iTL控制的所有路口道路）來計算每個可能的下一個道路選擇的路徑成本。隨後，通過傳回的重新規劃路徑結果，建議成本最低的一個。最後，在車輛進入NRR建議的最佳下一條道路後，藉助其在線車載導航系統(VNS)重新計算其餘旅程的路線。而路徑成本考慮以下四個因素，分別為道路佔有率、行車

時間、新路徑到目的地的距離，新路徑與壅塞道路的地理接近度。論文[PPB17]提出車輛分散式重新路徑規劃系統(DIVERT)，車輛透過蜂窩網路與伺服器進行通訊，報告當前的交通密度資料並從伺服器接收路網中的全部的交通密度，伺服器會將這些車輛報告，建立出一個有向路網圖，每個邊皆具有當前交通密度的動態權重。當某個路段發生壅塞時，伺服器會將記錄了當前權重的路網圖傳送給接近壅塞路段的車輛，使車輛透過路網圖中的權重，重新計算當前位置到目的地的新路線，由此實現分散式重新路徑規劃。論文[SYMLV16] 提出了一個智能交通系統CIDMERA適當的重新規劃車輛，以保持交通負荷的平衡。此系統使用k-NN演算法檢測交通情況，以道路的平均行車速度和車輛密度作為輸入的參數，將道路雍塞分類定義為自由流量、輕度壅塞、適度雍塞、嚴重雍塞。當檢測到壅塞的道路時，會定義一個範圍，在這個範圍內的車輛都會經過壅塞道路，接著，使用重新路徑規劃演算法計算新路線以避免車輛通過壅塞道路。而重新路徑規劃演算法使用了車輛的當前位置作為起點，路邊單位(RSU)覆蓋範圍內的車輛，其路線的最後邊作為終點。接著，使用基於道路權重的K-Shortest Path算法來計算k個替代路線。最後，使用Boltzmann概率分佈為不同的車輛分配不同的路徑，從而在替代路徑上維持負載平衡。

至於[RT13]則討論 GSP（Generalized Shortest Path）的問題，該問題討論的是路徑必須經過特定的群組順序，如必須先經過餐館再經過加油站，該論文首先提出基本的動態規劃法，接著再提出改善的方法，如最佳化問題結構、最佳化圖形結構、修剪預估不必要的路線的方法等。該論文提出 Forward Dijstra 方法，方法是將所有節點標上 VertexID，從起點 V_s 開始搜尋大於 V_s 的 VertexID 之鄰近邊，假設 N 是網路中所有節點，則固定會執行 N-1 次，以及 Backward Dijstra 方法，也就是利用終止點 V_t 向起始點方向搜尋，執行次數為比 N 小，和修剪過後的方法(設定 a 參數來修剪多餘的節點樹枝)，實驗結果顯示在不同情況下各方法有著不同好處，其中向前向後的 Dijkstra 方法是 RTO-GSP(起始、終止皆為自己)最好的解法，其餘情況修剪過後的方法是最好的。

1.4 論文架構

本論文其餘各章節的架構如下：第二章介紹本論文所研究的開放資料來源與處理，以及資料庫的表格定義、維護與更新。第三章介紹本論文所結合的路徑規劃系統，說明其整體架構及如何利用 Google Maps API 所提供的路徑規劃服務快速找出迴避道路挖掘的路徑。於第四章中以實驗比較開放資料處理與路徑規劃系統中各模組的效率及成功率，並於第五章提出本論文的結論與未來方向。

第 2 章 開放資料的介紹與資料處理

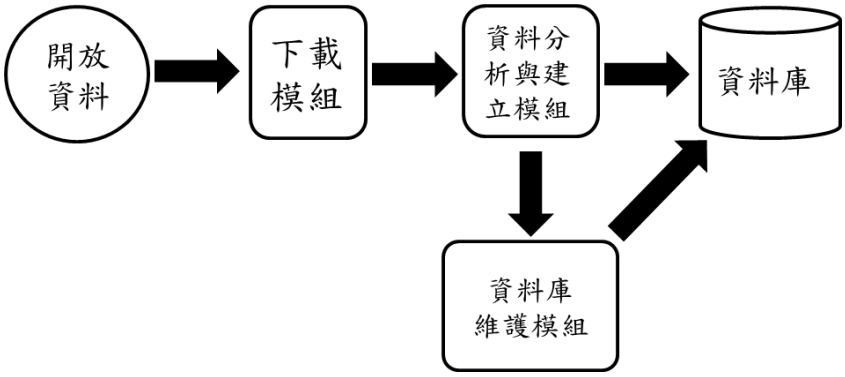

圖 2-1　開放資料處理架構圖

本論文提出的開放資料處理架構圖如圖 2-1 所示。首先經過『下載模組』每日定時下載道路挖掘開放資料，接著呼叫『資料分析與建立模組』來取得本論文所需的資料，並利用 SQL 查詢句將新資料建立於資料庫中且將舊資料做更新，最後執行『資料庫維護模組』將結束時間小於系統時間的資料刪除。

在所提出的系統中，我們使用微軟所推出的 SQL Server 規劃了一個關聯式資料庫，以便於日後建立與維護道路挖掘的開放資料。關聯式資料庫(Relational Database，RDB)是最被廣為使用的資料維護軟體，主要原因是其可以使用 SQL(Structured Query Language)進行各種不同的查詢，且提供快速的查詢處理，以及方便的資料維護介面。以下分節說明本系統所處理與定義的資料格式與各模組設計。

2.1　開放資料的來源與格式

本論文所處理的開放資料來自桃園市政府開放資料平台所提供的道路挖掘資訊[1]，其檔案格式為 XML。每筆檔案都包含多筆道路挖掘的記錄(records)，每筆紀錄的基本格式包含行政區(Addtownship)、施工範圍(Shape)、施工開始時間(Start)、工程單位(Factory)、施工編號(CaseID)、負責人(AppMan)、施工時段(ConstTime)、申請機關(PPBName)、施工類型(TypeDetail)、工程地點(SLocation)、工程名稱(ConstName)、施工結束時間(stop)、管理單位(LBName)
、連絡電話(Tel)，如圖 2-2 所示。

[1] 道路挖掘資訊 http://data.tycg.gov.tw/opendata/datalist/datasetMeta?oid=56c616fe-07d7-4b0c-bb75-e8f8cd75500a，

```
…
<records>
    <Addtownship> 桃園區</Addtownship>
    <Shape>121.277473172599,25.0262209067394;
            121.27803725202,25.025257214288;
            121.27800048896,25.0252310069248;
            121.277473172599,25.0262209067394
     </Shape>
    <Start> 2015/05/05 </Start>
    <Factory> 泰誠發展營工程股份有限公司</Factory>
    <CaseID> 27380 </CaseID>
    <AppMan> 黃致豪 </AppMan>
    <ConstTime>
        施工時段: 日間 7 時-17 時  夜間 19 時-24 時  凌晨 0 時-7 時
    </ConstTime>
    <PPBName> 日鼎水務企業股份有限公司</PPBName>
    <TypeDetail> 新設</TypeDetail>
    <SLocation>
       桃園區鄉道：桃 15 號(線)道路 4 公里+500 公尺
    </SLocation>
    <ConstName>
        促進民間參與桃園市桃園地區污水下水道系統建設之興建營運移轉
        計畫(BOT)
    </ConstName>
    <stop> 2017/12/31 </stop>
    <LBName> 工務局 </LBName>
    <Factory_Man_Tel> 0922936522 </Factory_Man_Tel>
    <Tel> (02)7733-8888 </Tel>
    <_id> 1 </_id>
</records>
…
```

圖 2-2　道路挖掘資訊

而本論文使用到其中四個標籤，包含施工範圍(Shape)、施工編號(CaseID)、施工時段(ConstTime)、施工結束時間(stop)，一筆範例資料如表 2-1 所示。其中，施工範圍記錄的是多邊形區塊，其格式為(經度 1,緯度 1;經度 2,緯度 2;…;經度 1,緯度 1)，若無資料則會以“NoShape”表示。

表 2-1　道路挖掘的範例資料

標籤名稱	舉例	格式說明
CaseID	27380	施工編號
stop	2017/12/31	施工結束時間
ConstTime	施工時段: 日間 7 時-17 時　夜間 19 時-24 時 凌晨 0 時-7 時	施工時段
Shape	121.277473172599,25.0262209067394; 121.27803725202,25.025257214288; 121.27800048896,25.0252310069248; 121.277473172599,25.0262209067394	施工範圍

2.2　各模組說明

首先說明架構中之『下載模組』，由於桃園市政府開放資料的更新時間為每日的凌晨 2 點 15-16 分左右[2]，因此使用 C#語言 Timer()函數，間隔 1 分鐘抓取系統時間，若系統時間為 2 點 16 分則開始執行下載模組。

而本論文的下載模組是使用桃園市政府開放資料平台提供的介接 API 網址[3]來取得平臺上符合條件之資料資源內容，介接 API 網址的介紹如下:
http://data.tycg.gov.tw/api/v1/rest/datastore/{resource ID}? format ={format}& limit ={limit}& offset ={offset}。其中 datastore 表示資料資源內容。{}內表示需要的資料條件，而 resource ID 表示資料資源編號，其為開放資料平台所提供的編號“52de3762-1490-4a86-a074-0062d746873b”。format 表示資料的存取格式，我們將其設定為“xml”。limit 表示指定最多回傳的資料筆數，API 預設為回傳 100 筆資料，而我們將其設定為 500 筆。offset 指定從第幾筆後開始回傳，可與 limit 配合使用達到分頁目的，預設值為 0。下載的檔案副檔名為.txt，因為內容格式為 XML，我們使用 C#程式語言內的 XML 處理套件 XmlTextReader()函數來取得 XML 資料。

在『資料庫維護模組』方面，本論文使用關聯式資料庫的 SYSDATETIME()函數查詢指令得到系統時間，將「Stop」欄位值小於當下系統時間的資料從資料庫中刪除。

[2] 藉由每日記錄對應附註 1 網頁的最後更新時間，歸納出道路挖掘資訊的每日更新時間約為凌晨 2 點 15-16 分。

[3] 道路挖掘介接 API 網址 http://data.tycg.gov.tw/api/v1/rest/datastore/52de3762-1490-4a86-a074-0062d746873b?format=xml &limit=500&offset=0

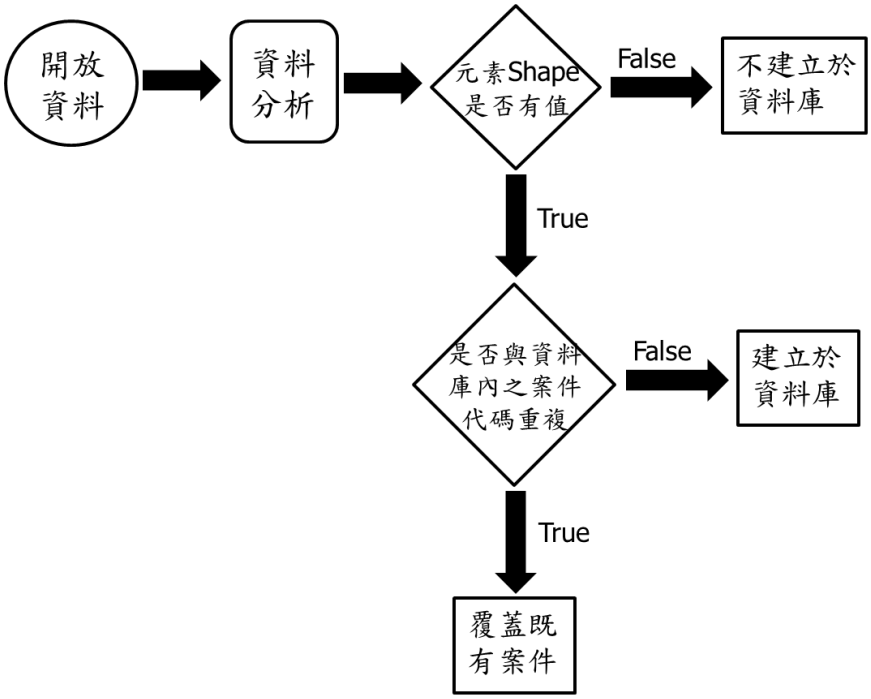

圖 2-3　資料分析與建立模組流程圖

接著說明『資料分析與建立模組』的流程圖，如圖 2-3 所示，我們先將下載的開放資料做資料分析，分析出本論文所需的標籤及其值，並將施工時段與施工範圍進行分割，以利於資料庫的存取與 SQL 查詢句的使用。由於施工範圍可能為“NoShape”，因此我們會先判斷「Shape」欄位是否有值，若為“NoShape”表示無值則不建立於資料庫內。反之，則判斷此筆資料是否與資料庫內之案件代碼重複，這是因為道路挖掘工程可能在結束期限時還未完工，而需延後結束時間，此時就會出有相同案件編號的情況，因此我們需判斷是否有相同的案件編號已經建立於資料庫中，若案件已存在於資料庫中，則利用新資訊覆蓋該筆資料，反之，則將此筆資料新增至資料庫中。

至於資料分析的方法，本論文使用文件物件模型 (DOM)的技術來處理所下載的開放資料，因為我們所下載的資料格式為 XML，而 DOM 提供程式設計方式來表示 XML 文件、片段、節點或節點集。首先我們使用 DOM 的 Load()函數將開放資料從檔案載入到 DOM 物件中，接著使用元素物件 element 方法中的 getElementsByTagName()函數來來指向表 2-1 所列的四個標籤並取得這四個標籤的值。而為了配合後面路徑規劃系統所使用的 SQL 查詢句，我們將 ConstTime 切割為 Morning、Night、Midnight 三個施工時段。此外，為了符合微軟 SQL Server 中 Geometry 空間資料類型的存取規則，我們需要改變 Shape 的儲存格式，也就是將 Shape 原本的儲存格式(經度 1,緯度 1;經度 2,緯度 2;…;經度 1,緯度 1)改為(經度 1 緯度 1, 經度 2 緯度 2,…,經度 1 緯度 1)，處理完的範例資料如表 2-2 所示。最後我們將資料依照此結構建立於資料庫中，詳見下節說明。值得注意的是，由於每筆資料會有不同的施工時段，Morning、Night、Midnight 三個施工時段都有可能為空值，為了日後資料庫的使用與維護我們將空值以“null”表示。

表 2-2　分析後的範例資料

標籤名稱	舉例	格式說明
CaseID	27380	施工編號
stop	2017/12/31	施工結束時間
Morning	7 時-17 時	施工時段(日間)
Night	19 時-24 時	施工時段(夜間)
Midnight	0 時-7 時	施工時段(凌晨)
Shape	121.277473172599 25.0262209067394, 121.27803725202 25.025257214288, 121.27800048896 25.0252310069248, 121.277473172599 25.0262209067394	施工範圍

2.3 資料庫綱要與維護

我們為開放資料所設計的表格定義如表 2-3 所示，其中主鍵以底線標示。表格 TY_roadclose 主要是記錄道路挖掘的開放資料，包含了案件編號(CaseID)、施工結束時間(stop)、日間的施工時段(Morning)、夜間的施工時段(Night)、凌晨的施工時段(Midnight)以及施工範圍(Shape)，其中案件編號、施工結束時間不可為空值，而日間、夜間、凌晨的施工時段若為“null”，則表示某個時段無施工。值得注意的是「Shape」欄位我們定義為“geometry”幾何資料類型，這是因為此資料類型可以記錄以座標系統表示的資料，且支援後面介紹的交集函數 **STIntersects()**等。

表 2-3　挖掘資料的表格定義

表格名稱: TY_roadclose		
欄位名稱	資料類型	說明
CaseID	nvarchar(50)	紀錄資料的案件 ID
stop	date	紀錄施工的結束時間
Morning	nvarchar(50)	紀錄日間的施工時段
Night	nvarchar(50)	紀錄晚間的施工時段
Midnight	nvarchar(50)	紀錄凌晨的施工時段
Shape	geometry	紀錄施工的影響範圍

此外，本論文還使用到交通部路網數值圖服務網[4]的資料以便做路徑規劃，我們將其中桃園市的路網圖事先建立於資料庫中，路網圖的 schema 如表 2-4 所示。值得注意的是「Node」欄位我們也定義為“geometry”類型，這是因為此資料類型可以記錄以座標系統表示的資料，且支援後面介紹的距離函數 **STDistance()**。

[4] 交通部路網數值圖服務網 https://gist-map.motc.gov.tw/

表 2-4 道路節點資料的表格定義

表格名稱: Tyu_node		
欄位名稱	資料類型	說明
NID	varchar(50)	紀錄道路節點的編號
Node	geometry	記錄節點的經度、緯度

由於本論文的基本想法是希望透過政府開放資料平台來更新施工資訊，以達成即時路徑規劃的目標，因此本論文透過 SQL 查詢句對資料庫內的道路封閉資料表做建立與維護，而我們之後會用到的 SQL 查詢句介紹如下:

[查詢句 2-1] 我們使用 **STIntersects()**函數來取出與某一物件「?Text」相交的道路挖掘區塊，其中道路挖掘區塊取自資料表 TY_roadclose 中的 Shape 欄位，對應的查詢句如下:

```
Select   [TY_roadclose].[Shape]
From     [TY_roadclose]
Where    geometry ::STGeomFromText('?Text', 4326).STIntersects
         ([TY_roadclose].[Shape])
```

注意到，由於我們的道路挖掘區塊是以 Geometry 空間資料格式存取，因此欲查詢的物件「?Text」也需轉換成 Geometry 格式。而我們使用 geometry::STGeomFromText('**Text**', 4326)來進行轉換，其中 4326 為座標系統的編號，而物件「?Text」可輸入的型態主要有 3 種，分別為點 Point、線段 LineString 與區塊 Polygon。

[查詢句 2-2] 另一方面，我們使用 **STDistance()**函數來找出路網圖中與物件「?Text」距離在特定範圍 m 公尺內的道路節點，其中路網圖的資料取自資料表 Tyu_node 的 Node 欄位，「?Text」也需轉換成 geometry 型態，需注意的是此函數計算出的距離若要以公尺為單位需乘上 100000，對應的查詢句如下:

```
Select   [Tyu_node].Node
From     [Tyu_node]
Where    ( geometry::STGeomFromText('?Text', 4326).STDistance
         ([Tyu_node].Node) *100000 <= m
```

最後我們舉例說明前置處理的流程，假設此時系統時間為 2017-01-16 的凌晨 2 點 16 分，系統會呼叫『下載模組』，直接從桃園市政府開放資料網站抓取檔名為“GetTodayCase.xml”的資料下載至本機中，其格式與內容如圖 2-2 所示。接著呼叫『資料分析與建立模組』，針對開放資料進行分析。首先我們將所需的資料選取出來，如表 2-1 所示，接著將施工時段切割為 3 個時段，並將施工範圍的格式修正為符合資料類型 Geometry 存取的格式，然後將已存在資料庫內的資料做更新並建立新資料，結果如圖 2-4 所示。需注意的是以幾何資料類型建立至資料庫的施工範圍(Shape)，MSSQL 會自動將其編碼。最後，我們呼叫『資料庫維

護模組』將小於系統時間的資料刪除。例如圖 2-4 實線方框中的 5 筆資料，在執行『資料庫維護模組』時會被刪除。執行上述模組後的資料庫內容如圖 2-5 所示，圖中實線方框的結束時間皆大於系統時間，表示這些道路挖掘的路段並未完成施工。

	CaseID	Date	Morning	Night	Midnight	Shape
1	27906	2017-11-30	9時-17時	null	null	0xE6100000010405000000DF5B317E704A5E40A9102ACEBF...
2	41894	2016-12-31	7時-17時	null	null	0xE61000000104080000002C24A7590C535E4030E32BD150F...
3	49953	2017-01-31	7時-17時	null	null	0xE6100000010406000000A298E8A637545E401D94B488BB0...
4	35711	2016-12-31	7時-17時	19時-24時	null	0xE61000000104200000041C1E5C83D505E402B599880FD...
5	42047	2017-02-14	7時-17時	null	null	0xE6100000010405000000449FEEB544525E40B2650132D70...
6	49919	2016-12-20	9時-16時	null	null	0xE6100000010405000000003616B2D6A535E4000A16A4D56F...
7	46426	2017-10-28	7時-17時	null	null	0xE6100000010412000000F324455078515E408AECAD1C7DF...
8	48498	2016-12-22	9時-16時	null	null	0xE610000001040500000003294384435555E40D7BFCCF73DF...
9	48229	2017-01-05	9時-16時	null	null	0xE6100000010408000003E9E5B91E4545E40176D5635DEF...
10	33689	2017-02-28	7時-17時	19時-21時	null	0xE6100000010407000004022A1A7D7525E40B3E8EC5FD6...

圖 2-4 資料庫更新與建立的結果

	CaseID	Date	Morning	Night	Midnight	Shape
1	27906	2017-11-30	9時-17時	null	null	0xE6100000010405000000DF5B317E704A5E40A9102ACEBFE...
2	49953	2017-01-31	7時-17時	null	null	0xE6100000010406000000A298E8A637545E401D94B488BB0...
3	42047	2017-02-14	7時-17時	null	null	0xE6100000010405000000449FEEB544525E40B2650132D70...
4	46426	2017-10-28	7時-17時	null	null	0xE6100000010412000000F324455078515E408AECAD1C7DF...
5	33689	2017-02-28	7時-17時	19時-21時	null	0xE6100000010407000004022A1A7D7525E40B3E8EC5FD60...

圖 2-5 資料庫維護的結果

第 3 章 路徑規劃系統

在本章中，我們討論如何利用網際網路上開放的路網圖和服務，進行迴避道路挖掘區塊的路徑規劃。本論文延用[洪 16]裡建置的雲端路徑規劃系統，如圖 1-1 所示，也就是整合圖資網站既有的路徑規劃功能，並搭配鄰近點篩選與 GSP 演算法，快速輸出迴避道路挖掘的路徑。不過，不同之處在於本系統使用開放資料，且以關聯式資料庫做為後端的管理平台。

3.1 系統架構

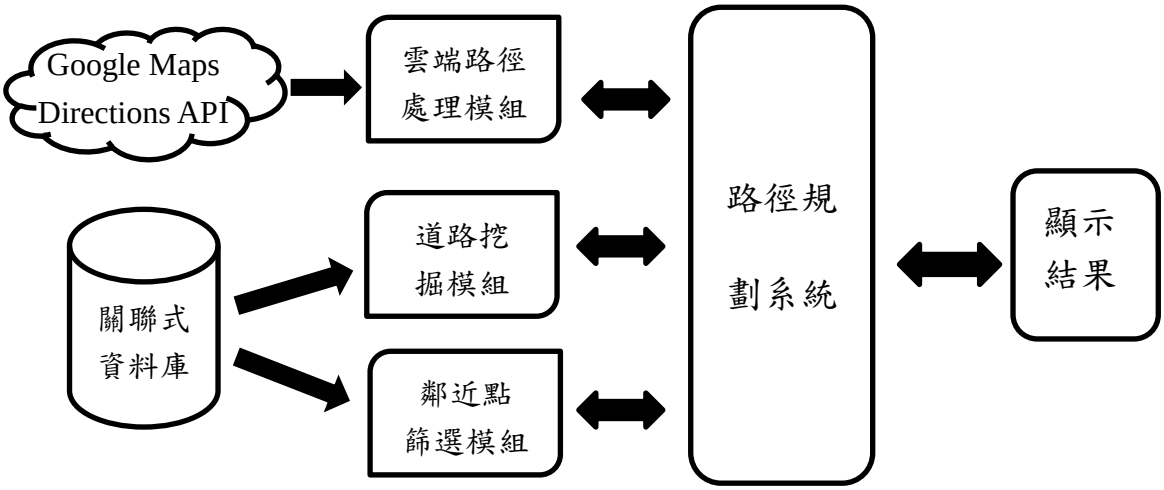

圖 3-1　路徑規劃系統架構圖

由於本方法需要經常與圖資平台做連結和溝通，為了方便往後的管理與維護，我們以模組化的方式建置各功能，整個系統架構如圖 3-1 所示。

主程式如圖 3-2 所示。首先，L01-L02 呼叫『雲端路徑處理模組』，其中 L01 為查詢起始點 V_s 至目的地 V_t 的最短路徑 GPs,t，L02 為合併路徑的演算法，此模組將於 3.2 節詳細介紹。接著，L03-L07 呼叫『道路挖掘查詢模組』，透過 IsClosed 演算法查詢 PathList 中是否存在某條路段與道路挖掘區塊相交，若存在則將經過道路挖掘區塊的路口點儲存於 ClosedRoad 集合。L08 如果 ClosedRoad 為空集合，表示 PathList 路徑不與道路挖掘區塊相交，因此輸出此最短路徑。反之，L10 呼叫『鄰近點篩選模組』，查詢道路挖掘路口點的鄰近點，並回傳分類完的所有群集 NPCategory。L11 如果 NearPointCategory 演算法回傳 0，表示此道路挖掘路口點沒有鄰近點，因此輸出 NoRoute。此模組將於 3.3 節詳細說明。L14 經由 GSP 演算法選出每個種類代表的點當作 waypoint。GSP(Generalized Shortest Path)演算法[RT13]會從每個種類中選出一個代表點，並確定這些代表點會形成從起點至目的地點的最短路徑，詳見[RT13]的說明。L15 將 V_s、V_t 以及 waypoint 經由『雲端路徑處理模組』查詢最短路徑 *WPath*。L16-L21 判斷 *WPath* 是否經過道路挖掘區塊，如果仍經過道路挖掘區塊，就刪除 waypoint，並將修正次數 Step 加 1，然後

重新規畫路徑。L24 輸出最後路徑。

演算法名稱：Main
輸入：*Vs*, *Vt* , *n* , RDB //道路挖掘資料庫
變數：NPCategory //記錄每個不能通行道路的所有鄰近點,
Closed //記錄是否為不能通行的道路
輸出：*PathList*

```
L01   GPs,t←GoogleMapsDirectionsAPI(Vs,Vt)
L02   PathList←Joint_Pro(GPs,t)
L03   For each Route(Vi , Vj) ∈ PathList do
L04     If( IsClosed (Route (Vi , Vj), RDB))
L05       ClosedRoad.insert(Vi , Vj)
L06     End If
L07   End For
L08   If(ClosedRoad.empty())   return PathList
L09   Else
L10     NPCategory←NearPointCategory (ClosedRoad, RDB, Vs,Vt)
L11     If (NPCategory == 0)   return "No Route!!"
L12     Initialize Step ← 0 ; Closed ← 1
L13     while(Closed && Step < n && NPCategory != 0)
L14     WayPointList←GSP(NPCategory)
L15     WPath ←GoogleMapsDirectionsAPI (Vs,Vt,WayPointList)
L16     For each NRoute(Vi , Vj)∈ WPath do
L17       Closed ← IsClosed (NRoute(Vi , Vj),RDB)
L18       If(Closed)
L19         Delete_Waypoint(NPCategory, WayPointList)
L20       End If
L21     End For
L22     Step++//記錄修正次數
L23     End while
L24     return PathList
L25   End If
```

圖 3-2　主程式

3.2 雲端路徑處理模組

近年來許多免費的地圖應用程式介面(API)紛紛開放，例如：Yahoo Maps 、TGOS 及 Google Maps，而我們選用 Google Maps 平台的原因，是因為 Google Maps 提供開發者許多 API 資源。本論文使用 Google Maps Directions API 服務，而 Google Maps Directions API 的呼叫方式如下:
https://maps.googleapis.com/maps/api/directions/json?origin= *Vs* &destination= *Vt* &mode=driving&language=zh-TW，其中 origin 為起點的緯經度座標，destination 為目地點的緯經度座標，mode 為指定計算路線時所要使用的交通模式(本論文使用 driving，開車模式)，language 為使用的語言，回傳格式為 JSON，回傳的所有元素說明參考 Google Maps Directions API 的說明網站[5]，如**附錄 A** 所示。

表 3-1　Google Maps Directions API 所回傳與路徑資料相關的元素

元素階層: Level_1		
父元素	元素名稱	元素說明
	routes	記錄從起點前往目的地的路線資訊，包含 legs、overview_polyline…等子元素。
元素階層: Level_2		
父元素	元素名稱	元素說明
routes	legs	有關於指定路線中的分段相關資訊，每個分段都包含一連串 steps。
	overview_polyline	記錄分段起點至分段終點的路徑經過平滑處理後的所有點，回傳格式經過編碼，需解碼才能取得資訊。
元素階層: Level_3		
父元素	元素名稱	元素說明
legs	end_location	此分段終點的緯度/經度座標，若只有一組分段，則會是路徑的終點。範例:"end_location" : {"lat" : 24.918134,"lng" : 121.1921773}。
	start_location	此分段起點的緯度/經度座標，若只有一組分段，則會是路徑的起點。範例:"start_location" : { "lat" : 24.9158248,"lng" : 121.1846736}。
	steps	步驟陣列會指出有關某行程分段每一步驟的資訊，每個步驟皆包含 distance、duration、end_location、polyline、start_location、html_instructions…等
元素階層: Level_4		

[5] Google Maps Directions API 的說明網站為 https://google-developers.appspot.com/maps/documentation/directions/intro?hl=zh-TW#DirectionsRequests

父元素	元素名稱	元素說明
steps	end_ location	包含此步驟終點的緯度/經度座標。
	polyline	在地圖上創建此步驟的路徑所需的線段集合，回傳格式經過編碼，需解碼才能取得資訊。
	start_ location	包含此步驟起點的緯度/經度座標。

表 3-2　有關路徑點之術語表

中文	自訂名稱	元素名稱	說明	圖示
起點	User_start		使用者指定的起點	
終點	User_end		使用者指定的終點	
近似折線	overview	overview_ polyline	存於 Level_2，記錄形成分段起點至終點路徑所需的點	以空心圓表示 ○。
分段起點	Leg_start	start_ location	存於 Level_3,記錄與 User_start 距離最近的點	
分段終點	Leg_end	end_ location	存於 Level_3,記錄與 User_end 距離最近的點	
步驟起點	Step_start	start_ location	存於 Level_4,記錄每個步驟的起點	以內含數字的實心圓表示
步驟終點	Step_end	end_ location	存於 Level_4,記錄每個步驟的終點	
折線	polyline	polyline	存於 Level_4,記錄形成每個步驟起點至終點路徑所需的點	以圖釘(pin)表示 。

我們從附錄 A 中取出與路徑相關的元素記錄於表 3-1 中。為了方便後面討論，我們統整使用者指定的起點與終點與表 3-1 中元素名稱為 start_location 與 end_location 的部分，並列出其術語表，如表 3-2 所示。為了避免元素階層 Level_3 與元素階層 Level_4 的「start_location」、「end_location」混淆，將其另外命名為分段起點(Leg_start)、分段終點(Leg_end)以及步驟起點(Step_start)、步驟終點(Step_end)。

接著我們針對表 3-2 的各術語做更詳細的介紹，首先介紹「routes」的第一個子元素，也就是 Level_2 的「legs」，其記錄了有關於指定路線中的分段相關資訊，若使用者只指定起點與終點則只會有一個分段，反之若有指定必須經過的點，此 API 會以必經點的個數進行路徑分段，也就是說假設使用者除了指定起點與

終點外，還指定了 1 個必經點，則此 API 會回傳兩個「legs」。而每個分段皆記錄了 Leg_start、Leg_end 以及完成此分段路徑的多個步驟(steps)，後面介紹皆以一組未指定必經點的起迄點，其只會有回傳一個分段路徑資訊(legs)做代表，此時 Leg_start、Leg_end 為此 API 自動選取在其路網資料中距離 User_start 與 User_end 最近的點。而「routes」的另一個子元素「overview_polyline」記錄了起點至終點經過刪除了過近點的平滑處理後的所有點，這裡的起點與終點為 Leg_start 與 Leg_end。

最後「legs」的子元素「steps」包含了開車步驟中特定的單一指示(html_instructions)，例如，「於中興路向右轉」，也包含了此步驟的起點(Step_start)與終點(Step_end)的緯經度以及在地圖上創建此步驟路徑所需的線段集合(polyline)。

[範例 3-1] 利用 Google Maps Directions API 查詢起點(24.9158455 121.1846745)至終點(24.9181332 121.192162)的最短路徑來介紹從接收 JSON 格式中剖析取得的元素階層 Level_4 的「start_location」(Step_start)、「end_location」(Step_end)以及「polyline」和階層 Level_2 的「overview」，如圖 3-3 所示，說明如下:

- 「Step_start」、「Step_end」:由於這兩個元素記錄的是開車步驟中每一步驟的起點與終點，因此其所記錄的點通常會對應到路口點，如圖 3-4 所示。例如，我們假設有一步驟為「於福德街向左轉」，如圖 3-4 中實線圓框部份所示，此時「Step_start」會記錄此步驟與上一步驟交界的路口點，即為圖中編號 3 的位置，而「Step_end」會記錄「福德街」與其下一步驟交界的路口點，即為圖中編號 4 的位置。
- 「polyline」:記錄了形成每一步驟路徑的所有線段的點座標，由於非直線延伸的道路是透過許多線段相互連接，因此回傳的點較多且密集，我們取圖 3-4 中編號 3、4 之間的路段來呈現，如圖 3-5 中的圖釘(pin)表示，可發現其中包含較多且密集的點。
- 「overview」:包含了在地圖上繪製 Leg_start 至 Leg_end 的所有步驟路徑所需的線段點集合(polyline 所記錄的點)再經過平滑處理刪除過近的點後所形成的路徑，由於回傳的點較多因此取圖 3-4 中編號 3、4 之間的路段來呈現，如圖 3-6 中的空心圓所示。注意到 polyline 和 overview_polyline 所記錄的資訊已經編碼過，所以取出之後必須透過解碼才能獲得所需要的緯經度資訊。此外，透過觀察我們發現「polyline」及「overview_polyline」所記錄的點皆包含了元素「Step_start」、「Step_end」所記錄的點。另外，Level_3 的 Leg_start 和 Leg_end 已被包含在上述三個元素中。

```
{   …
    "steps" : [
                {   …
                    "end_location" : {
                       "lat" : 24.9162561,
                       "lng" : 121.1855429
                    },
                    ...
                    "polyline" : {
                        "points" : "{jawCe{sbVISk@wAa@aA"
                    },
                    "start_location" : {
                        "lat" : 24.9158248,
                        "lng" : 121.1846736
                    },
                    ...
                },
                …
            ],
     ...
    "overview_polyline" : {
        "points" :
"{jawCe{sbVu@kBa@aAjAg@jAo@QkACIGEOAqAIGCUe@SYQmCEU
y@cBM]Mu@GYKY_@qA]uB{@{IEWUg@MQu@e@"
    },
   ...
}
```


圖 3-3　Google Maps Directions API 回傳的 JSON 格式

圖 3-4　Step_start 與 Step_end 所記錄的點

圖 3-5　polyline 所記錄的點

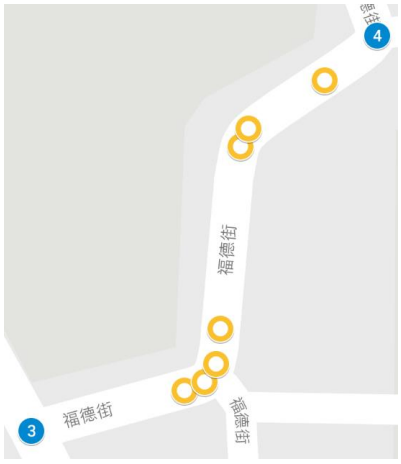

圖 3-6　overview 所記錄的點

接著我們針對元素「overview」、「Step_start」、「Step_end」以及「polyline」來討論我們要使用哪種元素來解決判斷與道路挖掘區塊交集的問題。我們首先考慮只使用最具代表性的元素「Step_start」與「Step_end」所記錄的點，而這兩個元素所回傳的點主要為路口點，因此當道路挖掘區塊在兩個回傳的點之間且道路並非直線延伸時，會產生判斷是否與道路挖掘交集時的誤判。以[範例 3-1]來說，元素「Step_start」與「Step_end」所記錄的點，如圖 3-4 中編號 1 到編號 6 的點所示。注意到，圖中編號 5 與編號 6 連接的直線(自行假設並繪製的虛線部分)並未與道路挖掘區塊交集(自行假設的不規則多邊形區塊，如圖 3-7 中的實線框所示)，此時會產生判斷是否與道路挖掘交集時的誤判，示意圖如圖 3-7 所示。

圖 3-7　Step_start 與 Step_end 誤判範例

接著我們考慮 overview 與 polyline 所記錄的點，這兩個元素所記錄的點皆可解決判斷是否與道路挖掘交集時的誤判問題，且透過觀察圖 3-5、圖 3-6，我們發現 polyline 所記錄的點包含了 overview 所記錄的點。在圖 3-8 中我們根據密集點的部分，分為三個實線框圓進行比較點的個數與兩點間的平均距離，結果參照表 3-3。由於我們判斷交集的方法是以兩點形成的線段與道路挖掘區塊之間做交集判斷，而點個數多且平均距離較短表示會因兩點間距離過近而做過多相似且不必要的交集判斷，因此我們選擇點個數較少且兩點平均距離較長的 overview 而不選 polyline。此外，我們將範例 3-1 中所有 overview 記錄的點中較為密集的部分，分為 6 個實線框圓，如圖 3-9 所示，並比較圓中兩點間的平均距離，結果參照表 3-4，根據結果我們假設在距離門檻值小於 15m 時，表示距離過近可以合併。在後面的實驗會以不同的門檻值進行比較。

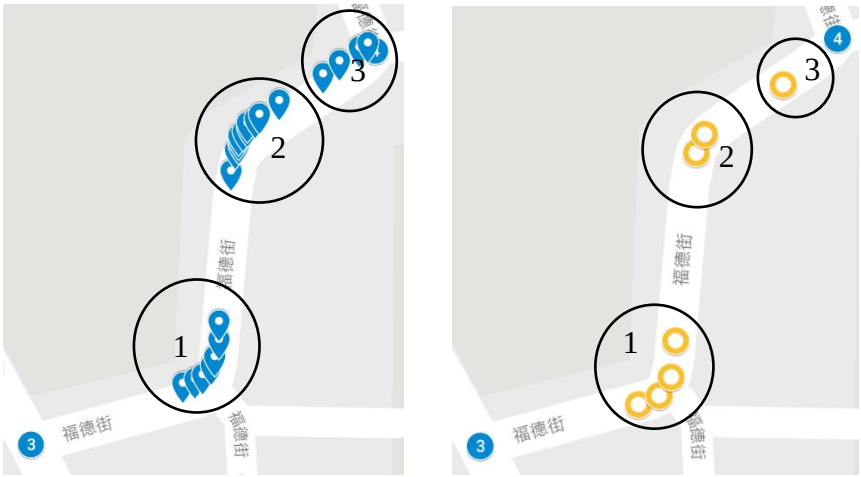

圖 3-8　polyline(左)與 overview (右)比較範例

表 3-3　overview 與 polyline 的三個實線框圓比較

名稱	點個數		兩點平均距離(m)	
	overview	polyline	overview	polyline
實線框圓 1	4	7	6.6215	3.3335
實線框圓 2	2	11	4.8891	2.3366
實線框圓 3	1	4	-	4.5328

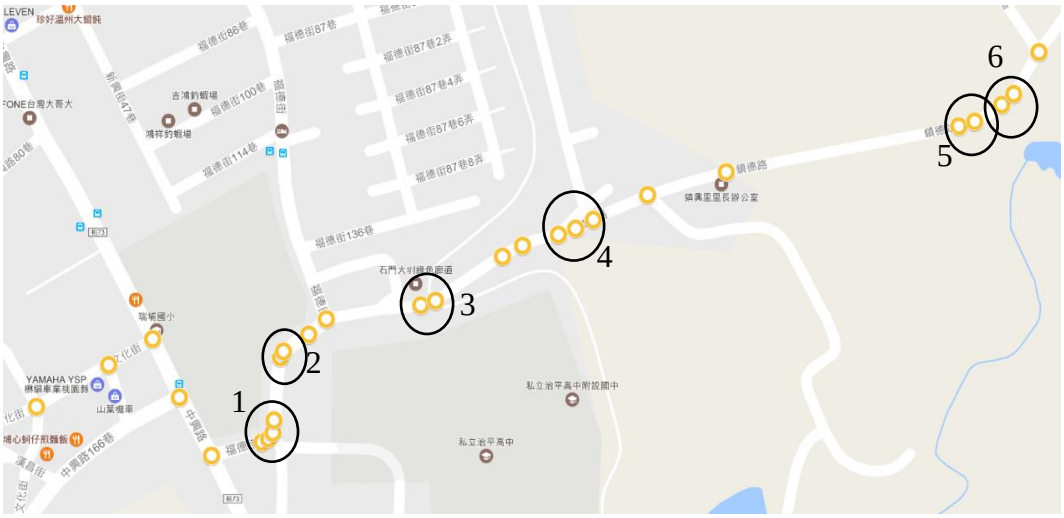

圖 3-9　legs_start 到 legs_end 的 overview 回傳點示意圖

表 3-4　overview 中的 6 個實線框圓比較

名稱	點個數	平均距離	名稱	點個數	平均距離
實線框圓 1	4	6.6215m	實線框圓 4	3	14.2924m
實線框圓 2	2	4.8891m	實線框圓 5	2	12.5664m
實線框圓 3	2	11.5964m	實線框圓 6	2	11.9696m

根據以上討論，overview 不僅可以幫助我們解決判斷與道路挖掘區塊交集的誤判問題，在某些較為彎曲的道路上，其所記錄的點在彎曲處已經比 polyline 少，但如圖 3-8 的 1 號實線框圓所示，仍然存在相鄰兩點距離相當近且在判斷交集上不會有影響的現象。因此我們針對 overview 所回傳的點序列整理出以下性質，希望能將點的數量進一步精減以提升效率，並確保判斷與道路挖掘區塊交集的誤判問題仍能被解決。性質如下，示意圖左邊的點為原來的點，右邊的點為希望保留的點:

[性質 1]若點序列中，某一點(V_i)屬於 Leg_start 或 Leg_end，我們會將 V_i 保留。

[性質 2] 若點序列中，某一點(V_i)屬於 Step_start 或 Step_end 所記錄的點，我們會將 V_i 保留。

提出此兩個性質，是因為我們的道路挖掘區塊通常是從一條道路的路口點開始延伸挖掘，若刪除掉道路的路口點，可能會造成判斷與道路挖掘區塊交集的誤判，因此我們希望在刪除距離過近的點時，能避免刪除到道路的路口點。

[性質 3]當相鄰的兩點(V_i ,V_{i+1})距離大於門檻值，我們會保留 V_i 與 V_{i+1}，如圖 3-10 所示。

圖 3-10　性質 3 示意圖

[性質 4]當一連串相鄰點(V_i ,V_{i+k},…,V_j)，其點之間的距離皆小於門檻值，我們會保留最前面與最後面的點 V_i 與 V_j。

依據此性質我們刪除中間的點 V_{i+1} 到 V_{j-1}，至於保留 V_j 是為了避免 V_i 與 V_{j+1} 的距離過長，而造成交集誤判，如圖 3-11 所示。但由於我們在處理此性質之前及之後會處理[性質 3]，因此在後面的演算法中，不會有性質 4 的處理。

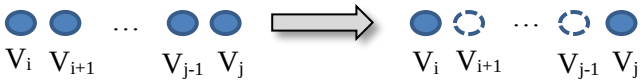

圖 3-11　性質 4 示意圖

我們的合併演算法的想法是首先建立一個布林陣列 Path 對應到 overview 中所有的點，並利用 True 對應到根據上述四個性質決定要保留的點。此外在演算法中我們使用 GPoint 陣列紀錄 overview 中所有點的緯經度，GStep 陣列記錄 Step_start 與 Step_end 中所有點的緯經度，值得注意的是 GPoint 陣列與 GStep 陣列皆包含了 Legs_start 與 Legs_end，而順序則皆為 Legs_start 往 Legs_end 的方向排列，且 GPoint 陣列包含了 GStep 陣列的點。

完整的 Join_Pro 演算法如圖 3-12 所示，其中 L01 為設定初始值，我們將 Path 布林陣列初始皆設定為 False，表示不保留。L02 由於 GPoint[0]為 Legs_start 屬於[性質 1]，因此將 Path[0]指定為 True 表示要保留的點。L04-07 比較 GPoint[i+1]是否存在於 GStep 陣列中，n 的起始為 1，若 GPoint[i+1]屬於 GStep[n]表示為[性質 2]，因此將 Path[i+1]指定為 True。我們一次只比較一個 GStep[n]，若 GPoint[i+1]屬於 GStep[n]，GPoint[i+1]後面將不會再有與 GStep[n]相同的點，於是將 n 加 1。L08-L11 為處理[性質 3]的方式，當相鄰兩點距離大於門檻值 t 時，我們保留 V_i 與 V_{i+1}，因此將 Path[i]與 Path[i+1]皆指定為 True。由於我們已將所有點的初始指定為 False，因此在演算法中[性質 4]刪除 V_{i+k} 的部分，我們維持 False 不做任何處理。L14 此時 GPoint[i]為 Legs_end 屬於[性質 1]，因此將 Path[i]指定為 True。L15-17 若 Path[m]為 True 表示 GPoint[m]為要保留的點因此將這個點加入到 PathList。L18 回傳最終路徑 PathList。

演算法名稱: join_pro
輸入: GPoint //overview_polyline 的點序列, t //合併距離值,
GStep//Step 的起迄點序列
輸出: PathList //合併出來的路徑集合

```
L01   Initialize Path[GPoint.size]←False, PathList←NULL, i←0,n←1,m←0
L02   Path[0]←True
L03   while(i <GPoint.size() - 1)
L04     If(GPoint[i+1]==GStep[n])
L05       Path[i+1]←True
L06       n←n+ 1
L07     End If
L08     If(Dist(GPoint[i],GPoint[i+1]) > t)
L09         Path[i]←True
L10         Path[i+1]←True
L11     End If
L12     i←i + 1
L13   End while
L14   Path[i]←True
L15   While(m < Path.size()-1)
L16     If(Paht[m]==True)
L17       PathList.Add(GPoint[m])
L18   Return PathList
```

圖 3-12　Join_Pro 演算法

[範例 3-2] 使用[範例 3-1]的起點與終點說明我們的演算法，首先取得 overview 所記錄的點，門檻值 t 為 15m，並將 Boolean 陣列 Path 的初始值設為 False 對應到每個回傳點，結果如表 3-5 所示，而 Step_start 與 Step_end 所記錄的點，如表 3-6 所示。首先執行 L02 將 V1 指定為 True。接著執行 L04-L06 若 V_{i+1} 屬於 GStep 中的點，則將其指定為 True。然後執行 L08-L10 計算兩點距離，若兩點距離大於 15m 則將 V_i 與 V_{i+1} 指定為 True，反之則維持 false。演算法流程示意圖如表 3-7 所示，由於回傳點較多，我們取表 3-5 中 V5 到 V9 做為執行範例，會保留[V5,V6,V9,V10]。最後執行一樣的判斷流程至最後一點並將保留的結果點輸出，最終結果如表 3-5 中以淺灰網底表示的點，如圖 3-13 所示。

表 3-5　GPoint 中的點

順序	點	Distance (V_i,V_{i+1})	順序	點	Distance (V_i,V_{i+1})
1	24.91582 121.18467	62.2526	15	24.91651 121.18766	59.9178
2	24.91609 121.18521	38.3150	16	24.9168 121.18816	17.0305
3	24.91626 121.18554	46.8723	17	24.91687 121.18831	28.3499
4	24.91588 121.18574	48.7483	18	24.91694 121.18858	13.8589
5	24.9155 121.18598	39.6501	19	24.91698 121.18871	14.7259
6	24.91559 121.18636	5.5169	20	24.91704 121.18884	45.0610
7	24.91561 121.18641	5.3851	21	24.9172 121.18925	61.8598
8	24.91565 121.18644	8.9624	22	24.91735 121.18984	178.8082
9	24.91573 121.18645	45.9183	23	24.91765 121.19158	12.5664
10	24.91614 121.1865	4.8890	24	24.91768 121.1917	23.6138
11	24.91618 121.18652	22.7568	25	24.91779 121.1919	11.9696
12	24.91629 121.18671	17.2093	26	24.91786 121.19199	35.6548
13	24.91639 121.18684	72.3755	27	24.91813 121.19218	
14	24.91648 121.18755	11.5964			

表 3-6　GStep 中的點

順序	點	順序	點
1	24.91582 121.18467	4	24.91639 121.18684
2	24.91626 121.18554	5	24.91694 121.18858
3	24.9155 121.18598	6	24.91813 121.19218

表 3-7　演算法流程示意圖

順序 (i)	V_i	V_{i+1}	Distance (m)	Path 的變化
5	24.9155 121.18598	24.91559 121.18636	39.6501	V5: false → true (L09) V6: false → true (L10)
6	24.91559 121.18636	24.91561 121.18641	5.5169	V6: true V7: false
7	24.91561 121.18641	24.91565 121.18644	5.3851	V7: false V8: false
8	24.91565 121.18644	24.91573 121.18645	8.9624	V8: false V9: false
9	24.91573 121.18645	24.91614 121.1865	45.9183	V9: false → true (L09) V10: false → true (L10)

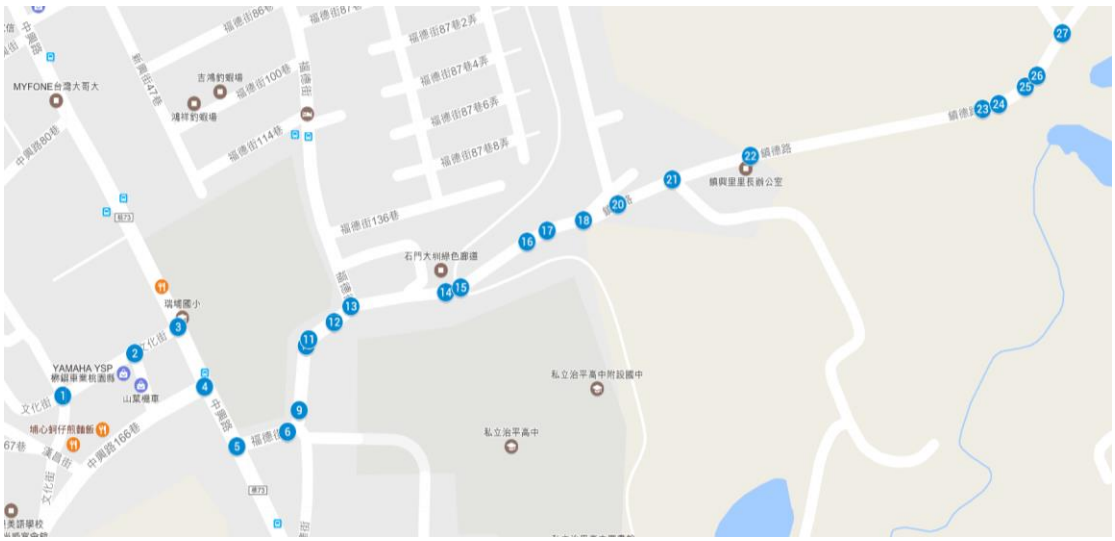

圖 3-13　合併的結果

3.3 道路挖掘查詢與鄰近點篩選模組

由於先前 join_pro 演算法所取得的 PathList 尚未判斷是否有碰到道路挖掘區塊的路段，因此主程式在 L04 呼叫 IsClosed 演算法來判斷 PathList 是否與道路挖掘區塊相交，而完整的 IsClosed 演算法如圖 3-14 所示。其中 L02 使用 2-3 節中的[查詢句 2-1]來判斷某一條路段是否與道路挖掘相交，此時[查詢句 2-1]中的欲查詢物件「?Text」為 PathList。L03-L04 若查詢句回傳的所相交道路挖掘區塊數量大於 0，則回傳 True，反之則回傳 False。

演算法名稱: IsClosed 輸入: PathList 輸出: True 或 False //True 表示有與道路挖掘相交
L01　Initialize IntersectsList←NULL L02　IntersectsList ← [查詢句 2-1] L03　If (IntersectsList.Size() > 0) return True; L04　Else Return False; L05　End If

圖 3-14　IsClosed 演算法

接著，當道路挖掘模組回傳為 True 時，表示所規劃的路徑與道路挖掘區塊相交，此時就必需尋找其他未碰到道路挖掘區塊的鄰近點，因此圖 3-2 的主程式在 L10 呼叫 NearPointCategory 演算法執行鄰近點篩選並將鄰近點分群，以便執行 GSP 演算法找出最佳的替代路徑。注意到，資料庫內已含有我們事先已建立好的桃園市路網圖資料，也就是 2-3 節介紹的表格 Tyu_node (道路節點)。本模組主要是向資料庫輸入選點的 SQL 查詢句(查詢句 2-2)進行資料篩選，但為了避免選出的鄰近節點也在道路挖掘區域內，因此會再使用判斷交集的 SQL 查詢句(查

詢句 2-1)確認每個鄰近點是否避開道路挖掘區塊，若鄰近點在道路挖掘區域內，則不輸出此點。

完整的 NearPointCategory 演算法如圖 3-15 所示。此演算法使用到與道路挖掘區塊相交的路段(ClosedRoad)、RDB 中的表格 Tyu_node 和 TY_roadclose、起點(V_s)以及終點(V_t)。L01 將 V_s 設定為第一群點集合，即 $C0=\{V_s\}$。L04 為 N 個碰到道路挖掘區塊的路口點找出其他鄰近點，其中 L05 是為了判斷是否有其他鄰近點，如果沒有則不建立群集，並在 L10-11 回傳 0。L12 將 V_t 設定為最後一群點集合，即 $C_{N+1}=\{Vt\}$，L13 回傳的 CategoryList 為最終分群完的鄰近點集合，記錄每個欲查詢點的所有鄰近點。

演算法名稱: NearPointCategory 輸入: ClosedRoad、RDB、V_s、V_t 變數: C//記錄每一群的鄰近點, V_i //表示與道路挖掘相交的點 輸出: NPCategory//點的群集，記錄每個群集的點
L01 C [0].insert(V_s) L02 Initialize i←1 //i 表示某個種類 L03 While(i <= ClosedRoad.Count()) L04 NPointList←NearPoint (V_i , RDB) L05 If (NPointList==NULL) break; L06 Else L07 C[i].insert(NPointList) L08 i←i + 1 L09 End While L10 If(i != ClosedRoad.Count()) L11 Return 0; L12 C[i].insert(V_t) L13 Return C

圖 3-15　NearPointCategory 演算法

接著，我們說明 L04 的 NearPoint 查詢句，完整的查詢句如圖 3-16 所示，此查詢句我們同時使用到 2-3 節介紹的兩個查詢句。L01-03 我們對應 2-3 節的[查詢句 2-2]找出 V_i 的鄰近節點，根據給定的查詢範圍 m 公尺內的所有點，此時欲查詢物件[?Text]為欲查詢的點 V_i，我們將所回傳的點取名為 NPoint。L04-L06 我們對應[查詢句 2-1]找出與 V_i 距離 m 公尺內的道路挖掘區塊，這是為了減少鄰近點是否碰到道路挖掘區塊時的判斷，此時查詢物件[?Text]為道路挖掘區塊，我們將所回傳的道路挖掘區塊取名為 NShape。L07 我們將先前 L01-06 兩個查詢句所回傳的 NPoint 與 NShape 以 **STIntersects()**函數判斷交集，回傳沒有與 NShape 相交的 NPoint。

SQL 查詢句: NearPoint 輸入: V_i,RDB 輸出: NPoint		
L01	**Select**	[RDB].[Tyu_node].Node **AS** NPoint
L02	**From**	[RDB]. [Tyu_node]
L03	**Where**	geometry::STGeomFromText('?Text', 4326).**STDistance** ([Tyu_node].Node) *100000 <= m
L04	**Select**	[RDB].[TY_roadclose].Shape **AS** NShape
L05	**From**	[RDB].[TY_roadclose]
L06	**Where**	geometry::STGeomFromText('?Text', 4326).**STDistance** ([TY_roadclose].Shape) *100000 <= m
L07	**Select**	(NPoint)**.STIntersects** (NShape) = 0

圖 3-16　NearPoint 查詢句

[範例 3-3]　延續範例 3-2 的結果，結果如表 3-8 所示。接著呼叫『道路挖掘查詢模組』，利用[查詢句 2-1]來判斷表 3-8 的路徑是否碰到道路挖掘區塊。舉例來說，順序 3 的路徑經過道路挖掘區塊，因此輸出其路口點，如表 3-9 所示。取得封閉路口點後，呼叫『鄰近點篩選模組』並將查詢鄰近的範圍設定為 100m 尋找其鄰近點，如表 3-10 及表 3-11 所示，然後將起點、目的地及封閉道路路口鄰近分為 4 群，*C0*={*Vs*}、*C*1={ *NP1*、...、*NP9*}、*C*2={ *NP10*、...、*NP19*}、*C*3={*Vt*}，利用 GSP 演算法選擇每一個群集走訪的點。GSP 演算法的作法如表 3-12 所示，我們將起始點 *Vs* 經過每個 V_i 內的所有項目，再到目的地 *Vt* 之最短距離，走訪的距離儲存於 X 矩陣中，直至 X 矩陣計算完成後，選擇每個 V_i 內距離最短的項目做為該 V_i 的代表點。最後，我們分別從 V_1、V_2 之鄰近節點集合裡選擇 NP3 和 NP19 當作 waypoint，將此四點<*Vs, NP3, NP19,Vt*>，送交 google 路徑查詢模組規畫路徑。圖 3-17 為未經過執行路徑規劃系統前的路徑，V_1 與 V_2 分別是圖中的 B 與 C 點。而圖 3-18 顯示重新規劃的路徑，NP3 和 NP19 都對應到圖中的 B 點，此路徑不經過道路挖掘區塊。

表 3-8　『雲端路徑查詢模組』回傳的路徑

順序	路徑		順序	路徑	
1	24.91582 121.18467	24.91609 121.18521	13	24.91651 121.18766	24.9168 121.18816
2	24.91609 121.18521	24.91626 121.18554	14	24.9168 121.18816	24.91687 121.18831
3	24.91626 121.18554	24.91588 121.18574	15	24.91687 121.18831	24.91694 121.18858

4	24.91588 121.18574	24.9155 121.18598	16	24.91694 121.18858	24.91704 121.18884
5	24.9155 121.18598	24.91559 121.18636	17	24.91704 121.18884	24.9172 121.18925
6	24.91559 121.18636	24.91573 121.18645	18	24.9172 121.18925	24.91735 121.18984
7	24.91573 121.18645	24.91614 121.1865	19	24.91735 121.18984	24.91765 121.19158
8	24.91614 121.1865	24.91618 121.18652	20	24.91765 121.19158	24.91768 121.1917
9	24.91618 121.18652	24.91629 121.18671	21	24.91768 121.1917	24.91779 121.1919
10	24.91629 121.18671	24.91639 121.18684	22	24.91779 121.1919	24.91786 121.19199
11	24.91639 121.18684	24.91648 121.18755	23	24.91786 121.19199	24.91813 121.19218
12	24.91648 121.18755	24.91651 121.18766			

表 3-9　封閉的路口點

	點
V_1	24.91626 121.18554
V_2	24.9155 121.18598

表 3-10　V_1 的鄰近點

	點		點
NP1	24.916542 121.18536	*NP6*	24.915456 121.186004
NP2	24.915844 121.184692	*NP7*	24.915475 121.186022
NP3	24.917095 121.185762	*NP8*	24.916278 121.185533
NP4	24.917224 121.185687	*NP9*	24.91553 121.185962
NP5	24.91578 121.185814		

表 3-11　V_2 的鄰近點

	點		點
NP10	24.91578 121.18514	*NP15*	24.915475 121.186022
NP11	24.915119 121.18556	*NP16*	24.915559 121.186484
NP12	24.915456 121.186004	*NP17*	24.916278 121.185533
NP13	24.914833 121.186356	*NP18*	24.91553 121.185962
NP14	24.915726 121.186491	*NP19*	24.917095 121.185762

表 3-12　GSP 演算法之運算過程

Matrix	Step	VisitID
X[0,1]	-	-
X[1,1]	Distance (*Vs*, *NP1*)	*Vs*
X[1,2]	Distance (*Vs*, *NP2*)	
X[1,3]	Distance (*Vs*, *NP3*)	
X[1,4]	Distance (*Vs*, *NP4)*	
X[1,5]	Distance (*Vs*, *NP5)*	
X[1,6]	Distance (*Vs*, *NP6)*	
X[1,7]	Distance (*Vs*, *NP7)*	
X[1,8]	Distance (*Vs*, *NP8)*	
X[1,9]	Distance (*Vs*, *NP9)*	
X[2,1]	X[1,1]+Distance (*NP1*, *NP10*) X[1,2]+Distance (*NP2*, *NP10*) X[1,3]+Distance (*NP3*, *NP10*) X[1,4]+Distance (*NP4*, *NP10*) X[1,5]+Distance (*NP5*, *NP10*) X[1,6]+Distance (*NP6*, *NP10*) X[1,7]+Distance (*NP7*, *NP10*) X[1,7]+Distance (*NP8*, *NP10*) X[1,9]+Distance (*NP9*, *NP10*)	*NP3*
X[2,2]	X[1,1]+Distance (*NP1*, *NP11*) … … X[1,9]+Distance (*NP9*, *NP11*)	
…	… …	

X[2,10]	X[1,1]+Distance (*NP1*, *NP19*) … … X[1,9]+Distance (*NP9*, *NP19*)	
X[3,1]	X[2,1]+Distance (*NP10*, *Vt*) … … *X*[2,10]+ Distance (*NP19*, *Vt*)	*NP19*

圖 3-17　原來的範例路徑

圖 3-18　迴避挖掘區塊的路徑

第 4 章 實驗

在本章，我們進行實驗來探討資料分析與建立模組的效率、兩個路段合併的方法的準確率與效率。針對鄰近點篩選模組，我們比較刪除與道路挖掘相交的鄰近點的兩個方法的效率，此外，我們還比較了兩種不同資料類型的效率。最後，我們探討路徑規劃系統內各模組的效率。

接下來說明我們進行實驗的環境，我們以個人電腦作為實驗的環境，其 CPU 為 i7-4770 四核心且核心時脈為 3.4GHz，而記憶體為 8GB，所採用的作業系統為 64 位元的 Windows 7 企業版。

4.1 系統實作方法與輸出範例

本論文所有的程式皆以 Microsoft Visual C# 2015 實作，資料分析與建立模組所建構的資料庫是使用 SQL server 2014，雲端路徑處理模組是利用 C# API 呼叫路徑查詢服務，而顯示路徑部分，則使用本機端架設 IIS Web Server & ASP.NET。

我們會將回傳的路徑結果輸出於網頁中，以起點(24.9376306,121.2518072)、終點(24.9390387,121.2593867)做為範例顯示本論文的輸出結果。如圖 4-1 為初次 Google Maps 規劃的最短路徑(尚未判斷道路挖掘區塊)。圖中框起來的部份表示一筆道路挖掘區域，而點 A 代表起點，點 D 代表終點。圖 4-2 則為透過 WayPoint，即為 B 點與 C 點，其可修改圖 4-1 中 B 點到 C 點的路段，進而順利規畫出迴避道路挖掘區塊的路徑。後續我們將對主要模組做進一步的實驗與比較。

圖 4-1　Google Maps 所回傳的最短路徑

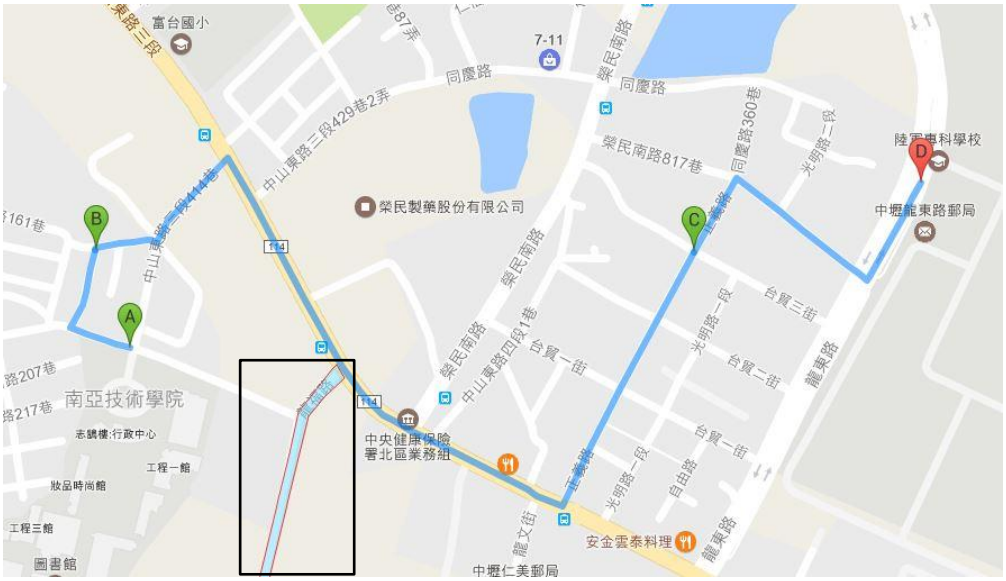

圖 4-2　修正後迴避道路挖掘區塊的路線

4.2 資料集

本實驗所使用的資料集包括交通部 102 年出版的路網圖，及桃園市政府開放資料平台所提供的道路挖掘資訊。

首先，桃園市的路網圖資料集共 70231 個節點、86768 條邊。而道路挖掘區塊來自於 105 年 12 月到 106 年 1 月中不同日期所下載的真實資料如表 4-1，其每筆資料為不規則的多邊形區域。而後面的實驗我們將道路挖掘區塊固定為 475 個，合併距離值預設為 50 公尺，鄰近距離值預設為 300 公尺，測試資料為隨機選取 20 組有碰到道路挖掘區塊的起迄點，表 4-2 所示。

表 4-1　不同日期的道路挖掘資料

日期	12/13	12/17	12/18	12/21	01/15	01/16
真實資料數	100	233	180	268	172	210

表 4-2　20 組桃園市起迄點之測試資料

	Q1	Q2	Q3	Q4	Q5
Vs	25.006722 121.089785	24.9376306 121.2518072	24.934112 121.252335	25.0140724 121.2926409	24.957713 121.226398
Vt	24.9998671 121.0943558	24.9390387 121.2593867	24.934545 121.254464	25.016187 121.292306	24.955845 121.227642
距離	1.004km	1.083km	0.275km	0.3km	0.35km
	Q6	Q7	Q8	Q9	Q10

Vs	25.006736 121.089801	24.985873 121.137241	24.9158455 121.1846745	25.033812 121.058229	25.014612 121.057995
Vt	25.000980 121.087112	24.99004 121.1436817	24.9181332 121.192162	25.024668 121.061557	25.028681 121.067358
距離	0.824km	1.040km	0.932km	2.648km	2.550km
	Q11	Q12	Q13	Q14	Q15
Vs	24.941488 121.058202	25.025793 121.057772	25.014272 121.057731	25.021305 121.05763	25.021305 121.05763
Vt	24.956302 121.0667	25.039285 121.069626	25.028251 121.06346	25.024757 121.071978	25.03005 121.063532
距離	2.999 km	3.670 km	2.441 km	1.956 km	1.443 km
	Q16	Q17	Q18	Q19	Q20
Vs	25.025518 121.05814	25.024278 121.058125	25.021453 121.057981	24.948323 121.057664	25.026838 121.057598
Vt	25.02325 121.075104	25.029539 121.073816	25.031632 121.072678	24.959619 121.072155	25.031625 121.069193
距離	2.476 km	2.252 km	1.885 km	2.607 km	2.209 km

4.3 資料分析與建立模組之效率實驗

本實驗利用表 4-1 的道路挖掘資訊，組合成“100”、“200”、“300”、“400”筆的等差資料量，而實驗方式為執行資料分析、建立新資料至資料庫及更新資料庫各十次，然後取這十次平均。觀察資料量對資料分析與建立模組的影響，由圖 4-3 可見，資料分析與建立的時間與資料量大小成正相關，且 CPU 時間皆很小，表示主要時間花費在 I/O。

表 4-3 資料量對資料分析與建立模組之影響

執行時間(s)	資料量			
	100	200	300	400
I/O Time	0.0589	0.0925	0.1594	0.2272
CPU Time	0.0078	0.0124	0.0156	0.0234
Total Time	0.0667	0.1050	0.1750	0.2506

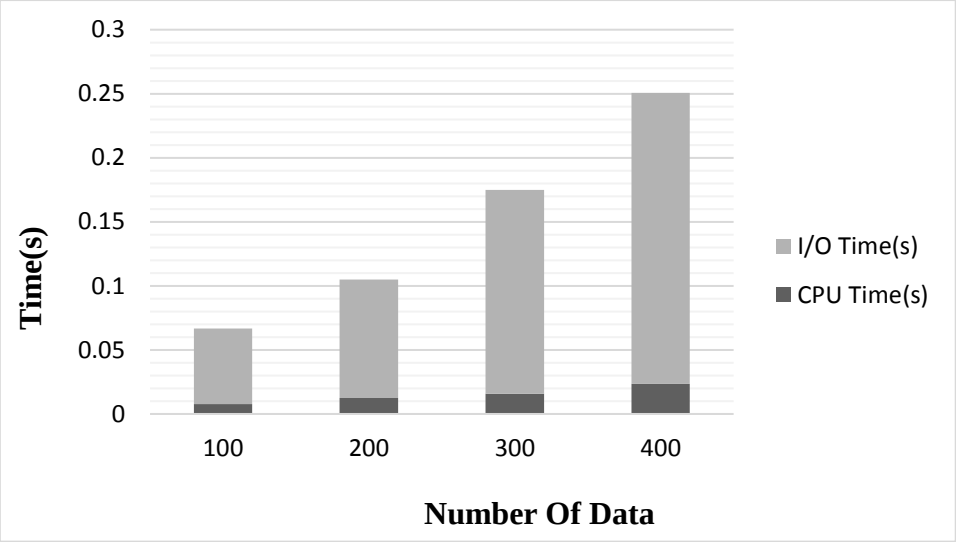

圖 4-3 資料量對資料分析與建立模組之影響

以下實驗是比較兩種建立與維護方法的效率，一種是先將資料庫內的舊資料更新並新增新資料，最後刪除小於系統時間的資料，也就是如第二章所描述的流程，下圖中以 update 表示。另一種是將資料庫內的舊資料全部刪除，再新增所有新資料，下圖中以 delete_all 表示。由結果可知整體時間皆隨資料量增加，但 delete_all 的效率皆比 update 較佳，資料量越大兩個方法的效率差距越大，所以未來將會改以使用“delete_all”的作法。

表 4-4　兩種建立與維護方法的效率記錄表

方法(ms)	資料量			
	100	200	300	400
update	36.6559	73.6384	120.508	179.3253
delete_all	34.6244	61.7972	88.95	127.5548

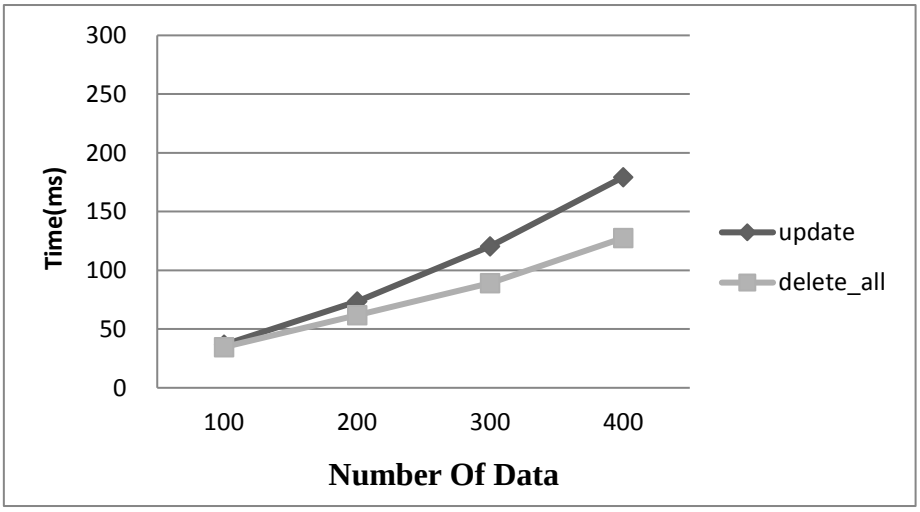

圖 4-4　兩種建立與維護方法之效率比較

4.4 合併路徑方法之準確率與效率實驗

這次實驗我們採用真實資料的 475 個道路挖掘區域，接著，選取 20 組碰到道路挖掘區域的起迄，如表 4-2，並將合併距離值以“50m”、“100m”、“150m”的等差變化，探討不同合併距離值之影響。我們不續用 3.2 節所討論的 15m 是因為此時兩個合併方法準確率皆很高，看不出兩個方法的差異，相關數據如表 4-5 所示，表中的第三欄記錄了兩個方法最終留下的點個數。

本實驗比較[洪 16]的合併方法(Join_Point)與本論文第 3.2 節所提出的合併方法(Join_Pro)之迴避道路挖掘區域路徑的準確率以及整體時間。[洪 16]所用的合併方法 Join_Point 是當兩點距離小於合併距離值時，將左邊點刪除，持續比對至最後一點，但這個方法並未考慮到全部的路徑情況，當遇到連續多個小於合併距離值的路段時，則會因刪除掉重要的轉折點，降低道路挖掘查詢模組的準確率。如圖 4-5、圖 4-6 所示，將圖 4-5 中的點 12-16 刪除，只剩下點 17 導致圖 4-6 中的點 11 與點 17 路段並未與道路挖掘區塊相交(斜線方塊)，而造成誤差。而本論文的合併方法會增加步驟點的判斷(如性質 1 和 2)，以增加道路挖掘查詢模組的準確率。

準確率的比較結果如圖 4-7 所示，我們可以發現當合併距離值增加，Join_Point 合併方法的準確率隨線性減少，但 Join_Pro 合併方法的準確率皆未改變。這是因為 Join_Point 合併方法未考慮到全部路徑情況，因此多刪除了重要的路口點，且當合併距離越大，越容易刪除道路口點，進而使準確率降低。而 Join_Pro 合併方法多增加了步驟點的判斷，可以減少路口點被刪除，進而使準確率達到 90%以上。

而效率的比較結果如圖 4-8 所示，我們可以發現當增加合併距離值，Join_Pro 的效率皆比 Join_Point 差，這是因為 Join_Pro 比 Join_Point 多了判斷是否為步驟點的時間，因此效率上比 Join_Point 差，此外兩個合併方法的效率皆隨著合併距離增加而微幅提升，這是因為當合併距離值越大，會保留較少的點，進而使效率提升。

表 4-5　合併距離 15m 之紀錄表

	準確率	時間(ms)	點個數
Join_Point	0.9	0.3934	17.28
Join_Pro	0.9	0.5089	20.21

圖 4-5　未合併之路徑

圖 4-6　Join_Point 之合併結果

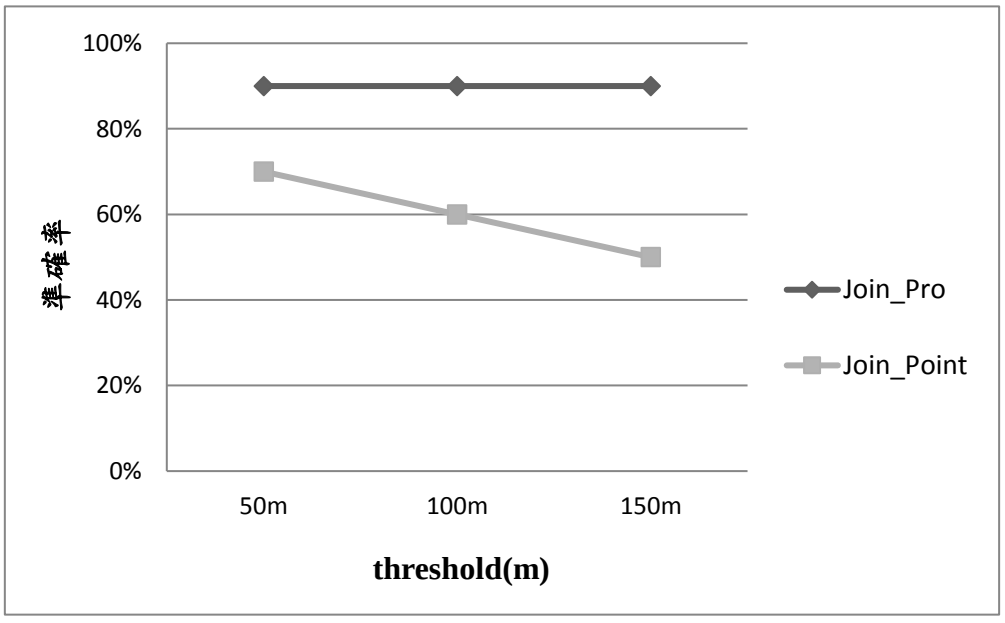

圖 4-7　合併方法之準確率比較

表 4-6　合併方法之效率紀錄表

執行時間(ms)	合併距離值(m)		
	50	100	150
Join_Point	0.34334	0.29664	0.2946
Join_Pro	0.45202	0.44073	0.38705

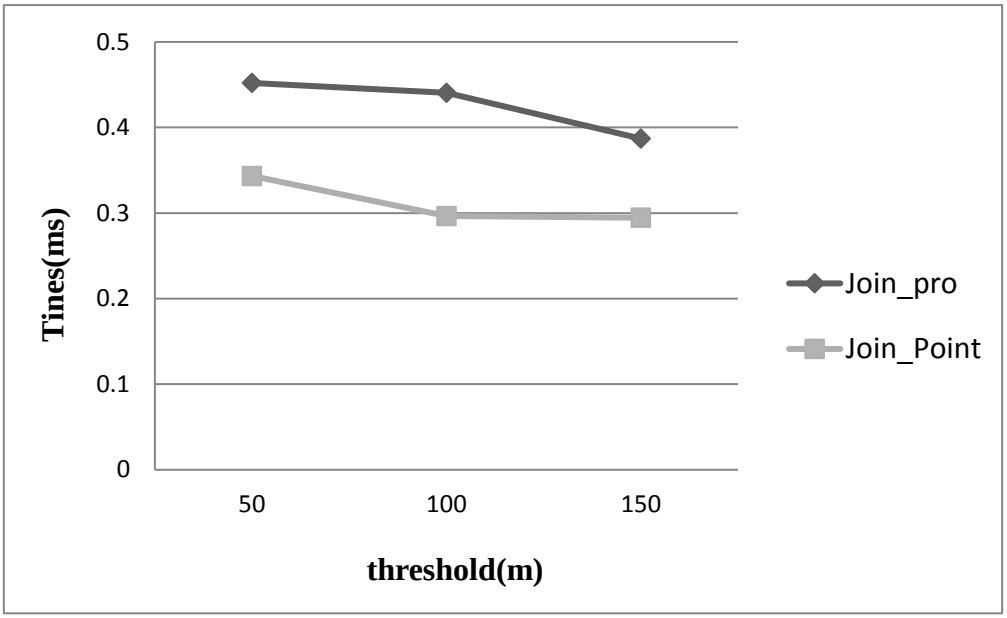

圖 4-8　合併方法之效率比較

4.5 鄰近點模組之效率實驗

這次實驗我們隨機選取 20 個點，鄰近距離值以“400m”、“500m”、“600m”的等差變化，記錄這 20 個點執行鄰近點模組的時間，然後取平均值，探討以下實驗:

(1) 探討鄰近點模組之刪除與道路挖掘區域相交的鄰近點的兩個方法的比較，一種為查詢全部挖掘區域的 all_shape，另一種為第 3.3 節所提到的將挖掘區域縮小至特定範圍的 range_shape (包含查詢特定範圍的 shape 的時間)。結果如圖 4-9，由實驗結果可知 range_shape 的效率要比 all_shape 好，因為 all_shape 查詢了過多不必要的道路挖掘區域。

表 4-7 鄰近點模組之刪除資料時間記錄表

鄰近距離	鄰近點個數	執行時間(s)	
		all_shape	range_shape
400m	101	1.8439	0.4894
500m	153	2.6265	1.0730
600m	216	3.4116	1.8973

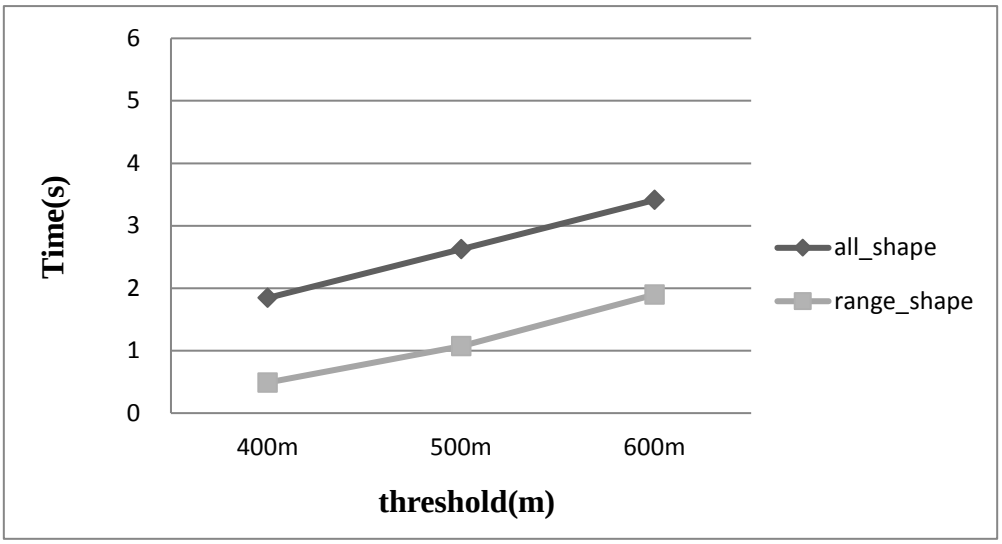

圖 4-9 鄰近點模組之刪除資料時間

(2) 這次實驗我們比較資料庫中兩個不同的資料類型的效率，這兩個資料類型分別是字串(nvarchar)與幾何(geometry)，結果如圖 4-10 所示，在不同鄰近點距離下，資料類型為幾何的執行效率皆優於字串，這也是為什麼在 2.2 節時我們選擇將資料以幾何的資料類型存取的原因，而使用幾何的效率會優於字串是因為我們使用資料庫所提供的 Distance()函數，是以地理空間資料格式來計算欲查詢的點與鄰近點的距離，因此我們先將資料以地理空間資料類型中的 geometry 來記錄至資料庫中，可以提升效率，而若以字串資料類型來存取則會因為需多做資料類型轉換而使效率降低。

表 4-8 鄰近點模組之資料類型效率紀錄表

鄰近距離	鄰近點個數	執行時間(s)	
		字串(nvarchar)	幾何(geometry)
400m	101	2.0267	0.4894
500m	153	2.4962	1.0730
600m	216	3.4465	1.8973

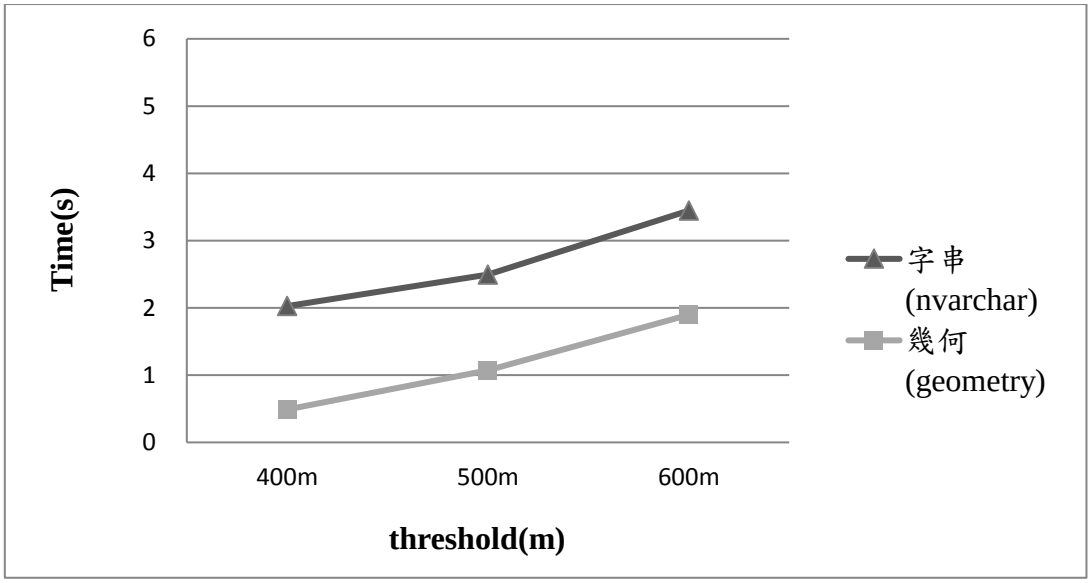

圖 4-10　鄰近點模組之資料類型效率

4.6 主要模組的時間討論

這次實驗我們隨機選取 20 組起迄點如表 4-2 所示，鄰近距離值以“200m”、“300m”、“400m”的等差變化，並將修正次數設定為 15 次，記錄每組起迄點皆執行 10 次後主要模組的平均值，觀察路徑規劃系統中主要模組的效率。其中，IsClosed 標示道路挖掘查詢模組、NearPoint 標示鄰近點篩選模組。

結果如圖 4-11 所示，隨著鄰近距離值增加，整體時間明顯增加，主要是鄰近點篩選模組與 GSP 演算法的時間會隨著鄰近距離值增加而微幅增加，這是因為當鄰近值增加，鄰近點會增加，因此會增加 GSP 演算法的計算。而道路挖掘查詢模組因為與距離值無關，所以時間無明顯變化，但耗時皆為 1 秒以上，且佔總時間的 50%，由此可知道路挖掘模組仍有改善的空間。

表 4-9 鄰近距離值之各模組效率紀錄表

模組時間(s)	鄰近距離值(m)		
	200	300	400
道路挖掘查詢 (IsClosed)	1.12063	1.48043	2.3923
鄰近點篩選 (NearPoint)	0.90957	1.02559	1.0857
GSP	0.00599	0.00628	0.0143
Total	2.14352	2.76462	3.81
鄰近點個數	17.6	42.7	73.2
修正次數	6.15	7.32	10

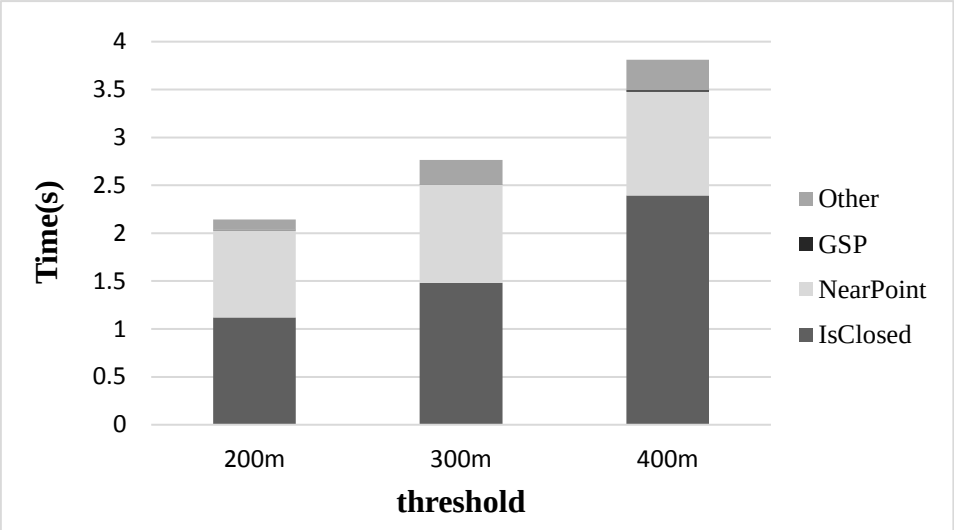

圖 4-11 鄰近距離值之各模組效率

接下來的實驗我們改變道路挖掘數量來探討各模組效率，首先隨機選取 20 組起迄點如表 4-2 所示，道路挖掘數量以“475”、“400”、“300”的等差變化，並將修正次數設定為 15 次，記錄每組起迄點皆執行 10 次後主要模組的平均值，觀察路徑規劃系統中主要模組的效率。

結果如圖 4-12 所示，隨著道路挖掘數量增加，整體時間明顯增加，主要是因為道路挖掘查詢模組隨著道路挖掘數量增加而遞增，且約佔總時間的 50%，因此得知該模組仍有改善的空間。

表 4-10　道路挖掘數量之各模組效率紀錄表

模組時間(s)	道路挖掘個數		
	300	400	475
道路挖掘查詢 (IsClosed)	0.75206	0.912	1.48043
鄰近點篩選 (NearPoint)	0.99782	0.99349	1.02559
GSP	0.00575	0.00574	0.00628
Total	1.9019	2.0544	2.7646

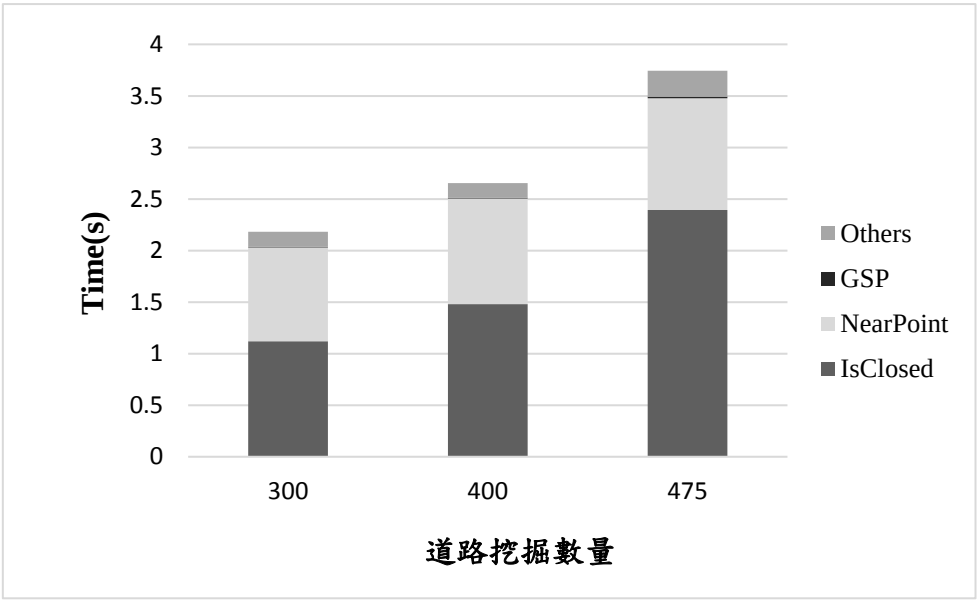

圖 4-12　道路挖掘數量之各模組效率

第 5 章 結論與未來方向

本論文將政府提供的道路挖掘開放資料結合先前研究的路徑規劃系統[洪16]，規劃出迴避道路挖掘區塊的最短路徑，並提出一個增加了步驟點的判斷的合併路徑演算法 Join_Pro，來改善之前[洪 16]研究提出的 Join_Point 演算法因刪除過多點而造成的準確率問題。我們進行一系列實驗，結果如下所示:

- 4.3 節的實驗顯示我們針對建立資料分析與建立模組的方法，另外提出的 delete_all 方法，比 2.2 節中提出的 update 方法來的更有效率，且資料量越大兩個方法的效率差距越大。
- 4.4 節的實驗顯示本論文提出的 Join_Pro 方法要比之前研究的 Join_Point 方法更能準確的判斷出是否避開道路挖掘，但所花費的時間較 Join_Point 方法來的慢。
- 4.5 節的實驗顯示在鄰近點模組之刪除與道路挖掘區塊相交的鄰近點時，將挖掘區域縮小至特定範圍的這個方法能有效地提升效率。此外，在兩個不同資料類型的比較中顯示地理空間資料類型比字串資料類型更有效率。
- 4.6 節的實驗顯示在所提的系統架構中，以道路挖掘查詢模組的時間佔總時間的 50%為最多。

綜合以上討論，本論文未來的改進方向有以下幾點:

1. 修正系統中的資料分析與建立模組，改用 delete_all 的方法。
2. 針對所提出的 Join_Pro 演算法，希望能在保有高準確性的情況下，更進一步的提升效率。
3. 針對道路挖掘查詢模組，探討是否能利用索引或其他技術，來提升判斷是否與道路挖掘區塊相交的效率。
4. 希望能夠整合不同縣市的開放資料於路徑規劃系統中。

參考文獻

[CJZG16] Zhiguang Cao, Siwei Jiang, Jie Zhang, Hongliang Guo, “A Unified Framework for Vehicle Rerouting and Traffic Light Control to Reduce Traffic Congestion”, IEEE Transactions on Intelligent Transportation Systems, Volume PP Issue 99, Pages 1-16, 2016.

[MH16] Cherlton Millette, Patrick Hosein, “A Consumer Focused Open Data Platform”, Proceedings of the International Conference on Big Data and Smart City (ICBDSC), 2016.

[PPB17] Juan Pan, Iulian Sandu Popa, Cristian Borcea, “DIVERT: A Distributed Vehicular Traffic Re-Routing System for Congestion Avoidance”, IEEE Transactions on Mobile Computing, Volume 16 Issue 1, Pages 58-72, 2017.

[RT13] Michael N. Rice, Vassilis J. Tsotras, “Engineering generalized shortest path queries”, Proceedings of the ICDE conference, 2013.

[SYMLV16] Allan M. de Souza, Roberto S. Yokoyama, Guilherme Maia, Antonio Loureiro, Leandro Villas, “Real-time path planning to prevent traffic jam through an intelligent transportation system”, Proceedings of the IEEE Symposium on Computers and Communication (ISCC), 2016.

[WDZ16] Shen Wang, Soufiene Djahel, Zonghua Zhang, “Next Road Rerouting: A Multiagent System for Mitigating Unexpected Urban Traffic Congestion”, IEEE Transactions on Intelligent Transportation Systems, Volume 17 Issue 10, Pages 2888-2899, 2016.

[ZY16] Chenxiao Zhang, Peng Yue, “Spatial grid based Open Government Data mining”, IEEE Symposium on Geoscience and Remote Sensing (ISGRS) symposium, 2016.

[洪 16] 洪蔚齊, “基於索引技術之淹水區域路徑規劃”, 國立臺灣海洋大學資訊工程研究所碩士論文, 2016。

[陳 15] 陳書齊, “基於開放架構之互操作性示警資訊建立及應用架構”, 國立成功大學測量及空間資訊學院碩士論文, 2015。

[蘇 15] 蘇郁婷, “時空地理資訊整合展示介面之探討”, 國立成功大學測量及空間資訊學院碩士論文, 2015。

附錄 A　Google Maps Directions API 回傳的元素

元素階層: Level_1		
父元素	元素名稱	元素說明
Directions Response	geocoded_ waypoints	陣列會有起點、目的地及途經地點的地理編碼詳細資料，包含 geocoder_status、place_id、types 子元素。
	routes	記錄從起點前往目的地的路線資訊，包含 legs、overview_polyline…等子元素。
	status	包含與要求相關的中繼資料，“OK”表示回應包含有效的 result。
元素階層: Level_2		
父元素	元素名稱	元素說明
geocoded_ waypoints	geocoder_ status	指出從地理編碼操作產生的狀態碼，"OK" 指出未發生任何錯誤，已順利剖析地址並且已至少傳回一個地理編碼。
	place_id	是可與其他 Google API 搭配使用的唯一識別碼。
	types	用於計算路線之地理編碼結果的 address type，address type 傳回的類型：street_address、route…等。
routes	bounds	包含此路線的檢視區邊框。
	copyrights	為此條路線顯示的著作權文字。
	legs	有關於指定路線中兩個位置之間的路線分段相關資訊，每個分段都包含一連串 steps。
	overview_ polyline	記錄起點至終點的路徑經過平滑處理後的所有點，回傳格式經過編碼，需解碼才能取得資訊
	summary	包含路線的簡短文字描述。
	warnings	警告陣列會在顯示這些路線時顯示，必須自行處理並顯示此資訊。
	waypoint_ order	指出計算路線中任何途經地點的順序。
元素階層: Level_3		
父元素	元素名稱	元素說明
legs	distance	表示此分段的總距離，包含 value:0.9 公里、text:932(單位為公尺)
	duration	表示此分段的總時間，包含 value: 3 分、text: 180 (單位為秒)

	end_ address	為此分段的 end_location 進行反向地理編碼所產生人類看得懂的地址 (通常為街道地址)，範例: "end_address" : "324 台灣桃園市平鎮區鎮德路"。
	end_ location	此分段指定目的地的緯度/經度座標，範例: "end_location" : {"lat" : 24.918134, "lng" : 121.1921773}。
	start_ address	由此分段 start_location 的反向地理編碼所產生人類看得懂的地址(通常為街道地址)範例: "start_address" : "326 台灣桃園市楊梅區文化街 35 號"。
	start_ location	此分段起點的緯度/經度座標。範例:"start_location " : { "lat" : 24.9158248,"lng" : 121.1846736}。
	steps	步驟陣列會指出有關某行程分段每一步驟的資訊，每個步驟皆包含 distance、duration、end_location、polyline、start_location、html_instructions…等
元素階層: Level_4		
父元素	元素名稱	元素說明
steps	distance	包含此步驟到下一步驟為止所涵蓋的距離。
	duration	包含到下一步驟為止，執行步驟所需的時間長度。
	end_ location	包含此步驟的終點位置
	html_instr uctions	此步驟的格式化指示 (以 HTML 文字字串呈現) 範例: "html_instructions" : "於\u003cb\u003e 中興路 \u003c/b\u003e/\u003cb\u003e 桃 73 鄉道\u003c/b\ u003e 向\u003cb\u003e 右\u003c/b\u003e 轉"。
	maneuver	此步驟的操縱指示，範例: "maneuver" : "turn-right"，表示在此步驟需右轉。
	polyline	在地圖上創建此步驟的路徑所需的線段集合，回傳格式經過編碼，需解碼才能取得資訊。
	start_ location	包含此步驟的起點位置
	travel_ mode	表示旅行模式，旅行模式有開車、步行、大眾運輸…等