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

迴避淹水區域之快速路徑規劃研究

The Research on Efficient Route Planning for Avoiding Flooded Regions

研究生：李承翰 撰

中華民國 104 年 2 月

迴避淹水區域之快速路徑規劃研究

The Research on Efficient Route Planning for Avoiding Flooded Regions

研究生：李承翰　　Student：Cheng-Han Li

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
February 2015
Keelung, Taiwan, Republic of China

中華民國 104 年 2 月

摘要

台灣常年因颱風與豪雨事件所帶來的降雨，釀成嚴重的淹水災害，因此在淹水發生前的預警措施成為重要的研究議題。在之前的研究裡，我們已經建置一個洪氾預警系統，可以快速辨認出有淹水可能的地標。而本文則進一步探討到地標救災之路徑規劃的問題，也就是藉由路徑規劃技術與淹水區域的資訊，快速規劃出由起點至目的地且迴避淹水區域的路線。我們提出了兩個方法，首先，Baseline 方法利用 R-tree 讓淹水區域與道路的最小邊界矩形作空間連接，進而找出淹水的道路，於剔除所有淹水的道路後利用 Dijkstra 找出最短路徑。接著，我們提出 Cloud 方法，先將起點與終點送至 Google Maps 取得其規劃的最短路徑，再找出其中經過淹水區域的道路路口，另尋找其替代的路口節點，接著，利用 GSP(Generalized Shortest Path)的演算法計算起點經過替代節點再到終點的最短路徑，然後將所得的路口點送交 Google Mapse 規劃與呈現最後的路徑。

我們實作 Baseline 和 Cloud 兩個方法，並進行一系列的實驗，比較兩個方法的效率、路徑長度及成功規劃出可行走之路徑的比率。實驗結果顯示，Baseline 的成功率比 Cloud 高 15%、輸出路徑長度較 Cloud 短 12%，但在效率方面隨著路網資料量提升至萬筆後，Baseline 在計算空間連接及路徑規劃的耗時會大幅增加，反之 Cloud 方法則有穩定的效率表現。

Abstract

The disaster brought by heavy rain has become more and more serious in Taiwan, and it has been an important research issue to provide warning messages before flood. In the past research, we have built a flood forecasting system. It can identify the landmark which might be inundated. In this thesis, we further investigate the problem of routing planning for landmark rescuing. That is, we wish to quickly determine a route which starts from a given departure point, such as a landmark, avoids flooded areas, and gets to the destination.

We propose two methods. The first one is called the "Baseline" method. It uses the R-tree index to perform spatial join between flooded areas and the minimum bounding rectangles of roads to identify the flooded roads. The Dijkstra algorithm then operates on those remaining unflooded roads to find the shortest path avoiding flooded areas.The second one is called the "Cloud" method. We first submit the departure point and the destination point to the Google Maps routing planning service to get an initial shortest path. We then identify those flooded roads within the path, and find nearby alternative intersections. Then, the departure point, alternative intersections, and the destination point are operated by the GSP (Generalized Shortest Path) algorithm to find an improved shortest path, and finally, the end points of each road within this path will be submitted to Google Maps again to get the final route.

We have implemented these two methods and performed a series of experiments to compare their efficiency, route lengths, and the ratio of successfully getting a route without passing through flooded areas. The results show that the Baseline method has higher success rates, shorter route paths. However, when the roadnetwork is larger, its efficiency will decrease a lot. In contrast, the Cloud method performs quite fast even on a large dataset.

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，從老師的身上學到了抽絲剝繭的實驗精神，與精確嚴謹的做事態度，使學生能順利的完成論文。

除此之外，感謝許為元博士和臺北科技大學劉傳銘博士百忙之中抽空參與論文審查工作，提供寶貴的意見，使論文更趨近完善。

最後，感謝佩珊學姐及祐愷、昱德、餘恩學長，帶領著我們適應實驗室環境與經驗的傳承。感謝我的同學韋錫與學弟妹們，陪伴著我一同在研究所生涯歷練與成長。更要感謝“ENL.TW 今生”的朋友，陪我征戰百岳及紓解壓力！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。ChuMi

目錄

圖目錄

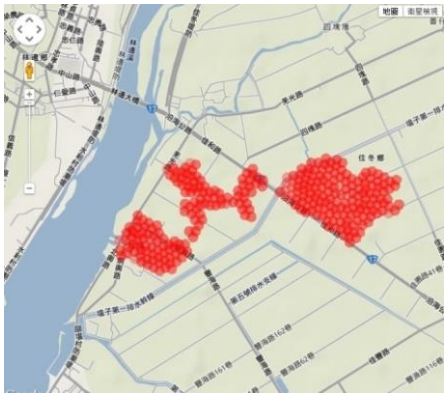
表目錄

第 1 章　序論技術背景

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

台灣地區降雨豐沛，年平均降雨量高達 2515 公釐，但分布極不均勻的降雨量，常造成大雨宣洩不及釀成災害。之前莫拉克颱風造成的八八水災，就是一個實例。在之前的研究中[吳 12]已經建置一個洪氾預警系統，可以預測未來 1-3 小時後可能淹水的區域，如圖 1-1(a)之深色圓圈所示。[劉 13]更進一步將淹水區域根據地標統整，可以快速辨認出有淹水可能的地標，地標如圖 1-1(b) 地圖上之 Marker 所示。



(a) 淹水區域畫面

(b) 地標警示畫面

圖 1-1　洪氾預警系統

本論文延伸此二論文的研究成果，探討到地標救災之路徑規劃的問題。以目前消防隊救災而言，救災路線仍然是人工識別淹水區域，再藉由個人經驗規劃出救災路線，一但當救災地點增加的情況下規劃所需的時間就相對增加。我們希望藉由路徑規劃技術與淹水區域的資訊，快速規劃出一條由起點(如即將淹水的地標)至目的地(如避難場所)且迴避淹水區域的路線。

1.2 研究方法與貢獻

此研究的主要議題，在於如何「快速」地輸出避開淹水區域的路徑，便於提升救難單位規劃路線的效率。我們首先提出基於 Dijkstra 演算法[EW59]的 Baseline 方法。該方法先找出經過淹水區域的道路，將其從路網圖中移除，然後將其餘可行走之道路經由 Dijkstra 演算法找出最短路徑，Dijkstra 演算法是常用來進行路徑規劃的演算法，該演算法利用動態規劃的技術，保證找到最短路徑，但當路網圖的資料量過大時，會導致執行時間延長。

我們另外提出的 Cloud 方法，則主要利用 Google Maps 圖資平台提供的路徑規劃雲端服務。我們首先向 Google Maps 查詢起點至目的地的最短路徑，然後，將這條路徑與淹水區域做相交計算，將所有經過淹水區域的路口點查詢替代節點，也就是鄰近但沒有淹水的路口。其次，將起點、替代節點和目地點透過 GSP-DF 演算法[RT13]進行最短路徑的規劃，再將所回傳的所有路口點交由 Google Maps 進行最後路徑之規劃與呈現。

我們完成此二方法，並將所得的路徑皆呈現於 Google Maps 上。我們也針對此兩方法進行廣泛的研究，比較其各自的優劣點。

1.3 相關研究

我們首先討論有關路徑規劃的相關議題。最基本的類型是針對單一起點的最短路徑問題（Single Source Shortest Path）。最基本的是 Dijkstra 演算法[EW59]，而加快其計算速度的 A* 演算法[HNR68]也經常被使用，以下討論與其相關的研究。

[張 09]是利用改良式 A*進行較佳路徑導引之研究，將路段時速加入 A*演算法啟發式評估公式中，所規劃出路徑適用於實際路況的環境。[彭 11]則利用 A* 演算法進行全域避障路徑之規劃，以解決機器人在作業環境中行走的路徑問題。論文中並利用 Diagonal、Euclidean、Dijkstra 和 Manhattan 等不同的評估函式，運用在不同的障礙物分佈環境上進行實驗驗證。[劉 12]則以 A* 演算法為基礎架構，針對目前已知的不同做法在執行時的時間、路徑以及不同未知節點展開數量的多寡進行分析，最後設計不同的資料型態，使得搜索速度更進一步提升。[吳 13] 針對救護車或消防車，提出「最短行車時間路徑規劃法」，也就是會即時依據路況訊息以 A*最短路徑演算法重新規劃路線，以避開塞車路段。首先救護車(簡

稱 E-car)登入系統，Server 會將每條道路的速限值當作初始 Weight，E-car 每 20 秒會向 Server 詢問是否有最新最短時間路徑，如果有就更新路徑。至於路況方面，則是由一般車輛(簡稱 G-car)每 20 秒將本身行車資訊（時間差和距離差）回傳給 Server，若當前路段為塞車時會通知 Server，以便 Server 即時重新規劃路徑，同時也會詢問附近是否有 E-car 以避開。

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。

其次，論文[ZLTZ13]研究如何在 Roadnetwork 上利用索引支援 kNN search，也就是給定一個 Query Location 以及在 Roadnetwork 上一組候選物件(點)的集合，找出距離 Query Location 前 k 名鄰近物件(點)。

最後，本論文會用到 R-Tree[AG84]來做 Spatial join 查詢，R-tree 是一種階層式資料結構(Hierarchical Data Structure)，被用來儲存搜尋空間中的矩形物件，可以便於進行空間區塊的重疊等查詢。

1.4 論文架構

本論文其餘各章節的架構如下：第二章敘述本論文的背景定義與相關做法，藉以對本論文的研究有基礎的認識。第三章介紹 Baseline 方法，說明其整體架構及如何利用常見的 Dijkstra 演算法找出迴避淹水區域的最短路徑。第四章介紹 Cloud 方法，說明其整體架構及如何利用 Google Maps 所提供的路徑規劃服務快速找出迴避淹水區域的路徑。於第五章中以實驗比較 Baseline 與 Cloud 作法的效率、長度及成功率，並於第六章提出本論文的結論與未來方向。

第 2 章　背景定義與相關做法

我們在此章說明基本的定義，包括路網圖型資料的表示法、淹水區域及輸出，以及本論文欲解決的問題。我們也會介紹文獻中關於路徑規劃重要的相關作法。

2.1　背景定義

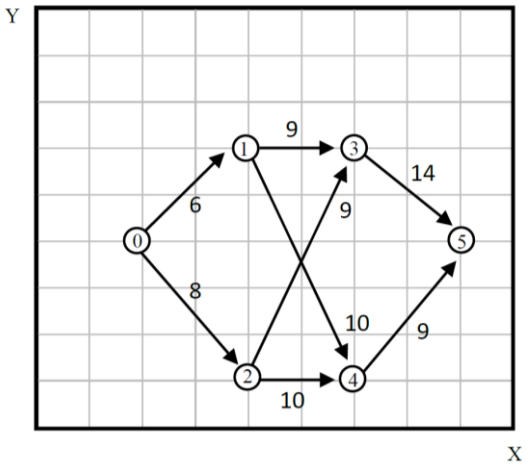


圖 2-1　路網圖

首先，我們介紹本論文中所使用的路網圖(Roadnetwork)。我們假設 $G(V, E, W)$為一個有方向性的連通圖(connected graph)，其中每個節點 $v \in V$ 代表一個道路的路口點，為了可以方便辨識圖上的節點，我們會對圖中的節點做編號。每個邊 $e \in E$ 表示成(V_i, V_j)代表一條實際的道路，箭頭代表道路的通行方向。現實道路的地下道或高架道路，在本文中的圖型表示會以交叉的線段呈現。每個權重 $w \in W$ 則代表兩個路口點的矩離。舉例來說，在圖 2-1 中每個圓圈內的編號便為該點的編號，而 V_1 到 V_3 的距離為 9 單位，在本論文中我們會以最小的編號作為起始點 V_s、最大的編號作為目的點 V_t。

接著，我們說明與路徑相關的基本定義與問題。首先，定義通用的最短距離與最短路徑如下：

[定義 2-1]　最短距離、最短路徑：給定兩個節點 V_i 與 V_j，若 $D_{i,j}$ 的值是 V_i 所有可走訪到 V_j 的路徑中，長度為最短，則稱為 V_i 到 V_j 的最短距離，而對應 $D_{i,j}$ 的路徑則稱為 V_i 到 V_j 的最短路徑以 $P_{i,j}$ 表示。

而最常見的最短路徑問題如下：

[問題 2-1]　Single Source Shortest Path (SSP) Problem[EW59]：給定一個 $G(V, E, W)$、起點 V_s 和目的點 V_t，查詢 V_s 至 V_t 的最短路徑。

舉例來說，在圖 2-1 中我們可觀察到若想求得 $P_{0,5}$，我們可先列出所有路徑如表 2-1 所示。由所有產生的距離值當中，我們可得知 $D_{0,5}$ 為最小的值 25，而對應的路徑即為最短路徑，也就是 $P_{0,5} = V_0 \rightarrow V_1 \rightarrow V_4 \rightarrow V_5$。

表 2-1　SSP 範例走訪表

組合	路徑	距離
1	$V_0 \rightarrow V_1 \rightarrow V_3 \rightarrow V_5$	29
2	$V_0 \rightarrow V_1 \rightarrow V_4 \rightarrow V_5$	25
3	$V_0 \rightarrow V_2 \rightarrow V_3 \rightarrow V_5$	31
4	$V_0 \rightarrow V_2 \rightarrow V_4 \rightarrow V_5$	27

但是，在現實生活中，我們有時需要進行更複雜的路徑規劃。舉例來說，我們可能需要規劃一條由家裡到餐廳的最短路徑，但中途需先經過加油站及郵局再到餐廳，這個問題我們可以如下正式表示：

[問題 2-2] Generalized Shortest Path (GSP) Problem[RT13]：給定一個 $G(V, E, W)$ 和種類序列 $C=\{C_1、....、C_k\}$，C 裡面的每個種類 C_i 皆不重複，起始種類為 C_1、終止種類為 C_k，另外 $C_i=\{C_{i,1}, C_{i,2},...., C_{i,|Ci|}\} \subseteq V$，若另外給定起始節點 V_s 與終止節點 V_t，此問題是依序從每一個種類 C_i 選取一點節點走訪，且找出所有可能路徑中具有最短長度的路徑。

假設如圖 2-1　路網圖所示，將節點編入 4 個種類(Category) $C_0=\{V_0\}$、$C_1=\{V_1,V_2,V_3\}$、$C_2=\{V_4\}$、$C_3=\{V_5\}$，每個 C_i 必須挑選其中一個 $C_{i,j}$ 當作走訪的節點，且走訪的順序必須依序是 C_1、C_2，且路徑也要是最短距離。如表 2-2 我們列出所有路徑，從中看出 $D_{0,5}$ 為最小的值 25，其對應的路徑即為最短路徑 $P_{0,5} = C_{0,1} \rightarrow C_{1,1} \rightarrow C_{2,1} \rightarrow C_{3,1}$，而對應節點後即 $V_0 \rightarrow V_1 \rightarrow V_4 \rightarrow V_5$。

表 2-2　GSP 範例走訪表

組合	路徑	距離
1	$C_{0,1}(V_0) \rightarrow C_{1,1}(V_1) \rightarrow C_{2,1}(V_4) \rightarrow C_{3,1}(V_5)$	25
2	$C_{0,1}(V_0) \rightarrow C_{1,2}(V_2) \rightarrow C_{2,1}(V_4) \rightarrow C_{3,1}(V_5)$	27

最後我們提出本論文要解決的問題。水災時常需要規劃一條由救難隊到災區的搶救路線，但這條路的規劃必須盡可能迴避淹水區域，而我們也希望規劃出來的是最短路徑，以便迅速抵達救難現場，此問題的正式定義如下：

[問題 2-3] UnFlooded Shortest Path (USP) Problem：給定一個 $G(V, E, W)$ 及淹水區域 $F=\{F_1、F_2、...、F_k\}$，針對起點 V_s 至目的點 V_t，查詢 V_s 至 V_t 的最短路徑，但此路徑不得有任何部分與 F_i 相交。

圖 2-2 為一個含有淹水區域的路網圖，其中淹水區域以灰色的矩形方塊表示，我們要從中查詢避開淹水區域 F_1 後 V_0 至 V_5 的最短路徑。如表 2-3 我們可列出所有避開淹水區域的路徑，其中 $D_{0,5}$ 為最小的值 27，而對應的路徑即為最短路徑 $P_{0,5} = V_0 \rightarrow V_2 \rightarrow V_4 \rightarrow V_5$。

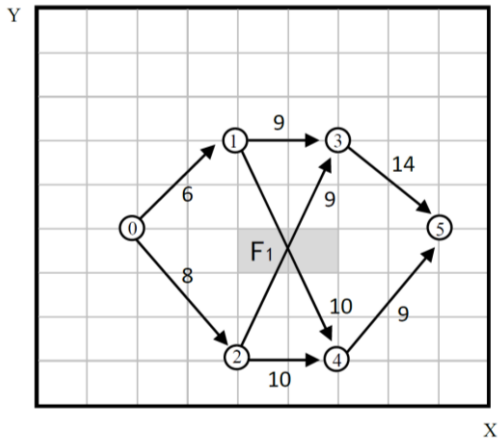


圖 2-2　含有淹水區域的路網圖

表 2-3　USP 範例走訪表

組合	路徑	距離
1	$V_0 \rightarrow V_1 \rightarrow V_3 \rightarrow V_5$	29
2	$V_0 \rightarrow V_2 \rightarrow V_4 \rightarrow V_5$	27

在本論文的第三章和第四章，我們會提出針對 USP Problem 的兩個作法，而在第五章，會針對效率、路徑長度及成功得到路徑的比率進行比較。在以下小節中，則先列出了文獻中針對**[問題 2-1]**和**[問題 2-2]**的作法。

2.2　Dijkstra 演算法介紹

在本節中我們介紹解決 SSP 最常使用到的 Dijkstra 演算法[EW59]。Dijkstra 演算法是走訪路網圖中的每個節點 v，保留目前為止所找到從 V_s 到各節點 v 的最短路徑。Dijkstra 演算法使用了廣度優先的搜尋，以及動態規劃的概念來尋找最短路徑，其輸入包含了 $G(V, E, W)$、起始節點 V_s 與目的點 V_t，其演算法如圖 2-4 Dijkstra 演算法所示，而圖 2-3 為呼叫之主程式。

主程式名稱：Dijkstra_Call 輸入：$G(V, E, W)$、V_s、V_t 變數：PreviousList 代表 V_s 到各節點 v 目前已知的最短路徑 Min_DistanceList 代表 V_s 到各節點 v 目前已知的最短距離 輸出： $P_{s,t}$、$D_{s,t}$
L01　**Initialize** PreviousList←null, Min_DistanceList←infinity; L02　{PreviousList, Min_DistanceList}←

```
      Dijkstra(G(V, E, W), Vs, PreviousList, Min_DistanceList);
L03   vertex ← Vt ;
L04   for ( ; vertex != Vs; vertex ← previous[vertex])
L05      path.push_front(vertex);
L06   end for
L07   path.push_front(Vs);
L08   Ps,t ← path ;
L09   Ds,t ← Min_DistanceList [Vt];
```

圖 2-3　Dijkstra Call 演算法

演算法名稱：Dijkstra

輸入：$G(V, E, W)$、V_s、PreviousList、Min_DistanceList

變數：$v \in V$ 代表各節點、u 表示當下節點

```
L01   Initialize   Q←V;
L02   Min_DistanceList[Vs]←0;
L03   while Q is not an empty set
L04      u ←Extract_Min(Q);
L05      for each edge outgoing from u as (u,v)
L06         if Min_DistanceList[v]>Min_DistanceList[u]+W(u,v) then
L07            Min_DistanceList[v]←Min_DistanceList[u]+W(u,v) ;
L08            PreviousList[v] ← u ;
L09         end if
L10      end for
L11   end while
```

圖 2-4　Dijkstra 演算法

以下解釋圖 2-4。演算法控制 Q 集合，Q 集合初始為所有 V 內的節點，因為初始節點到起點間不需移動任何距離，所以可以直接將起點到 V_s 的最小距離設為 0。L03-04，每一步都有一個被選擇的節點 u，而 u 是 Q 中擁有最小 Min_DistanceList[u]值的節點。L05，當一個節點 u 從 Q 中被選出後，演算法對每條外接邊(u, v)進行擴展。L06-L08，進行擴展時如果存在一條從 u 到 v 的邊，即表示 V_s 到 v 有路徑可以走過去，這條路徑的長度是 Min_DistanceList[u]+$W(u,v)$，

若此長度比目前已知的 Min_DistanceList[*v*]值還要小，我們即用此值來替代當前 Min_DistanceList[*v*]中的值。當所有的邊都擴展完後，Min_Distanc[*v*]就是 V_s 至各節點的最短路徑距離，若路徑不存在時 Min_DistanceList[*v*]將是無窮大。

[範例 2-1] 針對圖 2-1，使用 Dijkstra 演算法取得 $P_{0,5}$。首先，初始的狀態表如表 2-4 所示，因為 V_0 可以直接到達 V_1、V_2 所以距離為已知，而 V_3、V_4、V_5 未知能否到達，所以暫以∞標示，接下來是從尚未找到最短路徑的節點中，選擇路徑長度最小的節點，將它設為 True 代表已找到最短路徑，並計算透過節點間的路徑後是否為較短。接著，如表 2-5 所示，繼續從 V_1、V_2 中尋找最短路徑的節點，可以發現 V_0 到 V_1 的是最短距離，也發現 V_1 可通往 V_3 和 V_4，總距離分別為 15、16。如表 2-6，選擇 V_2 後，觀察由 V_2 到 V_3 的路徑長度為 17、由 V_2 到 V_4 的路徑長度為 18，將因繞道而變得更長，所以我們不修正路徑。表 2-7 為選擇 V_3 後，V_3 到達目的地 V_5 的路徑長度為 29。最後，如表 2-8，選擇 V_4 後，V_4 到達目的地 V_5 的路徑長度為 25，將因繞道而變得更短，所以我們將走訪到 V_5 的路徑長度修正為 25，$P_{0,5} = V_0 \rightarrow V_1 \rightarrow V_4 \rightarrow V_5$ 、$D_{s,t}=25$。

表 2-4 Dijkstra 演算法圖例初始狀態

頂　點	0	1	2	3	4	5
是否找到最短路徑	True					
路徑長度	0	6	8	∞	∞	∞
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$			

表 2-5 Dijkstra 演算法圖例進行第 1 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True				
路徑長度	0	6	8	15	16	∞
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$	$V_0 \rightarrow V_1 \rightarrow V_3$	$V_0 \rightarrow V_1 \rightarrow V_4$	

表 2-6 Dijkstra 演算法圖例進行第 2 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True	True			
路徑長度	0	6	8	15	16	∞
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$	$V_0 \rightarrow V_1 \rightarrow V_3$	$V_0 \rightarrow V_1 \rightarrow V_4$	

表 2-7　Dijkstra 演算法圖例進行第 3 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True	True	True		
路徑長度	0	6	8	15	16	29
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$	$V_0 \rightarrow V_1 \rightarrow V_3$	$V_0 \rightarrow V_1 \rightarrow V_4$	$V_0 \rightarrow V_1 \rightarrow V_3 \rightarrow V_5$

表 2-8　Dijkstra 演算法圖例進行第 4 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True	True	True	True	
路徑長度	0	6	8	15	16	25
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$	$V_0 \rightarrow V_1 \rightarrow V_3$	$V_0 \rightarrow V_1 \rightarrow V_4$	$V_0 \rightarrow V_1 \rightarrow V_4 \rightarrow V_5$

2.3　GSP-DF 演算法

在此節中，我們討論論文[RT13]如何解決**[問題 2-2]**，該作法是基於第 2.2 節的動態規劃作法而修正得來的，我們稱為 GSP-DF 演算法。GSP-DF 是將起始點 V_s 經過每個 C_i 內的所有項目，再到目的地 V_t 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 C_i 內距離最短的項目做為該 C_i 的代表節點，最後依序走訪代表節點即為最短路徑 $P_{s,t}$，而 GSP-DF 計算式如圖 2-5 所示。i 代表第 i 個種類編號、j 代表第 i 種類的第 j 種項目、Distance 代表兩種項目節點間的距離，而剛開始走訪表示從起點 C_0 開始走，所以 $X[0, 1]=0$。

$$X[i,j] = \begin{cases} 0 & \boldsymbol{if}\ i = 0 \\ \min\limits_{1 \le L \le |C_i - 1|} \{X[i-1, L] + \text{Distance}(C_{i-1,L}, C_{i,j})\} & \boldsymbol{if}\ i > 0 \end{cases}$$

圖 2-5　GSP-DF 計算式

演算法名稱：GSP-DF
輸入：C, iMax , jMax
輸出：final_road 最短路徑
變數：X 代表儲存最短路徑距離之矩陣
　　VisitID 代表最後走訪路徑的順序(為二維陣列，其中列代表種類，行代表項目)
　　Ci_min 代表各種類內路徑長度的最小值陣列

```
L01     for i←0 to iMax do
L02       Ci_min_dis←∞;
L03       for j←1 to jMax do
L04         if  i is 0 then
L05           X[0][1]←0;
L06           VisitID[0][1]←0;
L07         else
L08           min_dis←∞;
L09           tmp_dis←0;
L10           for L←1 to Ci-1.length do
L11             tmp_dis=X[i-1][L]+Distance(Ci-1,L, Ci,j);
L12             if min_dis  ≥  tmp_dis then
L13               mis_dis←tmp_dis;
L14               tmp_vid←C[i-1][L];
L15             end if
L16           end for
L17         end if
L17         X[i][j]←min_dis;
L18         VisitID[i][j]←tmp_vid;
L19         if Ci_min_dis>min_dis then
L20           Ci_min_dis←min_dis;
L21         end if
L22       end for
L23       Ci_min[i]←Ci_min_dis;
L24     end for
L25     for i←0 to Ci_min.length do
L26       for j←0 to C[i].length do
```

```
L27          min_dis←Ci_min[i];
L28          if min_dis is X[i][j] then
L29             final_road.insert(VisitID[i][j]);
L30              break;
L31          end if
L32       end for
L33    end for
L34    final_road.insert(Vt);
L35    return final_road;
```

圖 2-6 GSP-DF 演算法

圖 2-6 為 GSP-DF 的演算法，L01-L24 的演算範圍是 C_i 個數 ($0 \le i \le$ iMax)，L03-22 演算範圍是 C_i 內的項目個數 j($1 \le j \le$ jMax)。L05-06 是代表第一個走訪節點 $C_{0,1}$ 所以距離為 0，L08-15 則依照 GSP-DF 的計算式，其中 min_dis 是記錄著 *X[i][j]*的最小值，最小值的節點則記錄於 VisitID*[i][j]* 內。L19-23 是計算 *X[i]* 內最短的距離值並儲存於 Ci_min 內，當整個 *X* 矩陣皆計算完成後，L25-L33 是挑選出每個種類內最短距離的節點並記錄於 final_road 內。最後，整個演算法執行完畢後，final_road 即為避開淹水區域之最短路徑。

[範例 2-2] 規劃一條由家裡出發繞經加油站及郵局再到餐廳的最短路徑。

以圖 2-1 為例，將所有節點編入 4 個種類， $C_0=\{V_0\}$代表家裡、$C_1=\{V_1,V_2,V_3\}$代表加油站、$C_2=\{V_4\}$代表郵局及 $C_3=\{V_5\}$代表餐廳。我們執行 GSP-DF 演算法及觀察 *X* 的變化，如圖 2-7(a)、(b)所示。首先，*X[1, j]*儲存著從 $C_{0,1}$ 走到 C_1 各節點的距離分別為 6、8、14，而 *X[2,1]*是儲存由 C_1 各節點再走到 $C_{2,1}$ 的最小距離，由於 $C_{1,3}$ 走不到 $C_{2,1}$ 所以距離是無限大。最後，最短路徑走訪節點的順序即為 min_dis 欄位對應的 C_i 代表節點 VisitID，我們可以得到路徑 $C_{0,1} \rightarrow C_{1,1} \rightarrow C_{2,2} \rightarrow C_{3,1}$，而對應節點編號後 $P_{0,5}=V_0 \rightarrow V_1 \rightarrow V_4 \rightarrow V_5$、$D_{0,5}=25$。

初始狀態

X	0	1	2	3
1	∞	∞	∞	∞
2	∞	∞	∞	∞
3	∞	∞	∞	∞

i =0

X	0	1	2	3
1	0	∞	∞	∞
2	∞	∞	∞	∞
3	∞	∞	∞	∞

i =1

X	0	1	2	3
1	0	6	∞	∞
2	∞	8	∞	∞
3	∞	14	∞	∞

i =2

X	0	1	2	3
1	0	6	16	∞
2	∞	8	∞	∞
3	∞	14	∞	∞

i =3

X	0	1	2	3
1	0	6	16	25
2	∞	8	∞	∞
3	∞	14	∞	∞

(a) *X* 矩陣之變化

Matrix	Step	min_dis	VisitID
X[0,1]	-	0	-
X[1,1]	*X*[0,1]+Distance($C_{0,1}$, $C_{1,1}$) = 0+6 = 6	6	$C_{0,1}$
X[1,2]	*X*[0,1]+ Distance ($C_{0,1}$, $C_{1,2}$) = 0+8 = 8	8	
X[1,3]	*X*[0,1]+ Distance ($C_{0,1}$, $C_{1,3}$) = 0+14 = 14	14	
X[2,1]	*X*[1,1]+ Distance ($C_{1,1}$, $C_{2,1}$) = 6+10 = 16 *X*[1,2]+ Distance ($C_{1,2}$, $C_{2,1}$) = 8+10 = 18 *X*[1,3]+ Distance ($C_{1,3}$, $C_{2,1}$) =∞	16	$C_{1,1}$
X[3,1]	*X*[2,1]+ Distance ($C_{2,1}$, $C_{3,1}$) = 16+9 = 25	25	$C_{2,1}$

(b) *X* 矩陣計算步驟

圖 2-7　GSP-DF 演算法範例進行計算之結果

2.4 問題假設

在本論文中，我們假設所有的路網圖資料均分佈在歐式空間，其中，路網圖是由節點與邊組成。函數 Distance(V_1,V_2)代表從點 V_1 座標至點 V_2 座標的大圓距離（Great-circle distance）[GB97]，使用的座標系統為 WGS84（World Geodetic System 1984），其公式為：

Haversine formula: $a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$

$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a})$

$\text{distance} = R \cdot c$

φ_1、φ_2 為 V_1 與 V_2 的經度，$\Delta\varphi$、$\Delta\lambda$ 為 V_1 與 V_2 經度和緯度相差的角度，R 為地球半徑(半徑=6378km)。函數 atan2 是正切函數的一個變種，對於任意不同時等於 0 的實參數 x 和 y，atan2(y, x)所表達的意思是坐標原點為起點，指向(x, y)的射線在坐標平面上與 x 軸正方向之間的角的角度。

另外，本論文的輸入資料皆以經緯度表示，但為了方便起見，經緯度會分別以 x 軸和 y 軸座標來標示。

第 3 章　Baseline 方法

在本章中，我們討論如何利用先前提到的 Dijkstra 演算法，來解決本論文提出的 USP 問題**([問題 2-3])**。本方法會先找出經過淹水區域的路段，將其從路網圖中剔除，然後交由 Dijkstra 演算法進行路徑規劃，此方法稱作 Baseline 方法。

3.1　資料結構

首先，我們介紹本方法輸入資料的資料結構，路網圖 *G(V, E, W)*的輸入檔案如表 3-1(a)所示，其中一行代表一段路兩個路口間的經緯度座標與距離，中間以空白隔開，順序為 Point1x、Point1y、Point2x、Point2y、weight。由於在判斷淹水區域與路網是否重疊時，我們會需要建立 R-tree，而插入 R-tree 的節點需為一個涵蓋空間物件的最小矩形(Minimum Bounding Rectangle，簡稱 MBR)，所以我們必須進行 MBR 的轉換，也就是將一個矩形或線段取 x、y 軸的最小及最大值，順序為 Xmin、Ymin、Xmax、Ymax 中間皆以空白隔開，如表 3-1(b)所示。另外，本方法另一個輸入為淹水區域的檔案 FRFile，如表 3-2(a)所示，其中一列代表一個淹水區域，一個淹水區域有四個節點，每個節點是一組經緯度座標。每一個淹水區域也必須進行 MBR 的轉換，如表 3-2(b)所示，轉換為 MBR 後再建立 R-tree。

表 3-1　輸入的路網圖檔案(RoadFile)

(a) RoadFile 範例

行數	內容				
1	25.05	121.0	25.10	121.1	0.2
2	25.10	121.0	25.15	121.1	0.3
3	25.15	121.0	25.05	121.1	0.4

(b) RoadFile 轉換為 MBR 的範例

行數	內容			
1	25.05	121.0	25.10	121.1
2	25.10	121.0	25.15	121.1
3	25.05	121.0	25.15	121.1

表 3-2　輸入的淹水區域檔案(FRFile)

(a) FRFile 範例

行 數	內 容
1	25.6　121.7　25.6　121.8　25.7　121.8　25.7　121.7
2	25.6　121.5　25.6　121.6　25.8　121.6　25.8　121.5

(b) FRFile 轉換為 MBR 的範例

行 數	內 容
1	25.6　121.7　25.7　121.8
2	25.6　121.5　25.8　121.6

接著，由於 Dijkstra 在執行路徑規劃運算時，需要一份完整路網圖的相鄰串列(Adjacent List)，所以我們根據路網圖建立成 Adjacency List，依序儲存著路網圖中所有 V_i 到 V_j 的邊及距離，如圖 3-1 即為 Adjacency List 的結構，針對一個節點 V_i，紀錄其所能走到的所有節點 V_j、V_{j+1}…等及兩點間的距離。

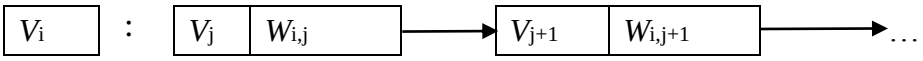


圖 3-1　Adjacency List 結構

3.2　R-tree 建樹方法

本方法中的一個重要運算，是找出經過淹水區域的道路。一個最基本的想法，是將如表 3-1(b)的 RoadFile 中的每一列，和如表 3-2(b)的 FRFile 中的每一列，判斷兩者是否有交集，但是此作法的時間複雜度很高。在此我們討論使用不同種方法來建立 R-tree 的效率影響。為了方便討論，本節以圖 3-2 舉例，其中共有 8 個路口節點、9 條道路邊及 6 個灰色淹水區域，接下來我們會討論兩種建立 R-tree 的作法及時間複雜度。

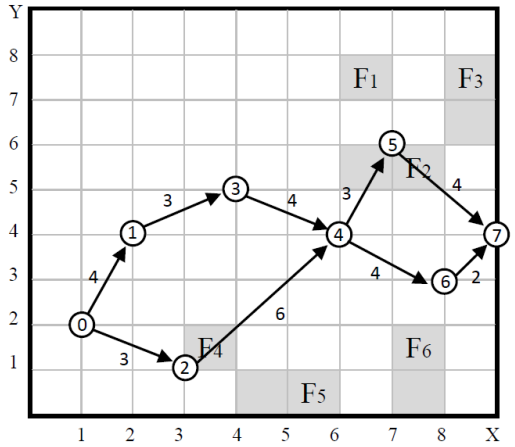


圖 3-2　範例路網圖

1、使用道路來建立 R-tree：

如圖 3-3 所示，我們首先將每條邊取 MBR，例如：R_{01}：$V_0 \rightarrow V_1$ 的 MBR 即(1 2 2 4)、R_{02}：$V_0 \rightarrow V_2$ 的 MBR 即(1 1 3 2)，然後再將相鄰的 MBR 合併成更大的 MBR，如圖中的虛線所示。根據這些 MBR 建立 R-tree，假設每個節點最多存放 3 個空間，也就是 order M 為 3，則所建立的 R-tree 如圖 3-4 所示，所建立的樹高度是 3 層，而建立樹的時間複雜度[1]為 $O(N)$即 $O(9)$，其中 N 為 value 個數。

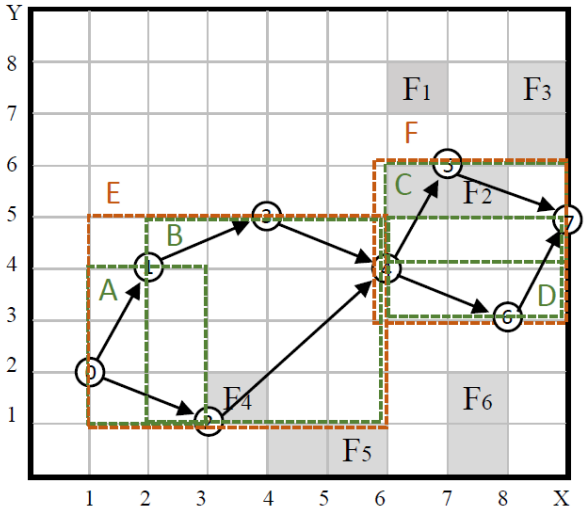


圖 3-3　道路形成的 MBR

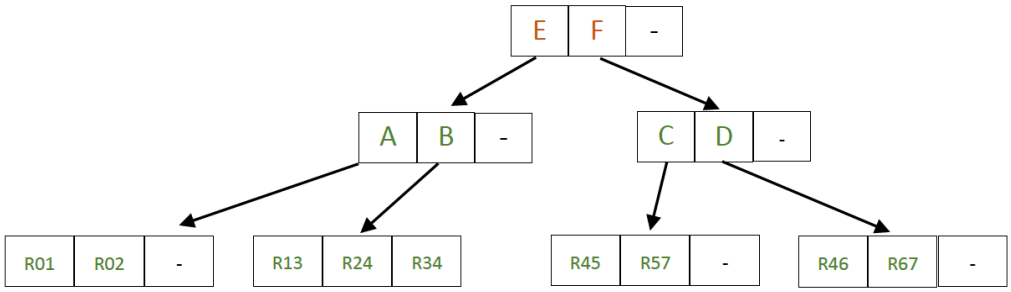


圖 3-4　使用道路建立的 R-tree

接著，我們使用淹水區域一一來查詢這棵 R-tree。首先 F_1 的 MBR(6 7 7 8)與 root 的兩個空間 E 和 F 都沒有交集，所以查詢不到重疊的道路；接著，F_2 的 MBR(6 5 8 6)則會與 root 的 F 空間和第二層的 C 空間相交，最後會回傳 leaf 的 R45 和 R57；...依此類推直至所有淹水區塊都判斷完成。查詢樹的時間複雜度[2]為 $O(M\log M^N)$即 $O(3\log 3^9)$=12.88。

[1] R-tree Time complexity (Worst Case-Insert)，資料來源 http://en.wikipedia.org/wiki/R-tree

[2] R-tree Time complexity (Average-Search)，資料來源 http://en.wikipedia.org/wiki/R-tree

2、使用淹水區域來建立 R-tree：

如圖 3-5 的虛線所示，我們將鄰近的 2-3 個淹水區域取 MBR。則此 6 個淹水區域所建立的樹高度是 2 層，如圖 3-6 所示，建立樹的時間複雜度為 $O(N)$即 O(6)。

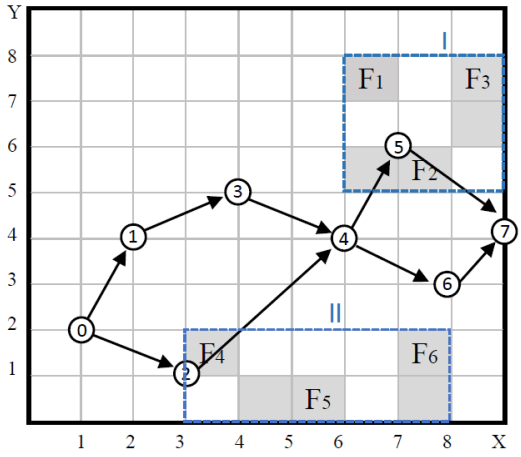


圖 3-5　淹水區域的 MBR

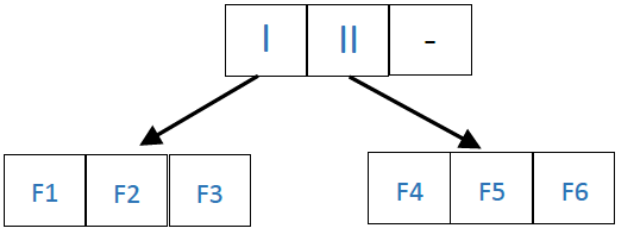


圖 3-6　使用淹水區域建立的 R-tree

接著，我們使用每條道路的 MBR 來查詢這棵 R-tree。首先，R01：$V_0 \rightarrow V_1$ 的 MBR (1 2 2 4)與 root 的兩個空間 I 和 II 都沒有交集，查不到重疊部分；接著，R24：$V_2 \rightarrow V_4$ 的 MBR(3 1 6 4)則查詢到與 “II→F_4”重疊、…依此類推直至所有道路都判斷完成。查詢樹的時間複雜度為 $O(M\log M^N)$即 $O(3\log 3^6)$=8.59。

由以上範例觀察得知，影響整個效率的關鍵包含插入樹節點的數量 N，以及 MBR 重疊的影響。由於以道路所建立的 R-tree，不僅樹較高，MBR 重疊的現象也嚴重，所以本論文採用以淹水區域建立 R-tree。

3.3 Baseline 演算法

在本節中我們列出完整的 Baseline 演算法，如圖 3-7 所示。首先，L01 是將相關變數初始化，L02-05 是將所有 FRFile 內的淹水區域矩形取 MBR 後建立 R-tree，而取得淹水矩形的 MBR 方法就是以順時鐘方向的左下節點 $V_{LD.x}$、右上節點 $V_{RU.x}$ 比較座標 x、y 的最小、最大值。L06-08 是將 RoadFile 內的道路取得兩點間的道路 MBR 再向 R-tree 查詢是否有重疊並將重疊區域記錄於 Intersect 內。接著，L09-20 若 Intersect 內沒有任何區域則將該條道路的 V_i、V_j 節點建立 C2IDTable 及 ID2CTable 對應表，資料依序由 0 開始建立節點編號，並同時記錄節點編號對應的經緯度，同時將邊與距離加入 AdjacencyList 裡。然後，L21 是我們開始進行 Dijkstra 演算法的路徑規劃，L22-23 即為 V_s 至 V_t 走訪的最短距離 $D_{s,t}$ 及最短路徑 $P_{s,t}$，而路徑的顯示部分我們會在第五章進一步的說明。

最後，我們來分析演算法中的各項目時間複雜度。首先，R-tree 的建樹總共的時間複雜度為 $O(N)$，N 代表淹水區域的個數；總共查詢樹的時間為 $O(E \cdot M\log M^N)$，E 為路網圖道路數及 leaf 最多可包含 M 個空間；Dijkstra 路徑規劃的時間複雜度 $O(E \cdot V\log V)$，V 代表路口點數量。所以整個方法的時間複雜度為 $O(N+E*(V\log V+M\log M^N))$。

演算法名稱：Baseline_main 輸入：RoadFile, RoadFileMBR, V_s, V_t, FRFile 變數：PreviousList、Min_DistanceList、 C2IDTable、ID2CTable 代表節點 ID 與經緯度的對應表 輸出： $P_{s,t}$、$D_{s,t}$
L01 **Initialize** C2IDTable←null; ID2CTable←null; AdjacencyList←null; PreviousList←null; Min_DistanceList←infinity; L02 **for each**$(V_{LD.x}, V_{LD.y}, V_{LU.x}, V_{LU.y}, V_{RU.x}, V_{RU.y}, V_{RD.x}, V_{RD.y})$∈FRFile **do** L03 {xMin, yMin, xMax, yMax}=TransMBR$(V_{LD.x},V_{LD.y},V_{RU.x},V_{RU.y})$; L04 Rtree.insert(xMin, yMin, xMax, yMax); L05 **end for** L06 **for each** $(V_{i.x}, V_{i.y}, V_{j.x}, V_{j.y}, W_{i,j})$ ∈ RoadFile **do** L07 {xMin, yMin, xMax, yMax}=TransMBR$(V_{i.x}, V_{i.y}, V_{j.x}, V_{j.y})$;

```
L08     Intersect ← Rtree.search(xMin, yMin, xMax, yMax);
L09     if ( Intersect.size( )≤ 0 ) then
L10       if C2IDTable[Vi] is NULL   then
L11         C2IDTable[VID++]←Vi ;
L12         ID2CTable[Vi]  ←VID;
L13       end if
L14       if C2IDTable[Vj] is NULL   then
L15         C2IDTable[VID++]←Vj ;
L16         ID2CTable[Vj]  ←VID;
L17       end if
L18       AdjacencyList.insert(C2IDTable[Vi], C2IDTable[Vj], Wi,j);
L19     end if
L20   end for
L21   {PreviousList, Min_DistanceList }←
        Dijkstra(Adjacency_List, Vs, PreviousList, Min_DistanceList);
L22   Ps,t← PreviousList[Vt] ;
L23   Ds,t← Min_DistanceList[Vt];
```

```
TransMBR(V1x,V1y,V2x,V2y)
```

```
L01     if (V1x ≤ V2x ) then
L02       xMin←V1x; xMax←V2x;
L03     else
L04       xMin←V2x; xMax←V1x;
L05     if (V1y ≤ V2y ) then
L06       yMin←V1y; yMax←V2y;
L07     else
L08       yMin←V2y; yMax←V1y;
L09     return xMin, yMin, xMax, yMax;
```

圖 3-7　Baseline 演算法

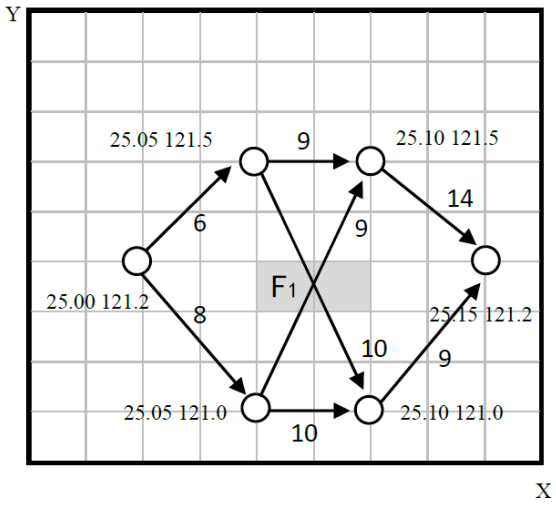


圖 3-8　含有淹水區域的經緯度路網圖

以下我們針對圖 3-8，給定起點 V_s 為(25.00 121.2)、目的地 V_t 為(25.15 121.2)，解釋如何使用 Baseline 方法，取得 $P_{s,t}$ 路徑。

首先，如圖 3-7 的 L02-05 我們先處理輸入的資料，我們使用淹水區域來建立 R-tree，將表 3-3 的淹水區域取 MBR。接著，L06-08 我們使用節點間的路道來查詢這棵 R-tree，將沒有查詢到重疊的道路建立 C2IDTable、ID2CTable 對應表，如 表 3-4 及 表 3-5。然後，L18 建立 AdjacencyList，因為 V_1 至 V_4、V_2 至 V_3 與淹水區域 F_1 有交集，所以我們不會將它們加入 AdjacencyList 裡，執行結果如表 3-6 所示。

表 3-3　FRFile 檔案

index	矩形位置
F_1	25.05 121.3 25.10 121.3 25.10 121.1 25.05 121.1

表 3-4　C2IDTable 執行結果

節點經緯度	節點編號
25.00　121.2	0
25.05　121.5	1
25.05　121.0	2
25.10　121.5	3
25.10　121.0	4
25.15　121.2	5

表 3-5　ID2CTable 執行結果

節點編號	節點經緯度
0	25.00　121.2
1	25.05　121.5
2	25.05　121.0
3	25.10　121.5
4	25.10　121.0
5	25.15　121.2

表 3-6　無淹水路段的 Adjacency List

V_i	V_j	$W_{i,j}$
V_0	V_1	6
	V_2	8
V_1	V_3	9
V_2	V_4	10
V_3	V_5	14
V_4	V_5	9

而 Dijstra 初始的狀態表如表 3-7 所示，因為 V_0 可以直接到達 V_1、V_2 所以距離為已知，而 V_3、V_4、V_5 未知能否到達，所以暫以∞標示，接下來是從尚未找到最短路徑的節點中，選擇路徑長度最小的節點，將它設為 True 代表已找到最短路徑，並計算透過節點間的路徑後是否為較短。接著，如表 3-8 所示，繼續從 V_1、V_2 中尋找最短路徑的節點，可以發現 V_0 到 V_1 的是最短距離，也發現 V_1 可通往 V_3 而總距離為 15。如表 3-9，選擇 V_2 後，V_2 到 V_4 的路徑總距離為 18。表 3-10 為選擇 V_3 後，V_3 到達目的地 V_5 的路徑總距離為 29。如表 3-11，選擇 V_4 後，V_4 到達目的地 V_5 的路徑總距離為 27，將因繞道而變得更短，所以我們將走訪到 V_5 的路徑長度修正為 27，$P_{0,5}=V_0\rightarrow V_2\rightarrow V_4\rightarrow V_5$，即“25.00 121.2”→“ 25.05 121.0”→“ 25.10 121.0”→“25.15 121.2”。

表 3-7　Baseline 方法進行初始狀態

頂　點	0	1	2	3	4	5
是否找到最短路徑	True					
路徑長度	0	6	8	∞	∞	∞
路　徑	V_0	$V_0\rightarrow V_1$	$V_0\rightarrow V_2$			

表 3-8　Baseline 方法進行第 1 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True				
路徑長度	0	6	8	15	∞	∞
路　徑	V_0	$V_0\rightarrow V_1$	$V_0\rightarrow V_2$	$V_0\rightarrow V_1\rightarrow V_3$		

表 3-9　Baseline 方法進行第 2 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True	True			
路徑長度	0	6	8	15	18	∞
路　徑	V_0	$V_0\rightarrow V_1$	$V_0\rightarrow V_2$	$V_0\rightarrow V_1\rightarrow V_3$	$V_0\rightarrow V_2\rightarrow V_4$	

表 3-10　Baseline 方法進行第 3 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True	True	True		
路徑長度	0	6	8	15	18	29
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$	$V_0 \rightarrow V_1 \rightarrow V_3$	$V_0 \rightarrow V_2 \rightarrow V_4$	$V_0 \rightarrow V_1 \rightarrow V_3 \rightarrow V_5$

表 3-11　Baseline 方法進行第 4 次迭代後之結果

頂　點	0	1	2	3	4	5
是否找到最短路徑	True	True	True	True	True	
路徑長度	0	6	8	15	18	27
路　徑	V_0	$V_0 \rightarrow V_1$	$V_0 \rightarrow V_2$	$V_0 \rightarrow V_1 \rightarrow V_3$	$V_0 \rightarrow V_2 \rightarrow V_4$	$V_0 \rightarrow V_2 \rightarrow V_4 \rightarrow V_5$

第 4 章　Cloud 方法

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路徑規劃，我們提出的 Cloud 方法會整合圖資網站既有的路徑規劃功能，淹水區域資料與 GSP-DF 演算法，快速輸出迴避淹水區域的路徑。

4.1　系統架構

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 4-1所示。首先，當使用者輸入一個Query後，主程式會啟動『雲端路徑查詢模組』與圖資平台連結並查詢最短路徑。接著，我們會先判斷這條路徑是否有跟任何一個淹水區域重疊，若有重疊的道路我們將啟動『替代節點篩選模組』，查詢附近未淹水的路口點作為代替。最後，經由GSP-DF演算法來規劃繞經替代點集合的最短路徑，而為了能夠讓使用者便於觀看最後輸出的路徑，我們將查詢結果以網頁圖型方式顯示於『Result Display畫面』，作法是將規劃出來的替代節點設為必經點送至Google Maps再次規劃路徑，若還是不幸經過淹水區域，我們也在網頁中附上手動調整路線的功能，以增加成功率。

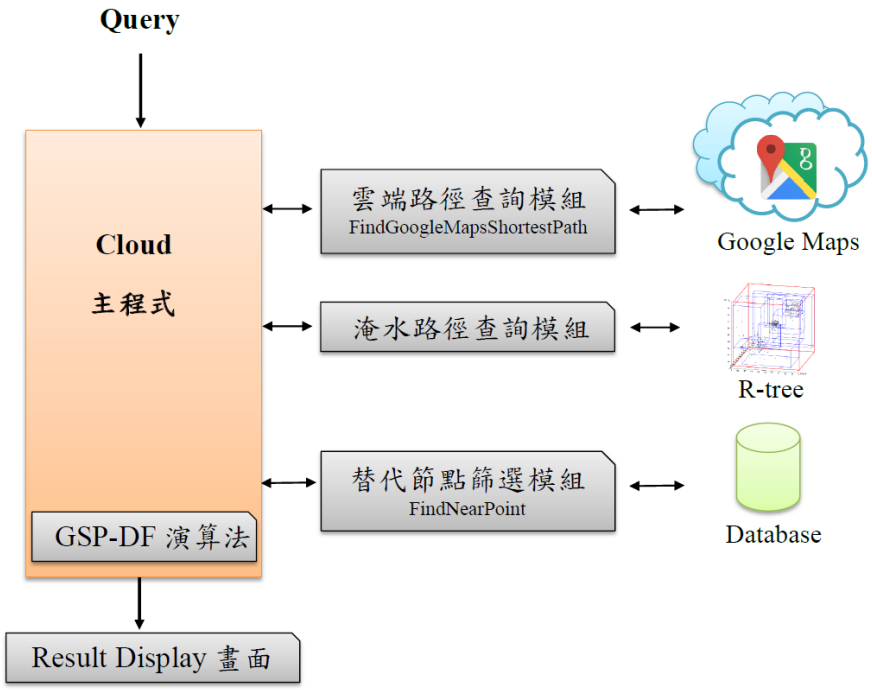


圖 4-1　Cloud方法系統架構

4.2 雲端路徑查詢模組

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[3]、TGOS[4]及 Google Maps[5]，而我們選用 Google Maps 平台的原因，是因為 Google Maps 在台灣是絕大多數使用者習慣的操作平台，且路網圖的資料也經常更新，最重要的是提供開發者許多 API 資源，例如我們需要的即時路徑規劃功能。而 Google Maps API 的路徑規劃僅需輸入 V_s 與 V_t 即可規劃出一條最短路徑，甚至更可以設定必經點 Waypoints，讓路徑特別經過 Waypoints 集合再到目的地 V_t。本論文選用的 Google Maps API 為 JavaScript 版本，所以我們在執行該模組時會開啟中介網頁並傳入參數，再由中介網頁向 Google Maps 網站查詢並輸出路徑，其演算法如圖 4-2 所示。

演算法名稱：FindGoogleMapsShortestPath 輸入：SearchURL、V_s、V_t 變數：RoadBox 道路暫存集合 輸出：GmspList 代表 GoogleMaps 規劃 V_s 到 V_t 的最短路徑檔案

L01 **Initialize** GmspList ; //初始值為空
L02 **WebRequest** WRGM = WebRequest.Create(SearchURL, V_s, V_t);
L03 RoadBox= WRGM.GetResponse( );
L04 **for each line** (V_i,V_j) ∈ RoadBox **do**
GmspList.insert((V_i,V_j));
L05 **end for**
L06 **outputFile** GmspList;

圖 4-2 FindGoogleMapsShortestPath 演算法

初始時，GmspList 為空，L02 是使用 WebRequest 通訊協定開啟查詢路徑的中介網頁 SearchURL[6]，並傳入起始點 V_s、目的地 V_t，L03，當接收到 Google Maps 回傳的資料後，將路徑走訪的路段存於 RoadBox 集合暫存。最後，L04-06 是將 RoadBox 集合內的所有路段輸出於 GmspList 檔案。而執行 Google Maps 路徑規劃的中介網頁，我們使用 JavaScript 版本的 Google Maps API 進行查詢，交通工具選為開車 DRIVING，程式片段如下：

3 Yahoo Maps : https://tw.maps.yahoo.com/
4 TGOS 地理資訊圖資雲服務平台: http://tgos.nat.gov.tw/
5 Google Maps : https://www.google.com.tw/maps
6 SearchURL 路徑規劃網址：http://localhost/flood/googlemaps.php

```
<script src="http://maps.google.com/maps/api/js?sensor=false">
var request = {
        origin: Vs,
        destination: Vt,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
};
</script>
```

[範例 4-1] 針對圖 3-8，若起點 V_s 是(25.00,121.2)、目的地 V_t 為(25.15,121.2)，使用雲端路徑查詢模組，取得 $P_{s,t}$ 路徑。我們可得到 GmspList 為“25.00 121.2”→“ 25.05 121.5”→“ 25.10 121.0”→“ 25.15 121.2”。

4.3 替代節點篩選模組

先前 4.2 節中取得的 GmspList 是尚未考慮淹水區域的路徑，接著，我們在主程式會再判斷 GmspList 與 FRFile 檔案內的淹水區域是否有任何重疊。而若有重疊的道路就必須尋找未淹水的替代點，而替代點集合的查詢通常是 Location Search，也就是給定一個 Query Location 及路網圖，找出 Query Location 之鄰近物件(節點)。而在本論文中，使用 RDB 資料庫來實作此功能。

首先，資料庫內已含有我們事先已建立好的路網圖資料。接著，向資料庫輸入選點的 SQL 查詢句進行 Table 的資料篩選，而回傳的結果我們也必須再次確認是否避開淹水區域，我們的作法輸入淹水區域資料 FRFile 並使用 RDB 的 MBRWithin 函數查詢，最後輸出的集合已經是避開淹水區域的節點。

Table：roadnetwork	
Columns	Type
VID	int
Point1x	double
Point1y	double
Point2x	double
Point2y	Double

圖 4-3 表格的 schema

路網圖資料的部份是儲存於 MySQL 內，表格定義如圖 4-3，roadnetwork 表格儲存著所有路網圖的邊，第 V_i 點的經緯度為 Point1x 與 Point1y，V_j 的經緯度為 Point2x 與 Point2y。接著，替代節點篩選模組會根據欲查詢的 V_i 節點向資料庫查詢替代節點，而查詢的範圍我們訂為 1km 內的所有節點，對應的 SQL 如下，注意到「*100」是為了進行單位的轉換：

```
Select distinct * from `roadnetwork` where st_distance(point(`point1x`,`point1y`),
point(Vi.x, Vi.y))*100 <= 1;
```

最後，為了避免選出的替代節點也在淹水區域內，我們使用 MBRWithin 函數判斷，若函數回傳值是“1”即代表與淹水區重疊，我們就不輸出該節點，而對應的 SQL 如下：

```
@FRList = GeomFromText('POLYGON({FRList})');
@FPoint = GeomFromText('Point(Point1x Point1y');
Select MBRWithin(@FRList,@FPoint);
```

以上功能以 PHP 實作，對應的程式碼如**附錄 A** 所示，而該程式碼所在位址為 DBURL[7]。接下來，完整的 FindNearPoint 演算法列於圖 4-4。初始時，NearPointSet 為空，L02 是使用 WebRequest 通訊協定向 DataBase 網站請求查詢，依序傳入網站網址 DBURL、FRFile 檔案及欲查詢 *v* 點。L03-04，當接收到 DataBase 網站回傳的資料後，將未淹水的節點輸出於 NearPointSet，而完整的 DBURL 網頁程式內容於**附錄 A** 內。

演算法名稱：FindNearPoint

輸入：DBURL、*v* 欲查詢的節點

　　FRFile 淹水區域檔案

輸出：NearPointSet 代表各 FPointList 的替代點集合

```
L01  Initialize NearPointSet ; //初始值為空
L02  WebRequest DBSearch = WebRequest.Create(DBURL, FRFile, v);
L03  NearPointSet = DBSearch.GetResponse( );
L04  outputFile NearPointSet;
```

圖 4-4　FindNearPoint 演算法

4.4 Cloud 主程式

完整的 Cloud 演算法如圖 4-5 所示，當使用者進行 Cloud 方法查詢最短路徑時，依序輸入起始點 V_s、目的地 V_t 與淹水區域 FRFile。首先，L01 我們先呼叫『雲端路徑查詢模組』查詢 V_s 至 V_t 的最短路徑 GmspList。接著，L02-05 是將淹水區域 FRFile 取得 MBR 後建立 R-tree，L06-13 查詢 GmspList 中道路的 MBR 是否與 R-tree 內的葉節點重疊，若有經過淹水區域的邊則儲存於 FEList 集合。L14-29 是呼叫『替代節點篩選模組』進行替代節點的查詢並建立 C2IDTable、ID2CTable 對應表及 IntersectionSet 種類表，而 L12、L20 是將 V_s 及 V_t 加入種類表的第一個種類及最後一個種類。最後，L31 我們呼叫 GSP-DF 演算法進行路徑規劃，而 $P_{s,t}$

[7] 替代節點篩選的 PHP 程式位置 http://localhost/flood/FindNearPoint.php

就是迴避淹水區域的最短路徑，而 Result Display 的部分會於第五章解釋。

演算法名稱：Cloud_main

輸入：V_s、V_t、FRFile

變數：*C* 淹水點路口點類別
GMRemoteURL 圖資服務網址
FNRemoteURL 替代點查詢網址

輸出：$P_{s,t}$

```
L01  GmspList ← FindGoogleMapsShortestPath(GMRemoteURL,Vs,Vt);
L02  for each(VLD,VLU,VRU,VRD) ∈ FRFile do
L03     { xMin,yMin,xMax,yMax }=TransMBR(VLD.x,VLD.y,VRU.x,VRU.y);
L04     Rtree.insert(xMin,yMin,xMax,yMax);
L05  end for
L06  for each (Vi,Vj) ∈ GmspList do
L07     { xMin,yMin,xMax,yMax }=TransMBR(Vi.x, Vi.y, Vj.x, Vj.y);
L08     if Rtree.search(xMin,yMin,xMax,yMax) is TRUE then
L09           FEList.insert(( Vi,Vj ));
L10     end if
L11  end for
L12  IntersectionSet[0][1]←Vs;
L13  Initialize Ci←1; Cij←0; PIDX←0; jMax←0;
L14  for each ( Vi,Vj )∈ FEList do
L15   NPoint←FindNearPoint(FNRemoteURL, Vi, FRFile);
L16   for each v ∈ NPoint do
L17      if C2IDTable[v] is NULL then
L18         C2IDTable[v]←PIDX;
L19         ID2CTable[PIDX]←v;
L20         IntersectionSet[Ci][Cij]←PIDX;
L21         PIDX++;
```

```
L22          Cij++;
L23        end if
L14      end for
L24      if Cij > jMax then
L25        jMax←Cij;
L26      end if
L27      Cij←0;
L28      Ci++;
L29    end for
L30    IntersectionSet[Ci][1]←Vt;
L31    Ps,t←GSP-DF(IntersectionSet, Ci, jMax);
```

圖 4-5　Cloud 演算法

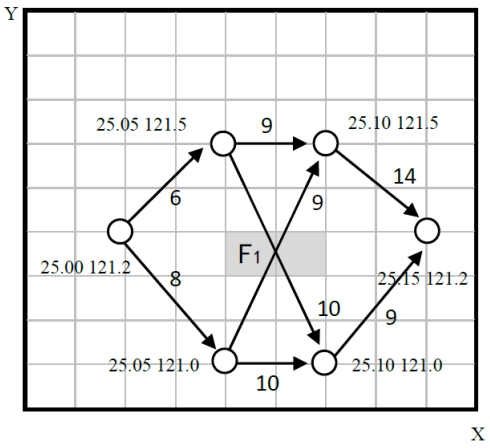


圖 4-6　含有淹水區域的經緯度路網圖

[範例 4-1]　針對圖 4-6，若起點 V_s 是(25.00 121.2)、目的地 V_t 為(25.15 121.2)，使用迴避淹水的路徑規劃 Cloud 方法，取得 $P_{s,t}$ 路徑。

首先，呼叫『雲端路徑查詢模組』查詢(25.00, 121.2)至(25.15, 121.2)的最短路徑 GmspList，如表 4-1 所示。接著，我們判斷 GmspList 走訪的路徑是否有經過淹水區域，若有則儲於 FEList 集合，由表 4-3 可得知節點 “25.05 121.5”→“25.10 121.5”的 MBR 與淹水區域重疊，所以呼叫『替代節點篩選模組』查詢經過淹水的路口點的替代點集合，由表 4-2 可得知 “25.05 121.5”是經過淹水區域的路口點，而查詢替代節點後回傳有 2 個替代節點。然後，將起點與目的地及替代節點建立 C2IDTable、ID2CTable 對應表，如表 4-4。最後，我們建立 3 個種類表，C_0={V_0}、C_1={V_1、V_2}、C_2={V_3}，並利用 GSP-DF 演算法計算最短路徑，如表 4-5。最後，我們得到走訪路徑為 $C_{0,1}$→$C_{1,1}$→$C_{2,1}$ 而對應編號表後，即 $P_{s,t}$

=“25.00 121.2”→“25.05 121.0”(→“25.10 121.0”)→“25.15 121.2”。

表 4-1　執行結果之 GmspList

順序	走訪節點
1	25.00 121.2
2	25.05 121.5
3	25.10 121.0
4	25.15 121.2

表 4-2　“25.05 121.5”之替代節點集合

順序	走訪節點
1	25.10 121.5
2	25.05 121.0

表 4-3　淹水的路口點

順序	節點
1	25.05 121.5

表 4-4　C2IDTable、ID2CTable

節點編號	節點經緯度
0	25.00 121.2
1	25.10 121.5
2	25.05 121.0
3	25.15 121.2

表 4-5　GSP-DF 演算法計算之 *X* 矩陣

Matrix	Step	min_dis	VisitID
X[0,1]	-	0	-
X[1,1]	*X*[0,1]+ Distance ($C_{0,1}$, $C_{1,1}$)=0+(6+9)=15	15	$C_{0,1}$
X[1,2]	*X*[0,1]+ Distance ($C_{0,1}$, $C_{1,2}$)=0+8=8	8	
X[2,1]	*X*[1,1]+ Distance ($C_{1,1}$, $C_{2,1}$)=15+14=29 *X*[1,2]+ Distance ($C_{1,2}$, $C_{2,1}$)=8+(10+9)=27	27	$C_{1,1}$

第 5 章　實驗

在本章，我們進行實驗來比較迴避淹水區域路徑之 Baseline 及 Cloud 兩種方法的差異。首先，先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 i7-2600 四核心且核心時脈為 3.4GHz，而記憶體為 4GB，所採用的作業系統為 64 位元的 Windows 7 企業版。

5.1　系統實作方法與輸出範例

Baseline 方法以 Microsoft Visual C++ 2010 進行實作。而 Cloud 方法除了主程式以 C++實作外，其替代節點篩選模組和雲端路徑查詢模組，則使用 Microsoft Visual C# 2010 與本機端架設 Apache Web Server & PHP5.5。

我們會將兩個方法回傳的路徑輸出於不同的網頁中，Baseline 方法是將得到的路徑以線段方式 (Polyline) 呈現在 Google Maps 上，而 Cloud 的方法則是將得到的路口以起點、必經點和終點的方法呼叫 Google Maps 進行路徑規劃與呈現。以下以起點 (25.03925, 121.52452)、終點 (25.03909, 121.51867) 做為範例顯示兩個作法的不同，如圖 5-1 藍色線段為 Google Maps API 的 Polyline 代表單純 Dijkstra 輸出之最短路徑規劃 (尚未判斷淹水區域)、圖 5-2 為 Baseline 迴避淹水區域的路徑規劃，注意到“丹陽街”的 MBR 與“仁愛路”的淹水區塊重疊，所以這條道路被視為是淹水的道路，Dijksktra 會繞遠路行走。而如圖 5-3 為初次 Google Maps 規劃的最短路徑 (尚未判斷淹水區域)、圖 5-4 為 Cloud 迴避淹水區域透過替代點的規劃路徑。注意到此再次規劃的路徑還是有小部分經過淹水區域，所以我們也提供手動調整的功能，圖 5-5 就是我們將節點“*B*”以手動方式上移一個路口來避開淹水區域。

圖 5-1　Dijkstra 最短路徑規劃畫面

圖 5-2　Baseline 方法迴避淹水區域的規劃路線

圖 5-3　GoogleMaps 最短路徑規劃的路線

圖 5-4　Cloud 方法規劃的路線(小部分經過淹水區域)

圖 5-5　手動調整 Cloud 方法的路線

5.2 資料集

我們實驗所使用的資料集包括交通部路網數值圖服務網[8]提供的『台北市路網圖』、國家災害防救科技中心[9]提供的台北市『一日暴雨 600mm 淹水潛勢圖』與人工產生出來的淹水區域資料。首先，台北市的路網圖資料集包含中正區及萬華區，共 7140 個節點、9314 條邊。淹水潛勢圖包含中正區及萬華區，如圖 5-6 為例，我們將範圍限制在淹水深度達 0.5m 以上的區域，如圖的灰色區域，為了方便計算，我們將其框成淹水區域面積皆為 0.01km^2的矩形(長寬皆 100m) ，共產生 130 淹水區塊。

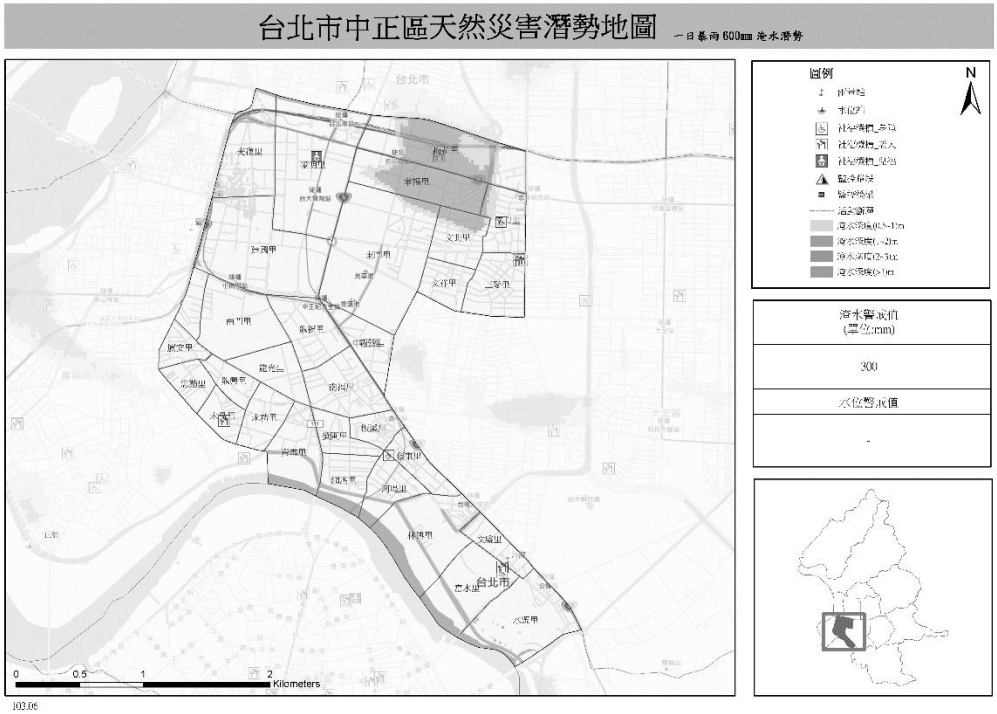


圖 5-6　淹水潛勢圖範例

人工所產生出來的台北市淹水區域資料的範圍為經度 121.4825~121.5353、緯度 25.0136~25.055，且根據隨機分佈與高斯分佈不重複的產生 0.01km^2(長寬皆 100m)的淹水區塊，分佈的情況如圖 5-7。這二種分佈代表的地理意義如下：隨機分佈中的淹水區塊代表著淹水區塊均勻的分佈於地圖中；而高斯分佈的淹水區塊代表淹水時，市中心為低窪的盆地地形，造成市中心淹水嚴重。所有資料集之各項參數如表 5-1。

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

[9] 國家災害防救科技中心 http://satis.ncdr.nat.gov.tw/

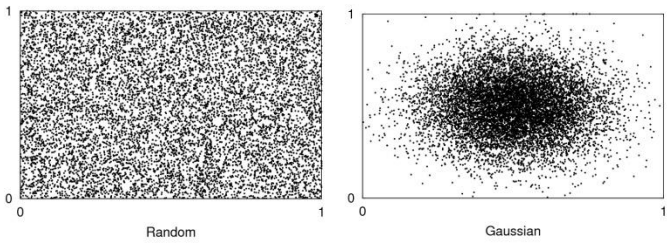


圖 5-7　資料集分佈的情況

我們測量執行時間的方式分為以下兩種：(1)從輸入淹水區域後至建立、查詢 R-tree 的 Spatial Join 時間、(2)從輸入資料開始至路徑規劃輸出的整體時間。每個查詢皆連續執行十次，量取系統第二次至第十次執行所花費時間的平均值作為比較。

表 5-1　資料集之各項參數

Parameter	**Setting**
真實數據資料	
Roadnetwork	台北市(節點：7140、路段：9314)
Number of Flooded Regions	130
人工產生資料	
Number of Flooded Regions	1, 10, 100
Flooded Regions Data Distribution	Random, Gaussian

5.3 淹水區域數量之實驗

第一個實驗，路網圖固定台北市路網圖 (7140 個節點及 9314 條邊)，我們著重於改變淹水區域資料集的數量，而查詢的路徑起點 V_s 為路網圖的極東北方節點 (25.04374444 121.5301511)、目的地 V_t 為極西南方節點 (25.02468882 121.4938111)，且分別記錄兩方法執行 Spatial Join 的時間與主程式整體的執行時間(路網圖 input 時間不列入計算)。

(1) 這次實驗，我們探討改變淹水區域的數量之影響，範圍限制在經度 121.4825~121.5353、緯度 25.0136~25.55 (面積大約為 $24km^2$) 內隨機產生“1”、“10”、“100”個淹水區域，而產生的淹水區域面積分別佔整體路網圖面積的$\frac{1}{2400}$、$\frac{1}{240}$、$\frac{1}{24}$，如圖 5-8。隨著淹水區域的增加，兩個方法的執行時間只有微幅上升，這是因為淹水區塊主要是用於建立 R-tree，受數量影響較小。不過以 Cloud 的效率較佳，達到 100 個淹水區域時，Cloud 的效率為 Baseline 的 8.7 倍。另外也觀察到，Spatial Join 在 Baseline 方法中佔據很大的部分，而在 Cloud 的方法中則隨著淹水區域增加快速增加，表示這部分仍有改善的空間。

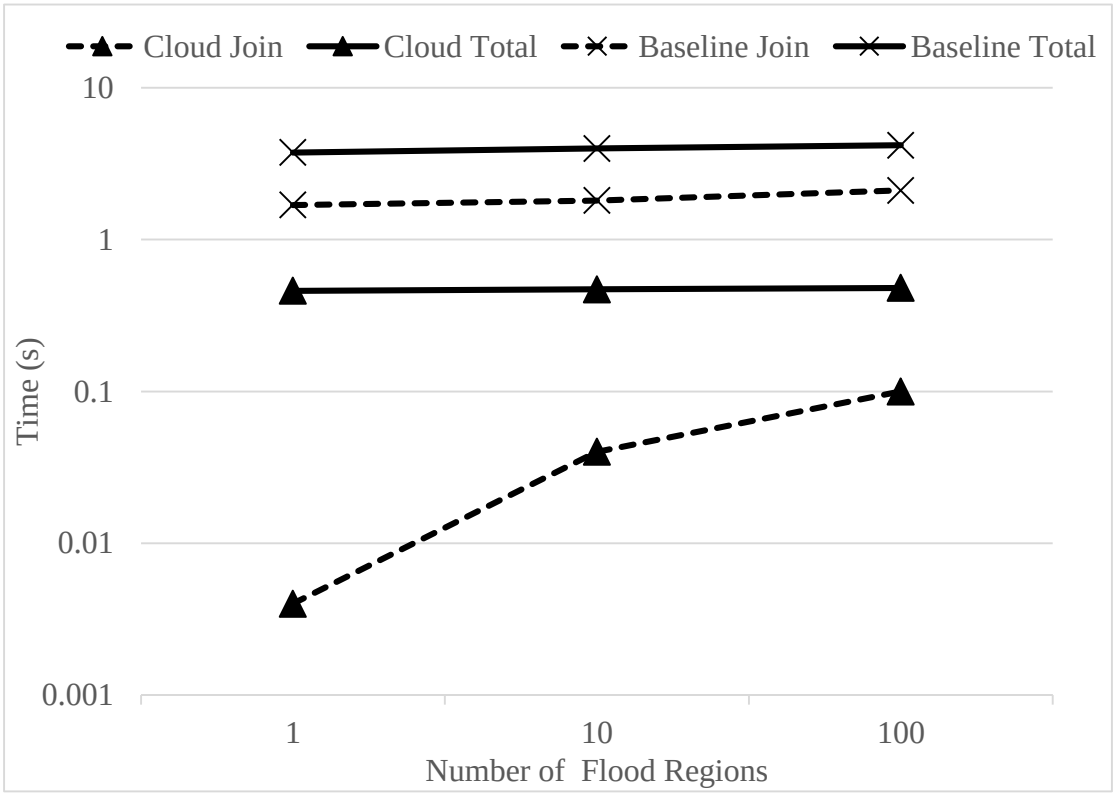


圖 5-8 隨機分佈資料集中改變淹水區域數量之影響

表 5-2　隨機分佈資料集中改變淹水區域數量之影響實驗紀錄

方法(時間 s)	淹水區域個數		
	1	10	100
Baseline Join	3.33	3.34	3.37
Baseline Total	4.16	4.17	4.18
Cloud Join	0.004	0.04	0.1
Cloud Total	3.46	3.47	3.48

(2) 這次實驗與上個實驗的差別僅在於改變淹水區域分佈為高斯分佈的資料集。如圖 5-9，花費的時間與隨機資料集大概相同，達到 100 個淹水區域時，Cloud 的效率為 Baseline 的 3.8 倍。

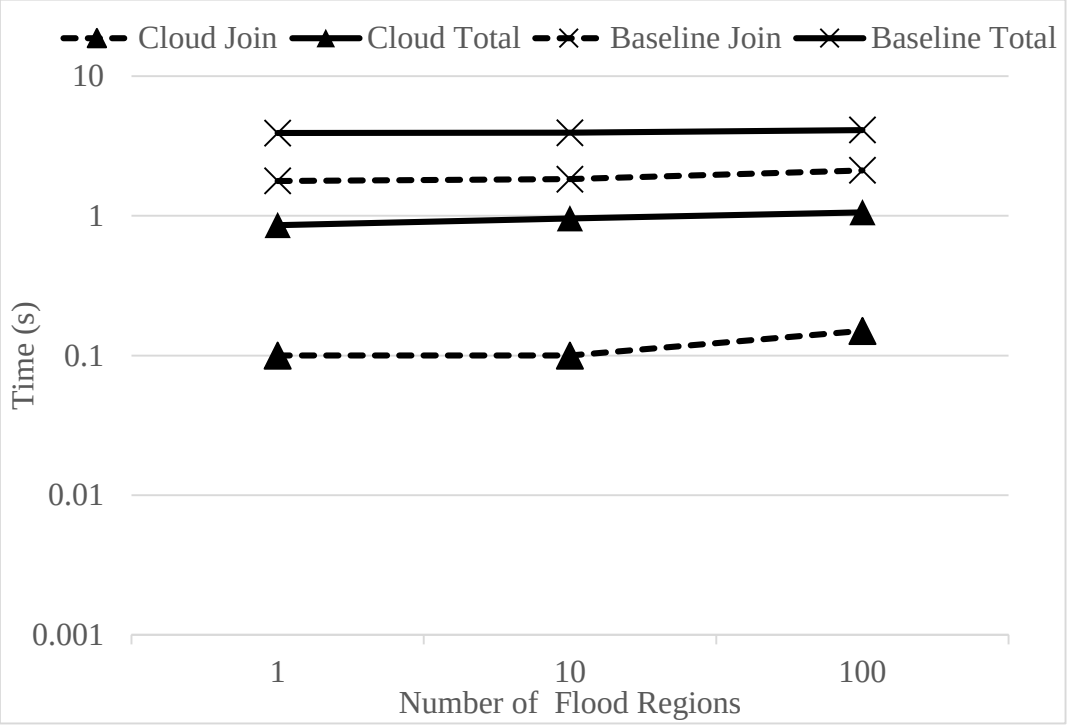


圖 5-9　高斯分佈資料集中改變淹水區域數量之影響

表 5-3　高斯分佈資料集中改變淹水區域數量之影響實驗紀錄

方法(時間)	淹水區域個數		
	1	10	100
Baseline Join	3.33	3.31	3.31
Baseline Total	4.15	4.16	4.25
Cloud Join	0.1	0.1	0.15
Cloud Total	3.86	3.96	4.06

(3) 在這次實驗中，我們將淹水區域範圍限制於兩處，第一處在經度 121.5024~121.5155、緯度 25.0238~25.0313；第二處為經度 121.4867~121.4978、緯度 25.0303~25.0381，然後分別在兩處隨機產生“1”、“10”、“100”個淹水區域。如圖 5-10，“Baseline Spatial”的花費時間會隨著淹水區域的增加而下降，這可能是因為淹水區域分佈密集於兩處，讓 R-tree 樹高降低後，增加了查詢的效率。但達到 100 個淹水區域時，Cloud 的效率為 Baseline 的 4.3 倍，Cloud 仍然優於 Baseline。

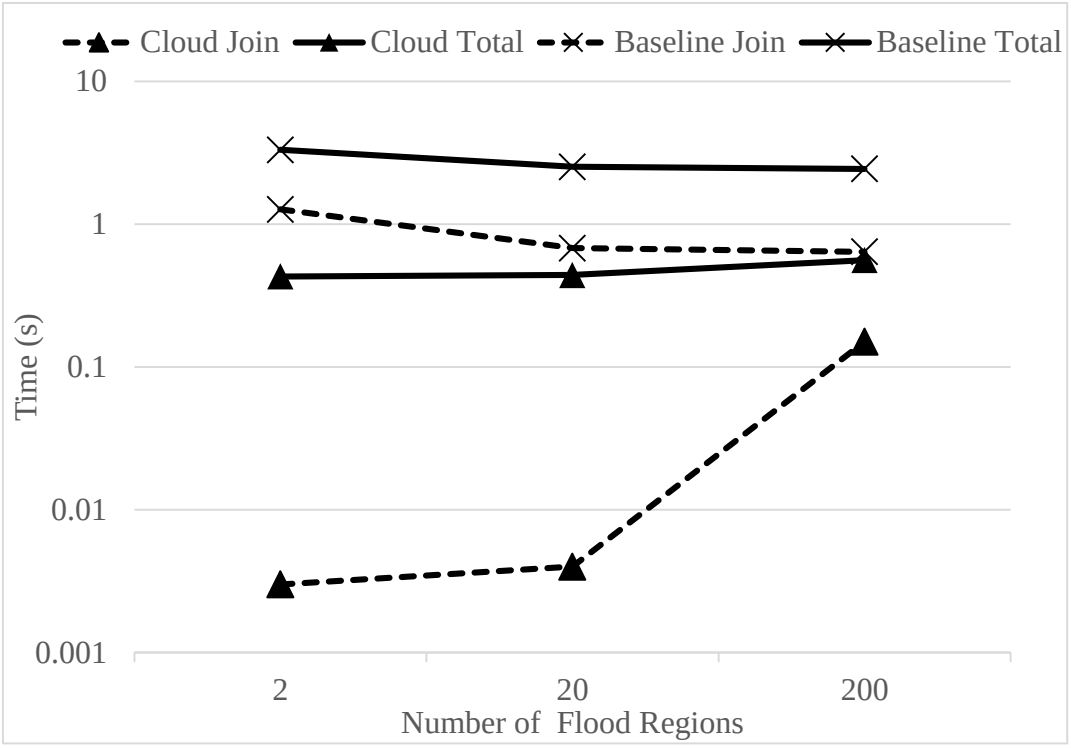


圖 5-10 緊密的隨機分佈資料集中改變淹水區域數量之影響

表 5-4 緊密的隨機分佈資料集中改變淹水區域數量之影響實驗紀錄

方法(時間)	淹水區域個數(各處)		
	1	10	100
Baseline Join	1.269	0.68	0.64
Baseline Total	3.31	2.52	2.44
Cloud Join	0.003	0.004	0.15
Cloud Total	0.43	0.44	0.56

5.4 路網圖數量之實驗

第二個實驗，我們著重於改變路網圖的節點及邊的數量，從台北市的路網圖(7140 個節點及 9314 條邊)分成“100”、“1000”及近似 1 萬條的“9314”條邊，如表 5-5，查詢的路徑起點 V_s 為各路網圖資料集的極東北方節點、目的地 V_t 為極西南方節點，且分別記錄兩方法執行 Spatial Join 的時間與主程式整體的執行時間 (路網圖 input 時間不列入計算)。

表 5-5 各子路網圖查詢之起終點

	100	1000	9314
V_s	25.04374444,121.5301511	25.04374444,121.5301511	25.04374444,121.5301511
V_t	25.03643283,121.4991632	25.04777972,121.5096935	25.02468882,121.4938111

(1) 這次實驗，我們探討改變路網圖的數量之影響，而固定使用隨機資料集中的 10 個淹水區域。如圖 5-11，兩個方法的所有時間皆會隨著路網圖數量增加而上升，這是因為當道路變多時 R-tree 的查詢次數也會增加，進而影響到整體時間。而路網圖數量達到近萬筆時 Cloud 的效率為 Baseline 的 19 倍。

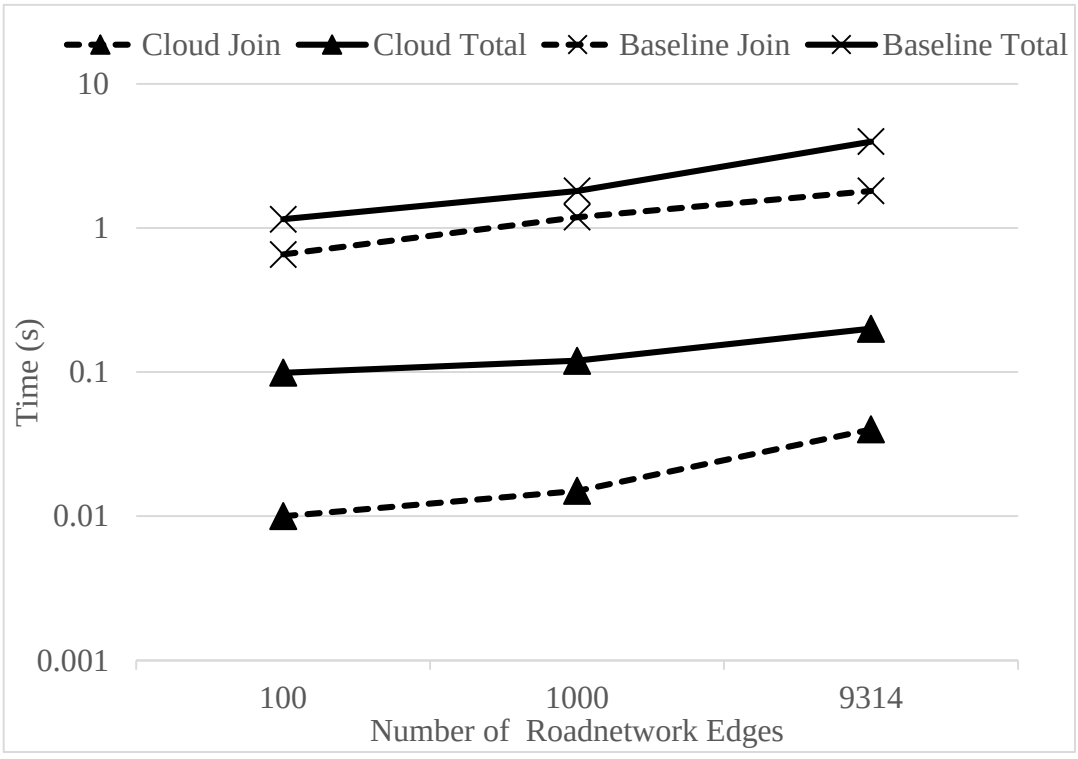


圖 5-11 隨機分佈資料集中改變路網圖數量之影響

表 5-6　隨機分佈資料集中改變路網圖數量之影響實驗記錄

方法(時間)	路網圖邊數		
	100	1000	9314
Baseline Join	0.655	1.19	1.81
Baseline Total	1.15	1.81	3.98
Cloud Join	0.01	0.015	0.04
Cloud Total	0.099	0.12	0.2

(2) 這次實驗相關的參數僅改變淹水區域為高斯分佈的資料集，如圖 5-12，高斯資料集與隨機資料集結果大致相同，而路網圖數量達到近萬筆時 Cloud 的效率為 Baseline 的 6.56 倍。

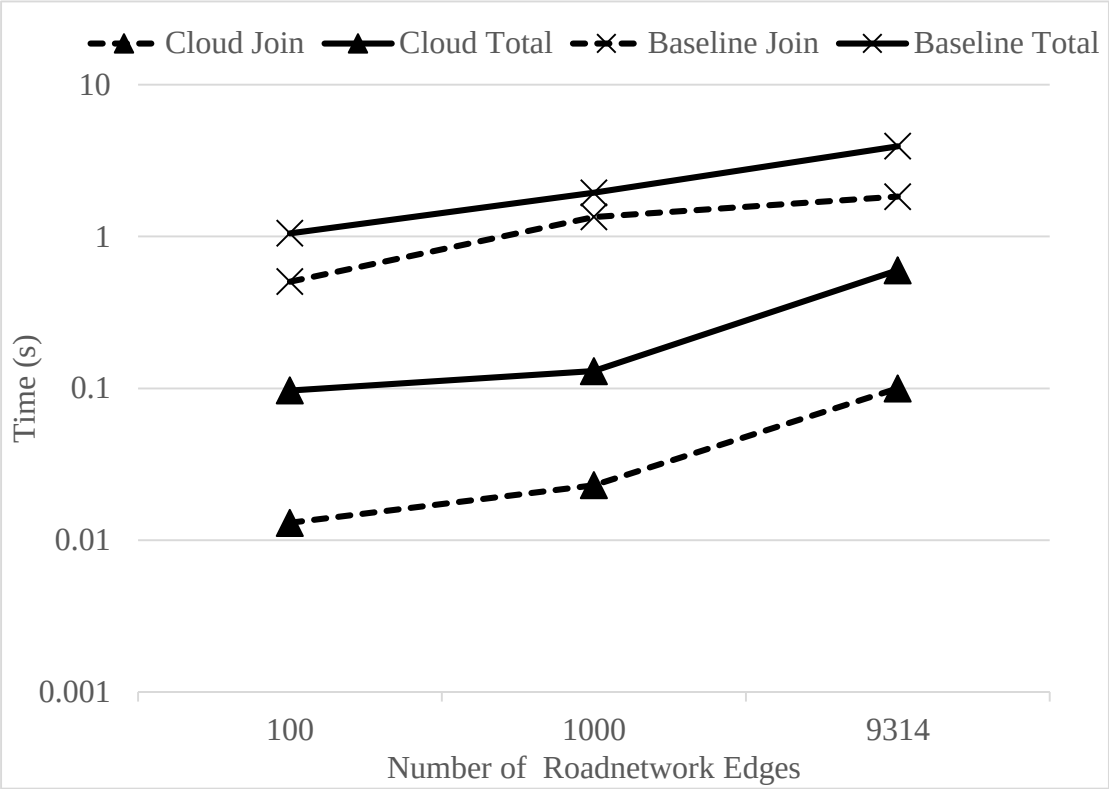


圖 5-12　高斯分佈資料集中改變路網圖數量之影響

表 5-7　高斯分佈資料集中改變路網圖數量之影響實驗紀錄

方法(時間)	路網圖邊數		
	100	1000	9314
Baseline Join	0.655	1.19	1.81
Baseline Total	1.15	1.81	3.98
Cloud Join	0.01	0.015	0.04
Cloud Total	0.099	0.12	0.2

(3) 這次實驗相關的參數同樣僅改變淹水區域為兩大區塊的隨機分佈，如圖 5-13，兩方法的 Spatial 部分花費時間皆約持水平，而“Baseline”整體的花費時間比“Cloud”有明顯增加的現象，可能是路網圖數量增加，而 Dijkstra 找尋路徑的時間也加長，當路網圖數量達到近萬筆時 Cloud 的效率為 Baseline 的 20 倍。

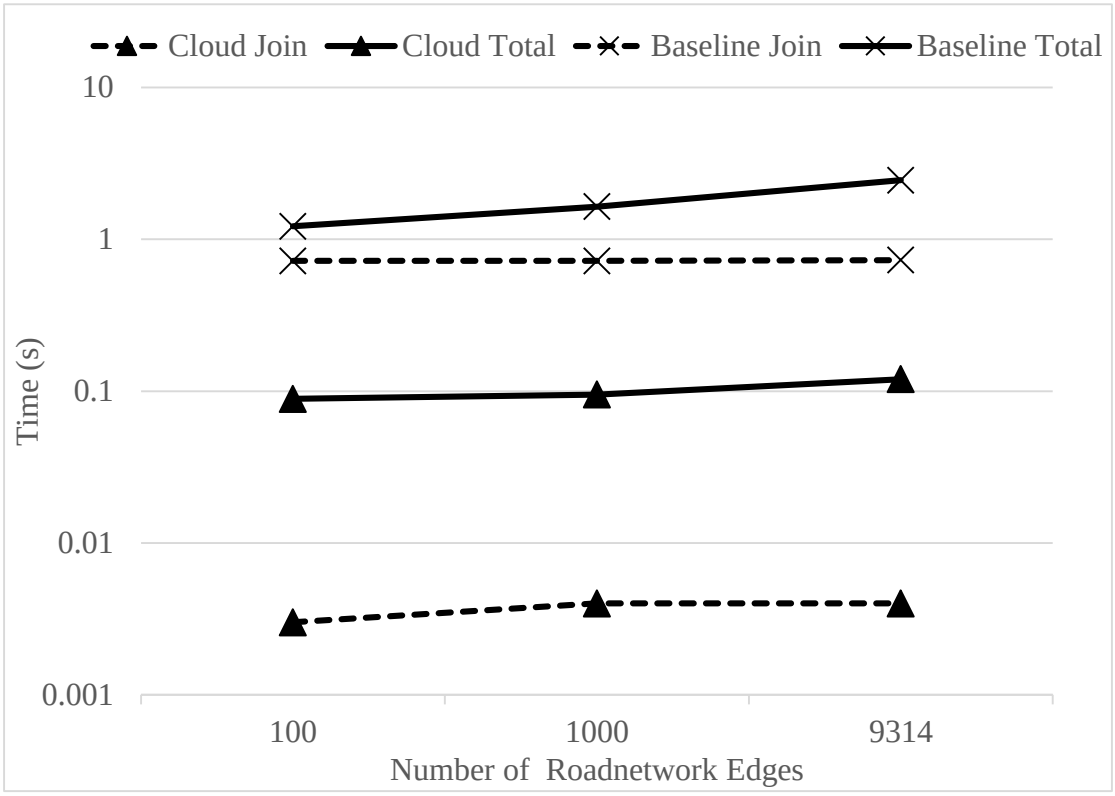


圖 5-13　緊密的隨機分佈資料集中改變路網圖數量之影響

表 5-8　緊密的隨機分佈資料集中改變路網圖數量之影響實驗紀錄

方法(時間)	路網圖邊數		
	100	1000	9314
Baseline Join	0.72	0.72	0.73
Baseline Total	1.218	1.641	2.45
Cloud Join	0.003	0.004	0.004
Cloud Total	0.089	0.095	0.12

5.5 規劃路徑長度之實驗

第三個實驗，我們比較兩方法最後輸出避開淹水區域的路徑長度，固定淹水區域為 10 個、路網為台北市的路網圖(7140 個節點及 9314 條邊)，如表 5-9，隨機選擇 20 個節點(共 10 組)且確定可成功規劃的路徑做為起點、終點。接著，我們將 Baseline 輸出的長度作為基數，再把 Cloud 的長度扣掉 Baseline 所得之路徑長度除上基數後即得到百分比，將百分比繪製成圖便於辨識。

表 5-9 隨機 10 組查詢之記錄表

	Q1	Q2	Q3	Q4	Q5
V_s	25.0283994 1,121.5111	25.02191031, 121.4997082	25.02089355, 121.5251417	25.040537,12 1.5077195	25.02178008,1 21.5203837
V_t	25.0196196 4,121.4960	25.02042815, 121.4946664	25.04781084, 121.5189284	25.02211405, 121.496363	25.03310349,1 21.4981719
	Q6	Q7	Q8	Q9	Q10
V_s	25.0470269 6,121.5107	25.02787782, 121.4904081	25.02528141, 121.4955926	25.03742779, 121.5225984	25.0221503,12 1.4963694
V_t	25.0300828 8,121.4903	25.03210845, 121.4913613	25.03694035, 121.5210355	25.01935385, 121.4951735	25.02196163,1 21.4926285

(1) 這次實驗，10 個淹水區域我們使用隨機分佈的資料集，查詢 10 組的路徑為 Q1~Q10。如圖 5-14 及表 5-10，我們可以發現 Cloud 比 Baseline 輸出的路徑平均長度多 12.35%。

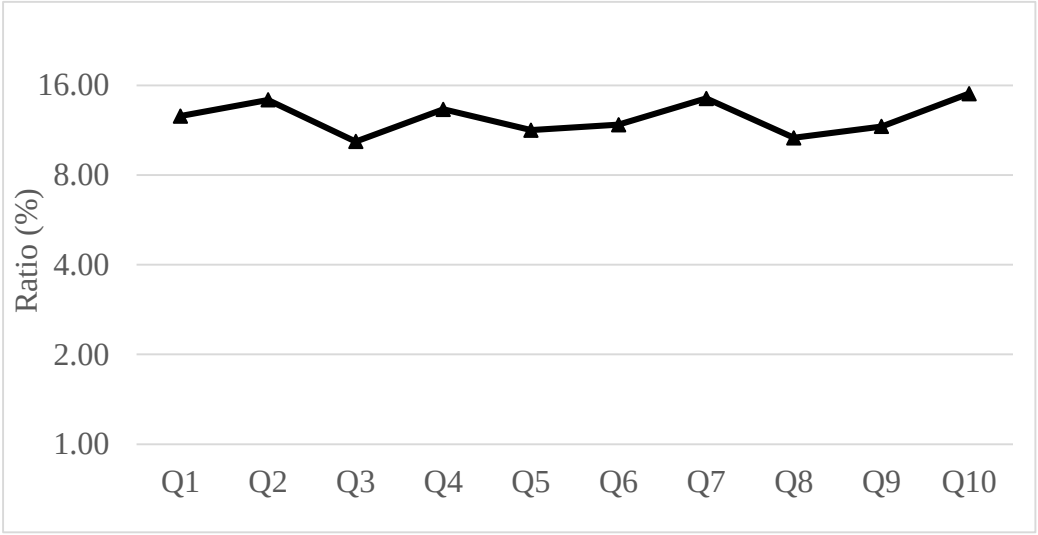


圖 5-14 隨機分佈資料集中規劃路徑長度

表 5-10　隨機分佈資料集中規劃路徑長度實驗紀錄

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Baseline(km)	2.14	0.7	3.763	2.182	2.916	2.881	0.9	3.754	3.778	0.6
Cloud(km)	2.7	1	3.9	2.9	3.3	3.4	1.3	4	4.4	0.9
距離差(km)	0.56	0.3	0.137	0.718	0.384	0.519	0.4	0.246	0.622	0.3
比率(%)	12.62	14.29	10.36	13.29	11.32	11.80	14.44	10.66	11.65	15.00

(2) 這次實驗相關的參數僅改變淹水區域為高斯分佈的資料集，而 Q1~Q10 的節點不變。如圖 5-15 及表 5-11，我們可以發現所規劃的路徑大致與隨機資料分佈結果相同，但 Q8、Q9 因為高斯資料集淹水區域集中的關係，導致鄰近淹水的道路過多所以兩者都要繞路，致使路徑距離皆為增加，如圖 5-16 為 Q8 的範例。Cloud 比 Baseline 輸出的路徑長度平均多 12.94%。

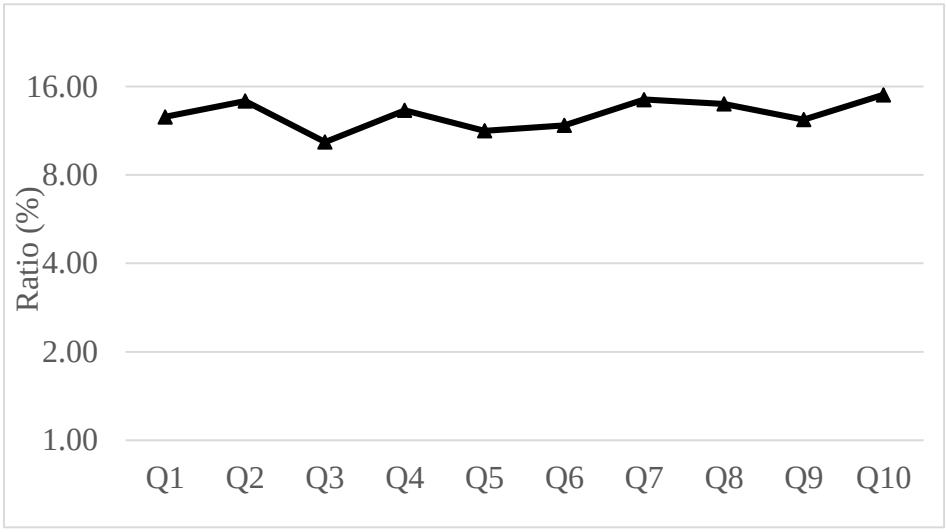


圖 5-15　高斯分佈資料集中規劃路徑長度比較圖

表 5-11　高斯分佈資料集中規劃路徑長度實驗紀錄

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Baseline(km)	2.14	0.7	3.763	2.182	2.916	2.881	0.9	4.154	4.778	0.6
Cloud(km)	2.7	1	3.9	2.9	3.3	3.4	1.3	5.8	5.9	0.9
距離差(km)	0.56	0.3	0.137	0.718	0.384	0.519	0.4	1.646	1.122	0.3
比率(%)	12.62	14.29	10.36	13.29	11.32	11.80	14.44	13.96	12.35	15.00

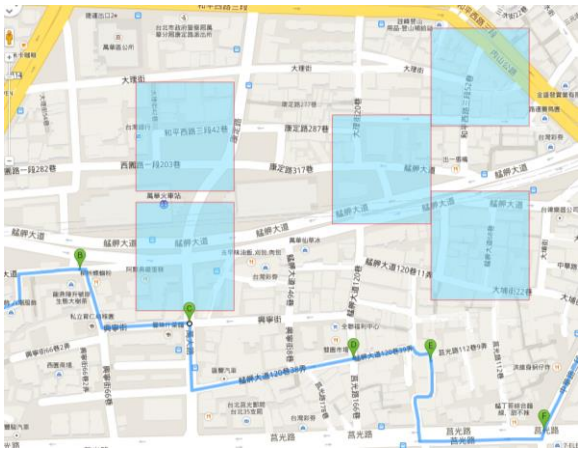


圖 5-16　Q8 的查詢結果

(3) 這次實驗相關的參數同樣僅改變淹水區域為兩大區塊的緊密隨機分佈資料集，而 Q1~Q10 的節點不變。如圖 5-17 及表 5-12，我們可以發現大致與前兩個實驗結果相同，但 Q2 及 Q8 和 Q9 因為淹水區域較為集中的關係，同樣導致鄰近淹水的道路過多所以要繞遠路。而 Cloud 比 Baseline 輸出的路徑平均長度多 13.01%。

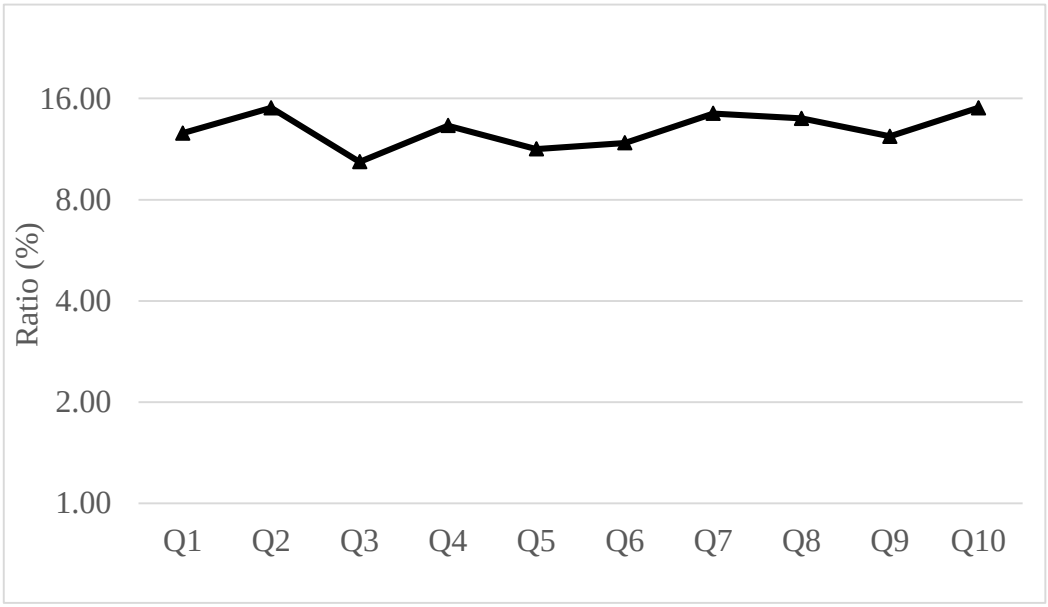


圖 5-17　群集分佈資料集中規劃路徑長度比較圖

表 5-12　群集分佈資料集中規劃路徑長度實驗紀錄

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Baseline(km)	2.14	1.2	3.763	2.182	2.916	2.881	0.9	4.154	4.778	0.6
Cloud(km)	2.7	1.8	3.9	2.9	3.3	3.4	1.3	5.8	5.9	0.9
距離差(km)	0.56	0.6	0.137	0.718	0.384	0.519	0.4	1.646	1.122	0.3
比率(%)	12.62	15.00	10.36	13.29	11.32	11.80	14.44	13.96	12.35	15.00

5.6 規劃路徑成功率之實驗

最後的實驗，我們著重於迴避淹水區域路徑的成功率，實驗中路網採用台北市的路網圖(7140 個節點及 9314 條邊)，淹水區域採用隨機分佈資料集中的 100 個淹水區域，而淹水區域佔路網圖的 $\frac{1}{24}$。接著，隨機選取 40 個節點(共 20 組)作為走訪的起點與終點，並記錄這 20 組執行的結果，若輸出路徑成功避開淹水區域則給 1 分，反之若無法避開淹水路徑或走訪不到終點則給 0 分。

如表 5-13，在這 Q1~Q20 查詢中可以得知 Baseline 無法達到 100%，其原因如圖 5-18 所示，在於道路無實際淹水，但其 MBR 與淹水區域的 MBR 相交，導致可以行走的道路被拿掉。至於 Baseline 還是比 Cloud-1 高出約 15%的成功率，絕大部分情況如圖 5-19 圈選處所示，因為 Cloud-1 方法選出的 Waypoints 再次執行雲端規劃路線時，Google Maps 規劃的路線還是有可能會經過淹水區域的一小部分。我們為了提高 Cloud-1 方法的成功率，所以將 Cloud-1 輸出的路徑作為 Cloud-2 的 Google Maps 規劃的輸入路徑，再次執行 Cloud 方法查詢並觀察成功率是否提升，根據我們實驗結果 Cloud-2 為 60%的確較 Cloud-1 提升了 10%的成功率。

表 5-13　規劃路徑有效性之實驗記錄

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Baseline	1	1	0	1	0	1	1	1	1	1
Cloud-1	1	1	0	1	1	1	0	1	0	1
Cloud-2	1	1	0	1	1	1	0	1	0	1
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Baseline	1	1	0	0	0	0	1	1	1	0
Cloud-1	1	1	0	0	0	0	1	0	0	0
Cloud-2	1	1	0	0	0	0	1	1	1	0
	總　分					成功率				
Baseline	13					65%				
Cloud-1	10					50%				
Cloud-2	12					60%				

圖 5-18　Baseline 方法的誤差範例

圖 5-19　Cloud 方法的誤差範例

5.7 真實資料集之實驗

在本節中，我們使用真實的數據資料集進行相關的實驗，其中資料集包含台北市中正區及萬華區的路網圖(節點：7140、路段：9314)、淹水潛勢資料(130 個淹水區域)。我們先進行兩個方法的『執行效率』及『路徑長度』之實驗，如圖 5-20，執行效率實驗的起點為(25.04374444 121.5301511)、終點為(25.02468882 121.4938111)；路徑長度的查詢起終點同表 5-9 所示。我們可以發現 Cloud 執行效率較 Baseline 高出 2.5 倍，但 Cloud 規劃出的路徑長度卻較 Baseline 多出 12.4%。另外，我們也做了兩個方法的成功率之實驗，也就是在路網圖中隨機挑選出 40 個節點(共 20 組)作為起迄點 (Query 同表 5-13)，並人工判斷各方式輸出的路徑是否完全避開淹水區域，如表 5-14，而在 20 組的查詢結果下 Baseline 的成功率 55%、Cloud 為 45%。與之前的人工資料集相比，成功規劃路徑的比率下降了 10%，原因是因為真實資料的淹水區域過於密集，導致周邊沒有替代道路可行走。

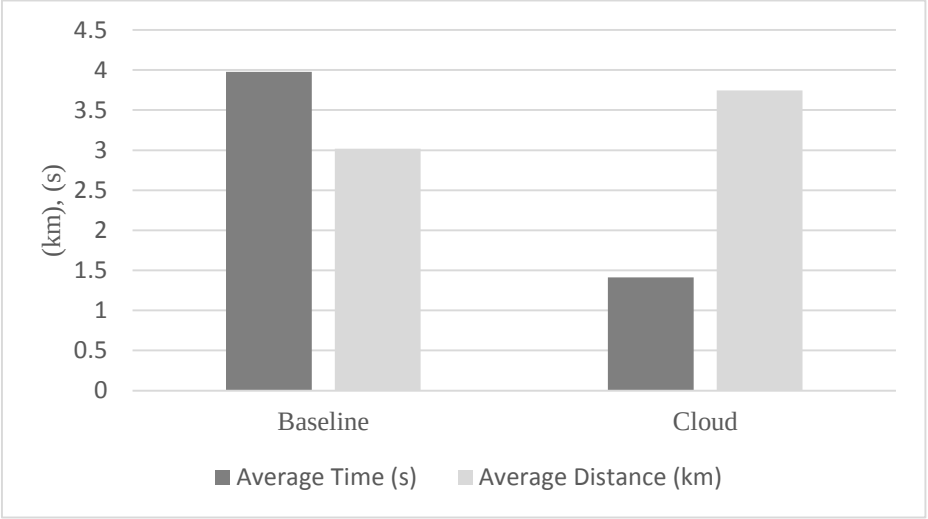


圖 5-20 真實數據資料之執行效率及長度比較圖

表 5-14 真實數據資料之成功率實驗紀錄

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Baseline	1	1	0	1	0	1	1	1	1	1
Cloud	0	1	1	0	0	0	1	1	0	0
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Baseline	0	0	1	0	0	0	1	1	0	0
Cloud	1	0	1	1	0	0	1	1	0	0

	總 分	成功率
Baseline	11	55%
Cloud	9	45%

第 6 章　結論與未來方向

在本論文中，我們提出迴避淹水區域之快速路徑規劃的兩個方法。首先，我們提出了利用 R-tree 將淹水區域與道路作空間連接，進而剔除淹水的道路，再透過 Dijkstra 尋找最短路徑的 Baseline 方法。其次，為了避免因為資料量提升而導致效率下降，我們也提出了 Cloud 方法。在此方法中，我們透過『雲端路徑查詢模組』事先規劃一條最短路徑(GMSP)，再利用 R-tree 將淹水區域與 GMSP 作空間連接，將經過淹水區域的路口點使用『替代節點篩選模組』尋找替代節點集合，最後，利用 GSP-DF 演算法計算起點經過替代節點集合後再到終點的最短路徑。在真實數據資料的實驗下 Cloud 的效率是 Baseline 的 2.5 倍，但 Baseline 輸出長度較 Cloud 少 12.4%，而 Baseline 成功率較 Cloud 高出 10%。我們也進行大量的人工資料實驗來評估兩種方法，包括使用真實路網資料和 3 種不同分佈的人工資料，並分別評估淹水區域數量和路網圖大小的影響。首先，針對效率部分，Cloud 因為需計算交集的次數較少，所以明顯比 Baseline 的效率好，在隨機分佈資料集中達到 100 個淹水區域時 Cloud 的效率更是 Baseline 的 8.7 倍。雖然 Baseline 輸出的路徑長度比 Cloud 少 12%、成功率較 Cloud 高出 15%，但我們也進行 Cloud 的再次規劃實驗，規劃的結果是 Cloud 提升了 10%的成功率。然後，我們也在 Cloud 的顯示網頁內提供手動調整的功能，透過微調來減少長度及提高路徑的可行性。

最後，關於本論文未來的研究方向，希望能夠在空間連接計算的部分提升準確率並同時保持高效率。

參考文獻

[AG84] Antomn Guttman, “R-trees: A dynamic index structure for spatial searching”, Proceedings of the ACM SIGMOD international conference on Management of data,1984.

[BD98] Beman Dawes, David Abrahams, http://www.boost.org/, 1998.

[EW59] Edsger Wybe Dijkstra, “A note on two problems in connexion with graphs, ”Numerische Mathematik, 1:269–271, 1959.

[GB97] Great Britain, “Admiralty manual of navigation”, The Stationery Office, Vol. 1, pp. 10, 1997.

[HNR68] Peter E. Hart, NILS J. Nilsson, Bertram Raphael, “A formal basis for the heuristic determination of minimum cost paths”, IEEE Transactions on Systems Science and Cybernetics, Vol. 4, No 2, pp. 100–107, 1968.

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

[ZLTZ13] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, Lizhu Zhou, “G-Tree: An efficient index for *k*NN search on road networks”, Proceedings of the CIKM conference, 2013.

[張 09] 張傑, “以改良的 A*演算法規劃較佳導引路徑之研究”, 大同大學資訊工程研究所碩士論文, 2009.

[彭 11] 彭祥瑋,“行走機器人避障路徑規劃演算法之配置與實現”, 國立中興大學生物產業機電工程研究所碩士論文, 2011.

[吳 12] 吳佩珊,“基於服務導向架構之洪氾預警系統”, 國立臺灣海洋大學資訊工程研究所碩士論文, 2012.

[吳 13] 吳錫欽, “緊急車輛之監控與迴避引導系統及壅塞路徑重規劃策略”, 國立臺北科技大學電機資訊學院碩士論文, 2013.

[劉 12] 劉欽鴻, “利用改良的雙向 A 演算法實現最佳路徑規劃”, 國立成功大學工程科學研究所碩士論文, 2012.

[劉 13] 劉昱德,“基於地標之淹水警示研究”, 國立臺灣海洋大學資訊工程研究所碩士論文, 2013.

附錄 A 替代節點篩選程式

首先，L2-3 是設定連結資料庫的使用者資訊，L4-9 是接收『替代節點篩選模組』傳送的查詢資料並解析節點的 *x*、*y* 座標，L10-29 是將淹水區域檔案串接成數個多邊形，作為接下來確認是否與道路重疊的依據，L31-33 是先篩選鄰近的路口點，L38-56 是將剛剛篩選的節點再次判斷是否與淹水區域重疊，最後，將沒有與淹水區域重疊的節點輸出於網頁中。

```
1.  <?php
2.  $dsn = "mysql:host=localhost;dbname=floodpath;charset=utf8";
3.  $db = new PDO($dsn,"帳號","密碼");
4.  if(@$_GET['range'] &&@$_GET['point']){
5.    @$range=$_GET['range'];
6.    @$point=$_GET['point'];
7.    $tmp2 = explode(" ",$point);
8.    $point1x =$tmp2[0];
9.    $point1y =$tmp2[1];
10.   $INSQL="POLYGON(";
11.   $i=0;
12.   if(file_exists("FRFile.txt")){
13.         $file = fopen("FRFile.txt", "r");
14.         if($file != NULL){
15.               while (!feof($file)) {
16.                     $Line = fgets($file);
17.                     if($Line!=""){
18.                           $tmp=explode(" ",$Line);
19.                           if($i==0){
20.                                 @$INSQL.="(".$tmp[0]." ".$tmp[1].",".$tmp[2]."
                                    ".$tmp[3].",".$tmp[4]." ".$tmp[5].",".$tmp[6]."
                                    ".$tmp[7].",".$tmp[0]." ".$tmp[1].")";
21.                                 $i=1;
22.                           }else{
23.                                 @$INSQL.=",(".$tmp[0]."
                                    ".$tmp[1].",".$tmp[2]." ".$tmp[3].",".$tmp[4]."
```

```
                                ".$tmp[5].",".$tmp[6]." ".$tmp[7].",".$tmp[0]."
                                ".$tmp[1].")";
24.                         }
25.                     }
26.                 }
27.         }
28.         fclose($file);
29.     }
30.     $INSQL.=")";
31.     $rs = $db->query("select distinct * from `roadnetwork` where
    st_distance(point(`point1x`,`point1y`), point(".$point1x.",".$point1y."))*100
    <=".$range." order by st_distance(point(`point1x`,`point1y`),
    point(".$point1x.",".$point1y."));");
32.     $rs->setFetchMode(PDO::FETCH_ASSOC);
33.     $result_arr = $rs->fetchAll();
34.     if(count($result_arr)==0){
35.         echo "null";
36.         exit();
37.     }
38.     for($i=0;$i<count($result_arr);$i++){
39.         $p1x=$result_arr[$i]['point1x'];
40.         $p2x=$result_arr[$i]['point2x'];
41.         $p1y=$result_arr[$i]['point1y'];
42.         $p2y=$result_arr[$i]['point2y'];
43.         $subsql1="GeomFromText('".$INSQL."')";
44.         $subsql2="GeomFromText('Point(".$p1x." ".$p1y.")')";
45.         $subsql3="MBRWithin(".$subsql1.",".$subsql2.");";
46.         $intersect = $db->query("select ".$subsql3);
47.         $result = $intersect->fetchAll();
48.         $ans=$result[0][0];
49.         if($ans=="0"){
50.             if($p1x && $p2x && $p1y && $p2y){
51.                 if($p1x!=$point1x && $p1y!=$point1y){
```

```
52.                             @$output .=$p1x ." ".$p1y .",".$p2x ." ".$p2y."\n";
53.                         }
54.                 }
55.         }
56.  }
57.  if(@$output!="")
58.         echo $output;
59.  else
60.         echo "null";
61. }
62. ?>
```