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

基於索引技術之淹水區域路徑規劃

The Route Planning in Flooded Areas Based on Indexing Techniques

研究生：洪蔚齊 撰

中華民國 105 年 04 月

基於索引技術之淹水區域路徑規劃

The Route Planning in Flooded Areas Based on Indexing Techniques

研究生：洪蔚齊　　Student：Wei,Chi-Hung

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

April 2016

Keelung, Taiwan, Republic of China

中華民國 105 年 04 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。之前已經有研究提出藉由道路與淹水區域的資訊，規劃出由起點至目的地且能避開淹水的路線，並提出兩種做法。首先，Baseline 作法是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。其次，Cloud 作法是利用雲端服務先行規劃出一條路徑，若該條路徑淹水，再針對淹水路口尋找鄰近點重新規劃。

本論文主要是根據其所提出的架構，利用索引技術，分別提升該二方法的效率。針對 Baseline 做法，我們探討分別利用道路資料建立索引，以及利用淹水區塊建立索引，來判斷道路是否淹水的效率。針對 Cloud 作法，我們利用索引技術尋找淹水路口的鄰近點，並設計兩種不同的索引，第一種是擴充 Baseline 方法所建立的道路索引，另一種則是利用路口點建立索引。

我們實作上述不同的索引，並進行一系列的實驗，比較執行時間以及成功規劃出可行走之路徑的比率。實驗結果顯示，利用索引的 Cloud 作法，效率遠勝過使用索引的 Baseline 做法，且在道路資料量大時，可以達到 10 倍以上。至於成功率則在最多修正 3 次的情況下，可以達到九成以上。

Abstract

The disaster brought by heavy rain has become more and more serious in Taiwan. In the past, a researcher has studied the problem of planning an unflooded path from the given origin to the destination, and proposed two approaches. The first one is called the Baseline approach. It mainly picks out the roads passing through flooded areas, and invokes the Dijkstra algorithm to determine the shortest path based on the remaining unflooded roads. The second proposed approach, called Cloud, utilizes the Google Maps routing planning service to get an initial shortest path. If the path passes through flooded areas, the system will identify nearby alternative roads and re-plan again.

The main goal of this thesis is to improve the efficiency of the two existing approaches by using indexing techniques. For the Baseline approach, we consider the task of determining whether a road is flooded or not, and explore the possibilities of using roads or flooded areas to build indices, respectively. As to the Cloud approach, we consider the task of identifying the nearby alternative roads for a flooded road, and discuss two types of indices. The first one extends the road index constructed for the Baseline approach. The second one uses intersections to build the index.

We have implemented these different indices, and performed a series of experiments to compare their performance. Experimental results show that the Cloud approach with indices is much more efficient than the Baseline approach with indices. The difference may be up to an order of magnitude when the road network is large. Besides, although the Cloud approach cannot always output a path without passing through flooded areas at the first time, it may achieve ninety percent of success rates if it is allowed to adjust its route up to three times.

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，也感謝承翰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。感謝我的同學思彥與學弟妹們，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

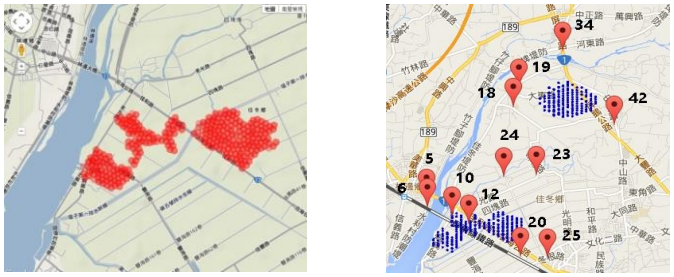
表目錄

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

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。因此本論文探討水災發生時如何規劃一條到災區的搶救路線，而這條路必須盡可能迴避淹水區域且最短，以便迅速抵達救難現場。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。[李 15]延伸此二論文的研究成果，探討如何避開淹水區域到達目的地之路徑規劃的問題，路徑如圖 1-2。



(a) 淹水區域畫面　　(b) 地標警示畫面

圖 1-1　洪氾預警系統

圖 1-2　避開淹水區域之路徑規劃

[李 15]提出 Baseline、Cloud 兩種作法。Baseline 作法是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[EW59]找出最短路徑。此作法的缺點是尋找淹水道路時會先以淹水區塊資料建立索引，但只利用道路 MBR 與索引節點做空間相交，缺少使用道路物件與節點 MBR 做空間相交，造成道路 MBR 與淹水區塊相交但道路本身可能與淹水區塊不相交時，會誤判為淹水道路。

[李 15]提出的 Cloud 作法則是利用雲端路徑規劃服務規劃一條路徑，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鄰近點，然後經由 GSP 演算法，分別從每個淹水路口鄰近點集合選出一個當作替代點，其次將起點、替代點和目地點透過雲端路徑規劃服務規劃出一條最短路徑。

該方法的缺點是鄰近節點篩選模組的作法是先將路網圖建立在資料庫中，然後執行 SQL 查詢，對路網圖所有路口點一一比對，判斷是否與淹水區域重疊以及在所要範圍內，導致效率差。

1.2 研究方法與貢獻

此研究的主要議題，在於如何建立索引以「快速」規劃出一條最短路徑。首先，本論文擴充[李 15]提出的 Baseline 作法，利用淹水區塊建立索引，但新增道路與淹水區塊判斷，稱做 BaseFI。本論文另外提出以道路建立索引，淹水區塊與道路節點、道路物件做空間相交，找出與淹水區塊重疊道路，並從路網中移除，然後將其餘道路由 Dijkstra 演算法規劃出最短路徑，稱做 BaseRI。

另外我們提出 CloudFRI 和 CloudFPI 兩個方法，以改善[李 15]查詢鄰近點的效率。這兩種方法差別在於尋找鄰近鄰近點的方式，一種是利用 BaseRI 道路索引，另一種是利用路網圖路口點資料所建立的索引，希望利用索引的技術，以改善查詢時需之時間。

我們完成此四種方法，並將所得的路徑皆呈現於 Google Maps 上，並針對此四種方法進行廣泛的實驗研究，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]，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。另外加快其計算速度的 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，我們討論有關搜尋鄰近點的論文。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)。論文[CC15]則提出一個群組的 NN 查詢，叫 NNH 查詢，其目的是找到最近的一群點而非一個點。文中針對 NNH 查詢提出一個快速演算法，此演算法是使用 Rtree 結構的鄰近點增量檢索。更具體地說，Rtree 儲存了在一個特定時間範圍內，從查詢點出發所能到達的所有點的座標。當透過增量檢索下一個離查詢點最近鄰近點時，他們會維持一個結構來持續追蹤到目前為止的群集，從而檢查這些群集是否可以由給定大小的圓覆蓋，再透過最小包圍圈的特性，在考慮所有群集下設計出有效的更新方法進行維護。

論文[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。至於本論文會用到論文[BKSS90]提出的 R*樹是基於減少所查詢的目錄矩形的面積、邊界及重疊。分裂方法是利用三種方法來找出最佳分裂軸，而輸入為遞增排序後分為兩群。方法一為面積值（Area-Value），是由第一群包圍矩形的面積加上第二群包圍矩形的面積，方法二為邊值（Margin-Value），是由第一群包圍矩形的邊長加上第二群包圍矩形的邊長，方法三為重疊值（Overlap-Value），是取第一群包圍矩形與第二群包圍矩形的交集後的面積。實驗顯示，R*-tree 在矩形和多維度資料的點資料方面都有最好的表現。

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。第三章介紹基於 Dijkstra 之方法，也就是 BaseRI 和 BaseFI，說明其整體架構，以及如何利用索引找出迴避淹水區域的最短路徑。第四章介紹基於雲端服務之方法，也就是 CloudFRI 和 CloudFPI，說明其整體架構，以及如何利用索引快速找出迴避淹水區域的路徑。於第五章中以實驗比較 Dijkstra 方法與雲端服務方法的效率、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，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

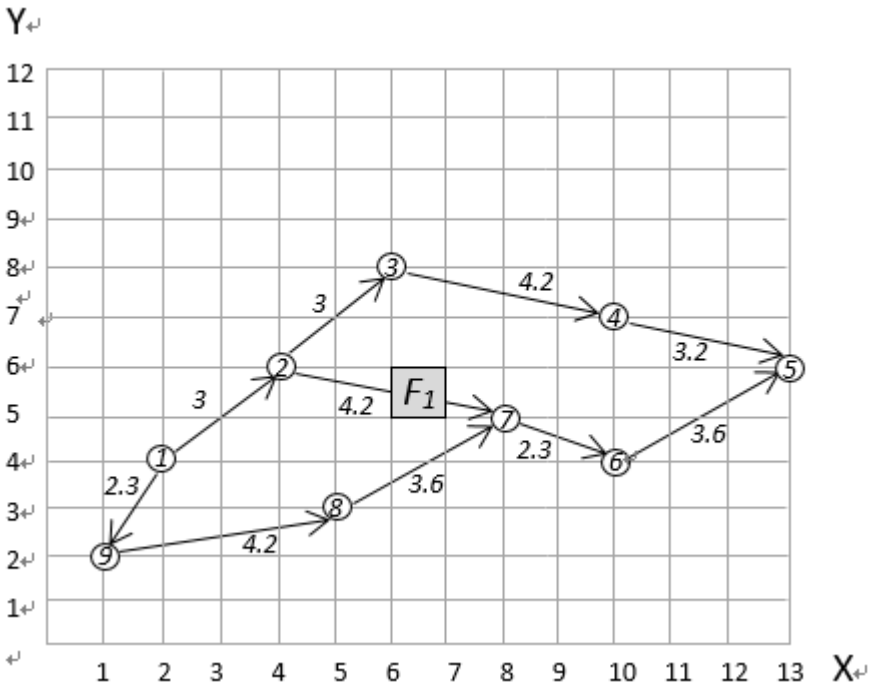


圖 2-1 路網圖

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，以白色圓圈表示，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，每個編號代表一組經緯度座標。每個邊 $e \in E$ 表示成 $R_{i,j}$，代表從 V_i 走到 V_j 的道路。每個權重 $w \in W$ 則代表兩個路口點的矩離，在範例中，距離是由 $R_{i,j}$ 道路 MBR 的長度平方加寬度平方開根號，每個淹水區域以淺灰色的矩形方塊表示，如圖 2-1。在本論文中我們假設 V_1 作為起始點 V_s，V_5 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 $P_{i,j}$，表示方式為$<V_i,..,V_j>$。

[定義 2-2] MBR：全名為 minimum bounding rectangle，表示涵蓋物件的最小邊界矩形，在後續的論文中有時會以該矩形的左下(x_min,y_min)和右上(x_max,y_max)兩點表示，其中 x_min 為矩形範圍內最小 x，y_min 為最小 y，x_max 為最大 x，y_max 為最大 y。若以 $R_{i,j}$ 為物件，涵蓋該 Road 的矩形定義為 Road 的 MBR，以 $M_{i,j}$ 表示。

舉例來說，考慮道路 $R_{1,9}$，其 $M_{1,9}$ 為(V_9.x V_9.y)和(V_1.x V_1.y)，如圖 2-2(a)。而淹水區塊 F_1 本身已經是 MBR，因此直接用淹水區塊代表，如圖 2-2(b)。

(a) 道路 MBR 範例　　(b) 淹水 MBR 範例

圖 2-2 MBR 範例

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 $P_{1,5}$，在不考慮淹水區塊情況下，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 $D_{1,5}$ 為最小的值 13.1，而對應的路徑即為最短路徑，也就是 $P_{1,5} = <V_1,V_2,V_7,V_6,V_5>$。

表 2-1　SSP 範例走訪表

組合	路徑	距離
1	$<V_1,V_2,V_3,V_4,V_5>$	13.4
2	$<V_1,V_2,V_7,V_6,V_5>$	13.1
3	$<V_1,V_9,V_8,V_7,V_6,V_5>$	16

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 $G(V, E, W)$ 路網圖、起點和目的地，並對路口點做分類，得到種類序列 $C=\{C_1、....、C_k\}$，而 $C_i=\{C_{i,1}, C_{i,2},...., C_{i,|Ci|}\}\subseteq V$，此問題是從起點走到目的地，途中依序從每一個種類 C_i 選取一點節點走訪，找出所有可能路徑中具有最短長度的路徑。

注意到，原始定義裡的路徑是根據道路而得，但為了後續的實作限制，我們會直接取兩點間的距離，也就是在本論文範例中，兩點間的距離為其形成 MBR 的長度平方加高度平方開根號，但實作時，則利用大圓距離公式來計算。

[範例 2-1] 假設如圖 2-1 路網圖所示，將起迄點之外的點編入 3 個種類 (Category)，也就是 $C_1=\{V_2,V_9\}$、$C_2=\{V_3,V_7,V_8\}$、$C_3=\{V_4,V_6\}$，如圖 2-3，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、C_3，且路徑也要是最短距離。如表 2-2 我們在不考慮淹水情況下列出所有路徑，其中 $C_{i,j}$ 為第 i 個種類，第 j 個點。而組合 5 的距離為 V_1 至 V_2 的直線距離(3)、V_2 至 V_8 的直線距離(3.2)、V_8 至 V_4 的直線距離(6.4)以及 V_4 至 V_5 的直線距離(3.2)相加得 15.8。從表 2-2 中看出 $D_{1,5}$ 為最小的值 13.1，其對應的路徑即為最短路徑 $P_{1,5} = <V_1,C_{1,1},C_{2,2},C_{3,2},V_5>$，而對應節點後即 $<V_1,V_2,V_7,V_6,V_5>$。

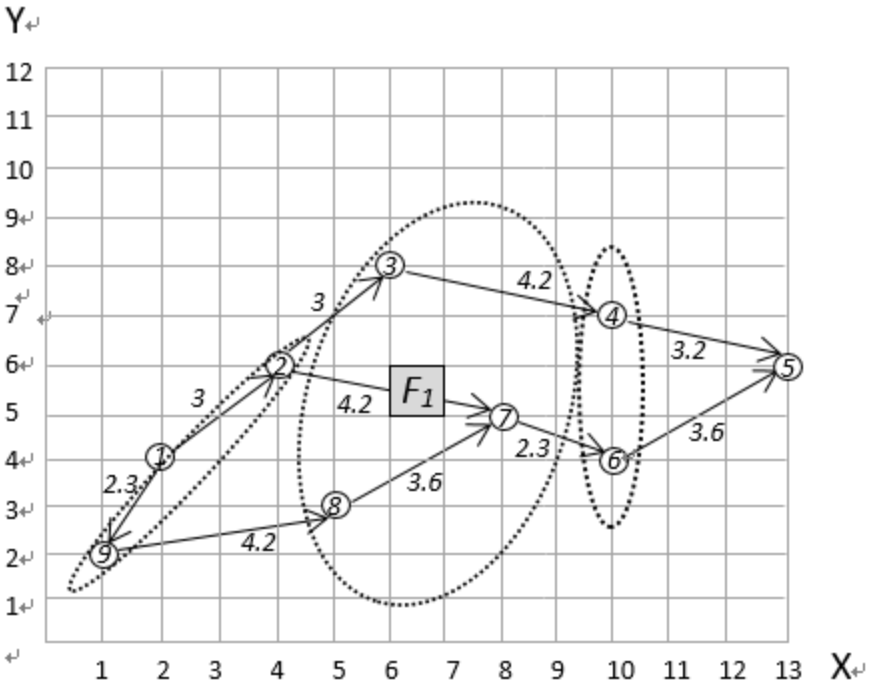


圖 2-3 將路口點分類

表 2-2 GSP 範例走訪表

組合	路徑	距離
1	$<(V_1),C_{1,1}(V_2),C_{2,1}(V_3),C_{3,1}(V_4),(V_5)>$	13.4
2	$<(V_1),C_{1,1}(V_2),C_{2,1}(V_3),C_{3,2}(V_6),(V_5)>$	15.2
3	$<(V_1),C_{1,1}(V_2),C_{2,2}(V_7),C_{3,1}(V_4),(V_5)>$	13.4
4	$<(V_1),C_{1,1}(V_2),C_{2,2}(V_7),C_{3,2}(V_6),(V_5)>$	13.1
5	$<(V_1),C_{1,1}(V_2),C_{2,3}(V_8),C_{3,1}(V_4),(V_5)>$	15.8
6	$<(V_1),C_{1,1}(V_2),C_{2,3}(V_8),C_{3,2}(V_6),(V_5)>$	14.9
7	$<(V_1),C_{1,2}(V_9),C_{2,1}(V_3),C_{3,1}(V_4),(V_5)>$	17.5
8	$<(V_1),C_{1,2}(V_9),C_{2,1}(V_3),C_{3,2}(V_6),(V_5)>$	19.4
9	$<(V_1),C_{1,2}(V_9),C_{2,2}(V_7),C_{3,1}(V_4),(V_5)>$	16.1
10	$<(V_1),C_{1,2}(V_9),C_{2,2}(V_7),C_{3,2}(V_6),(V_5)>$	15.8
11	$<(V_1),C_{1,2}(V_9),C_{2,3}(V_8),C_{3,1}(V_4),(V_5)>$	16.1
12	$<(V_1),C_{1,2}(V_9),C_{2,3}(V_8),C_{3,2}(V_6),(V_5)>$	15.2

最後本論文提出問題，規劃一條迴避淹水區域的最短路徑，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。

[範例 2-2] 我們從圖 2-1 中去掉淹水道路，針對此路網圖查詢避開淹水區域後 V_1 至 V_5 的最短路徑，如表 2-3，我們列出所有避開淹水區域的路徑，其中 $D_{1,5}$ 為最小的值 13.4，而對應的路徑即為最短路徑 $P_{1,5}$ = $<V_1,V_2,V_3,V_4,V_5>$。

表 2-3 USP 範例走訪表

組合	路徑	距離
1	$<V_1,V_2,V_3,V_4,V_5>$	13.4
3	$<V_1,V_9,V_8,V_7,V_6,V_5>$	16

2.2 問題假設

在本論文中，我們假設路網圖是由道路與路口組成。由於 CloudFRI 和 CloudFPI 方法計算點與點之間距離需利用大圓距離公式，因此介紹點 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:

$$a = \sin^2(\Delta\varphi/2) + \cos\lambda_1 \cdot \cos\lambda_2 \cdot \sin^2(\Delta\lambda/2)$$
$$c = 2 \cdot \operatorname{atan2}(\sqrt{a},\ \sqrt{1-a}\,)$$
$$\text{distance} = R \cdot c$$

λ_1、λ_2 為 V_1 與 V_2 的緯度，$\Delta\varphi$、$\Delta\lambda$ 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 基於 Dijkstra 之方法

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。配合不同索引的建置方式，又細分為 BaseRI 和 BaseFI 兩種方法

3.1 資料結構

本節首先介紹系統中幾個比較重要資料結構，此方法需輸入道路資料和淹水資料，程式中的道路資料結構名稱，包含 PointList、weight、Intersect 以及 MBR 等屬性，其中 PointList 代表道路兩個端點和轉折點[1]，weight 代表兩點間的距離，Intersect 代表是否與淹水區塊相交，值為 true 代表與淹水區塊重疊，false 表示不與淹水重疊，MBR 表示涵蓋道路的最小邊界矩形。根據圖 2-1 的路網圖，將每條道路上的點(一組經緯度座標)push 至屬性 **PointList**，路的距離輸出至 **weight**，計算每條道路最小邊界矩形並將結果輸出至屬性 **MBR**。在表 3-1 中，列舉部分道路作為範例。至於程式中的淹水區域資料結構名稱，包含 MBR 屬性，表示涵蓋淹水區塊的最小邊界矩形。

由於判斷道路與淹水區域是否重疊時，需建立索引，目的是加快查詢速度，因此以下先介紹如何以道路為物件建立索引，簡稱道路索引(RoadIndex)。道路索引資料結構定義了 Root、RoadPtr 以及 order 三個屬性，order 為一個節點所能存放的物件個數上限。RoadPtr 為指向所有道路指標，目的是查詢非淹水道路時，scan 所有道路，找出屬性 Intersect 為 false 的道路，而該道路不經過淹水區域。Root 為 entry 型態，記錄根節點 MBR 和指向根節點指標。另外，Leaf 節點定義 entry 屬性集合，每個 entry 記錄道路 mbr(屬性為 MBR)以及指向道路指標(屬性 RoadPtr);Non-Leaf 節點定義 entry 屬性集合，每一個 entry 表示小孩節點 mbr(屬性為 MBR)和小孩指標(屬性為 Ptr)。涵蓋節點 MBR 以 Mi 表示。一個 RTree 的結

[1]轉折點表示道路轉彎的地方，為了簡化討論，我們在本章不列道路的轉折點。

構如圖 3-1 所示。

我們使用圖 2-1 路網說明如何建立 RTree，假設 order 值上限為 3，下限為 2，表示每個節點最多表示 2-3 個物件。首先假設道路順序為 $R_{9,8}$、$R_{1,2}$、$R_{1,9}$、$R_{2,7}$、$R_{8,7}$、$R_{2,3}$、$R_{7,6}$、$R_{3,4}$、$R_{4,5}$ 和 $R_{6,5}$，對應 MBR 右上點為(5,3)、(4,6)、(2,4)、(8,6)、(8,5)、(6,8)、(10,5)、(10,8)、(13,7)和(13,6)，接著按照道路 MBR 右上點 Y 軸由小排到大，如果 Y 軸相同，則按照 X 軸，結果為 $R_{9,8}$、$R_{1,9}$、$R_{8,7}$、$R_{7,6}$、$R_{1,2}$、$R_{2,7}$、$R_{6,5}$、$R_{4,5}$、$R_{2,3}$、$R_{3,4}$，然後按照排序的結果將道路新增至節點，一開始先建立 Leaf 節點 N_1，新增 $M_{9,8}$、$M_{1,9}$ 和 $M_{8,7}$ 以及指向$<V_9,V_8>$、$<V_1,V_9>$和$<V_8,V_7>$道路指標，接著建立 Leaf 節點 N_2，新增 $M_{7,6}$、$M_{1,2}$ 和 $M_{2,7}$ 以及指向$<V_7,V_6>$、$<V_1,V_2>$和$<V_2,V_7>$道路指標，建立 Leaf 節點 N_3，新增 $M_{6,5}$ 和 $M_{4,5}$ 以及指向$<V_6,V_5>$、$<V_4,V_5>$道路指標，建立 Leaf 節點 N_4，新增 $M_{2,3}$ 和 $M_{3,4}$ 以及指向$<V_2,V_3>$、$<V_3,V_4>$道路指標，接著往上層建立 Non-Leaf 節點 N_5，新增 N_1、N_2 節點 MBR 以及指向 N_1、N_2 指標，Non-Leaf 節點 N_6，新增 N_3、N_4 節點以及指向 N_3、N_4 指標，持續往上層建立 Root 節點 N_7，新增 N_5、N_6 節點以及指向 N_5、N_6 指標，形成的道路索引(RoadIndex)，如圖 3-2、圖 3-3 所示，其中圖 3-2 中的 M_1 表示 N_1 節點的 MBR。另一個方法為以淹水區塊建立索引，稱作淹水區塊索引(FloodIndex)。由於其概念類似，我們就省略其介紹。

表 3-1　道路結構範例

PointList		weight	MBR
V_1	V_2	1	$V_{1.x}$ $V_{1.y}$ $V_{2.x}$ $V_{2.y}$
V_1	V_9	1	$V_{9.x}$ $V_{9.y}$ $V_{1.x}$ $V_{1.y}$

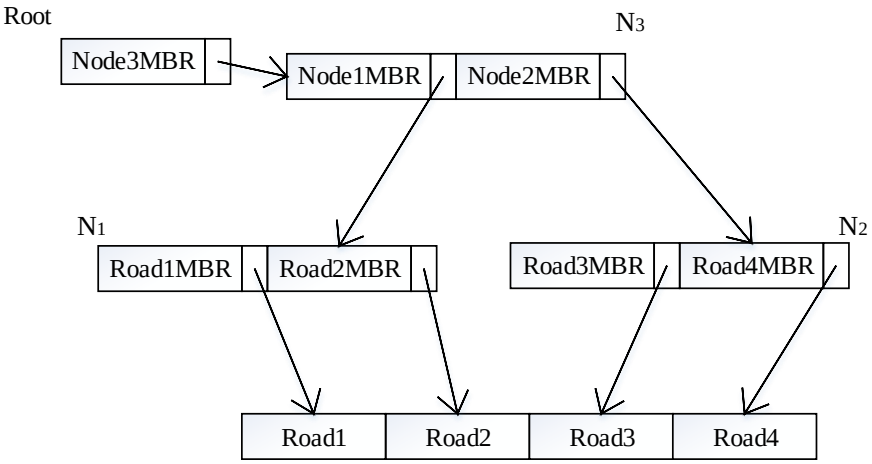


圖 3-1　RoadIndex 示意圖

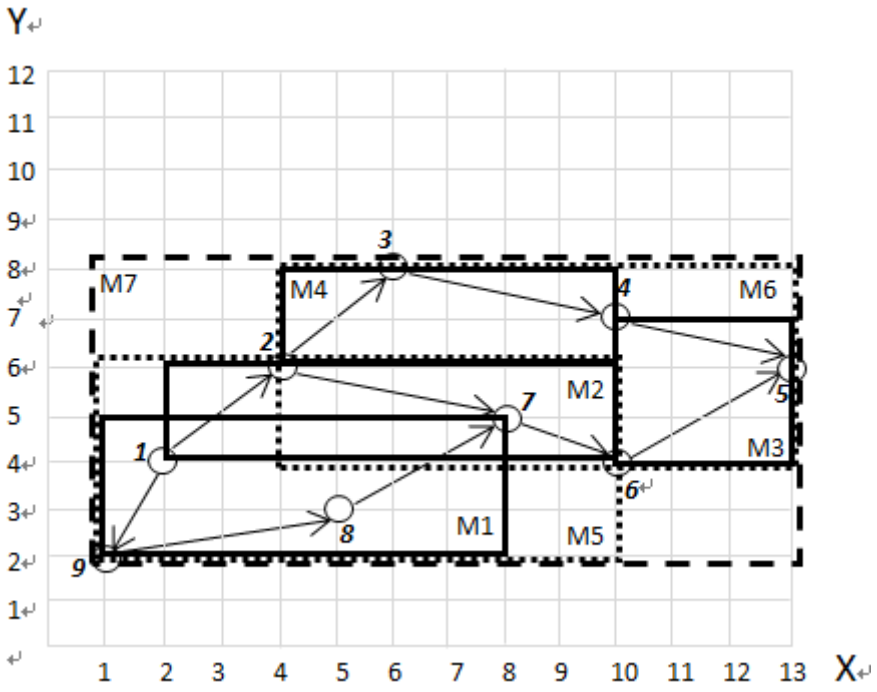


圖 3-2　路網圖及其 MBR

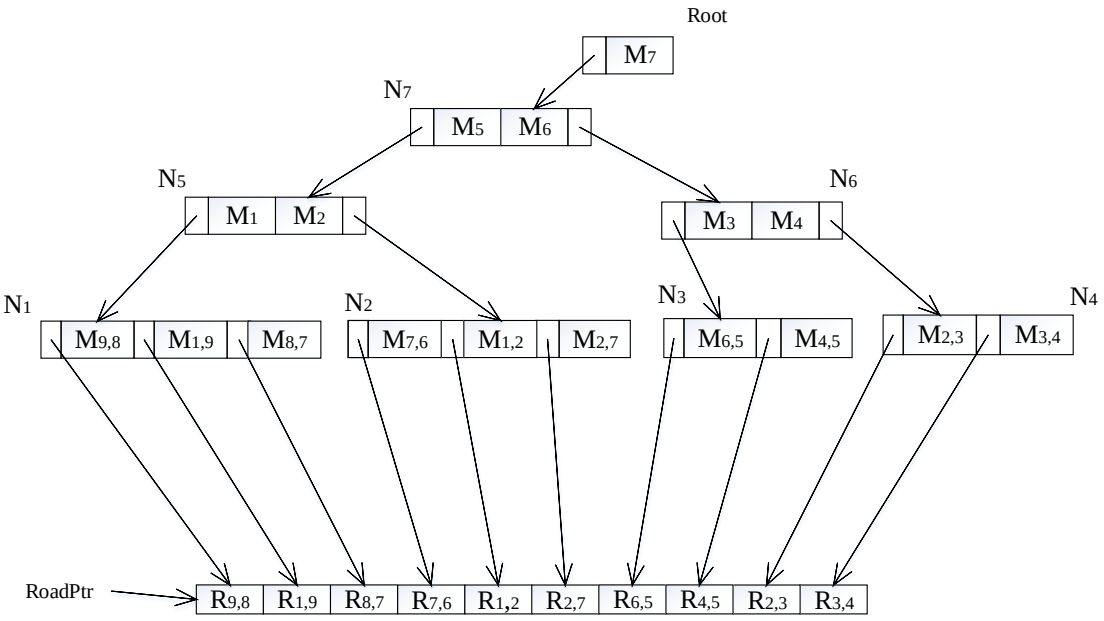


圖 3-3　道路索引範例

3.2 基於道路索引的演算法

此作法名稱為 BaseRI，也就是利用道路資料建立索引，然後利用淹水區塊走訪此索引，以判斷與哪些路重疊。好處是可預先建立，因為路的資料不會有太大變化，然後在 Online 階段直接查詢，減少 Online 階段所花時間。缺點是通常道路數目比較多，因此建立時間較長，而建立過程中可能發生 MBR overlap 情形，影響查詢時間。以下分節說明事前建立索引的演算法，如圖 3-4，online 查詢的演算法，如圖 3-5、圖 3-6。

3.2.1 Preprocessing 階段

演算法名稱：**BuildRoadIndex**

輸入：RoadFile, order //order 表示分割個數

變數：RoadSet

輸出：RoadIndex //表示道路索引結構

L01　RoadSet←ReadRoad(RoadFile)
L02　RoadIndex←BuildIndex(RoadSet, order)
L03　**return** RoadIndex

圖 3-4　BuildRoadIndex 演算法

首先，L01 是讀取道路資訊，並對每條道路產生 MBR，而一開始 Intersect 皆為 false。L02 是根據道路 MBR 建立索引，L03 輸出道路索引。BuildIndex 有三種作法，Sorted、Split 以及 Hilbert，其中 Sorted 和 Hilbert 索引皆將物件排序，然後按順序新增至節點，當節點個數超過 order 時就產生新的節點新增物件，直到所有物件新增至節點，差別在於排序方式不同。Sorted 按照物件 MBR 的右上點排序，先排序 Y 軸，如果 Y 軸值相同則排序 X 軸，如圖 3-2 和 3-3 的建立方式。Hilbert 則會根據道路 MBR 中心點計算出 Hilbert Value 並排序。Split 則採用 R*-tree 方式建立索引，也就是根據 MBR 的右上點 x、y 值排序，會先選取右上點，依照 x 軸或 y 軸來分割，且分割基準是 MBR 周邊長和最小的兩個區塊，演算法詳見附錄 A。

3.2.2　Online 階段

在此階段查詢事先建好的道路索引，輸入淹水區塊，並找出沒淹水的道路，接著做 Dijkstra 路徑規劃，並輸出最短路徑。

演算法名稱：**BaseRIMain**
輸入：FloodFile, RoadIndex, Vs, Vt　//Vs 表示起點，Vt 表示終點
變數：FloodSet
NoFloodRoad //記錄未淹水道路集合
輸出：*Ps,t* //輸出點 s 到點 t 最短路徑
L01　FloodSet←ReadFlood(FloodFile)
L02　**Initialize** NoFloodRoad←NULL
L03　**for** (each Flood **f** of FloodSet)

```
L04      SearchRoadIndex (RoadIndex.Root, f)
L05   End for
L06   NoFloodRoad ← OutputNoFloodRoad(RoadIndex.RoadPtr)
L07   Ps,t←Dijkstra(Vs, Vt, NoFloodRoad)
L08   return Ps,t
```

圖 3-5 BaseRIMain 演算法

首先，L01 是讀取淹水資訊，並對每塊淹水區域產生 MBR，目的判斷是否與道路物件矩形相交。L02 初始 NoFloodRoad 變數為空。L03-L05 是查詢道路索引，並指定道路屬性 Intersect。L06 scan 所有道路，找出屬性 Intersect 為 false 的道路，並輸出至 NoFloodRoad 集合，此處的參數 RoadIndex.RoadPtr 會指向索引最底層的全部道路。L07 是輸入起終點做路徑規劃，找出最短路徑，L08 輸出最短路徑。

```
演算法名稱：SearchRoadIndex
輸入：entry //表示節點Entry屬性
      flood //flood表示淹水區塊，判斷道路是否與此區塊相交
L01   If (entry.MBR intersects flood)
L02      If (entry.Ptr n is a leaf)
L03         for (each road r of n)
L04            If(r.MBR intersect flood && r intersect flood)
L05               r.Intersect ← true
L06            End If
L07         End for
L08      Else If (entry.Ptr n is not a leaf)
L09         for (each entry e of n)
L10            SearchRoadIndex(e, flood)
L11         End for
L12      End If
L13   End If
```

圖 3-6 SearchRoadIndex 演算法

首先，L01 判斷淹水區塊是否與 RoadIndex 節點相交，L03-L07 先判斷淹水區塊是否與 leaf 節點每條道路 MBR 相交，如果與道路 MBR 沒有相交，那麼淹水區塊不與道路本身相交，如果與道路 MBR 相交，需進一步判斷淹水區塊是否與道路本身相交，並將與淹水區塊重疊道路的 Intersect 屬性設成 true。注意到，此處的道路包含轉折點，所以會一段一段進行判斷。然後，L08-L12 判斷淹水是否與 non-leaf 節點 MBR 相交，如果相交就判斷其所有小孩節點是否相交，直到該節點為 leaf 節點。

以下我們針對圖 2-1，給定起點 V_s 為 V_1、目的地 V_t 為 V_5，解釋如何使用 Baseline 方法，取得 $P_{s,t}$ 路徑。

首先在 **Preprocessing** 階段利用 **RoadRTree** 演算法將輸入道路資料建立 MBR(L01)，利用此 MBR 建立 R-tree(L02)，如圖 3-3。接著在 **Online** 階段，我們使用 **BaseRIMain** 演算法讀取淹水檔案產生淹水區塊(L01)，並輸入淹水區塊來查詢這棵 R-tree，將沒有查詢到重疊的道路輸出(L03-06)，如表 3-2。然後建立 AdjacencyList，並利用 Dijkstra 演算法規畫路徑(L07)，最後回傳最短路徑 Ps,t，結果如表 3-3 所示。

表 3-2　無淹水路段的 Adjacency List

V_i	V_j	$W_{i,j}$
V_1	V_2	3
	V_9	2.3
V_2	V_3	3
V_3	V_4	4.2
V_4	V_5	3.2
V_6	V_5	3.6
V_7	V_6	2.3
V_8	V_7	3.6
V_9	V_8	4.2

表 3-3　Dijkstra 規劃路徑

V_i	V_j	$W_{i,j}$
V_1	V_2	3
V_2	V_3	3
V_3	V_4	4.2
V_4	V_5	3.2

3.3 基於淹水區塊索引的演算法

此作法名稱為 BaseFI。也就是利用淹水區塊建立索引，以道路查詢，好處是其數目較少，可以加快建立時間，而建立過程中 overlap 程度不嚴重，因此查詢速度也比道路索引還快，不過需在 online 階段建立，增加 online 階段時間。演算法如圖 3-7、圖 3-8。

演算法名稱：**BaseFIMain** 輸入：order, RoadFile, FloodFile, Vs, Vt 變數：RoadSet, FloodSet, NoFloodRoad FloodIndex //表示淹水區塊索引 輸出：*Ps,t*
L01 FloodSet←ReadFlood(FloodFile) L02 RoadSet←ReadRoad(RoadFile) L03 FloodIndex←BuildFloodIndex(FloodSet, order) L04 **for** (each road **r** of RoadSet) L05 **If**(**not SearchFloodIndex**(Root, **r**)) L06 NoFloodRoad.push(r) L07 **End If** L08 **End for** L09 *Ps,t*←Dijkstra(Vs, Vt, NoFloodRoad) L10 **return** *Ps,t*

圖 3-7 BaseFIMain 演算法

首先，L01 是讀取淹水資訊，並對每條淹水區塊產生 MBR，L02 是讀取道路資訊，並對每條道路產生 MBR，L03 是根據淹水區塊物件建立 RTree，L04-L08 查詢淹水索引，並輸出未淹水的道路，L09 是輸入起終點做路徑規劃，找出最短路徑，L10 輸出最短路徑。

演算法名稱：**SearchFloodIndex**

輸入：entry //表示節點entry屬性, r //表示道路

輸出：true/false，道路與淹水區塊相交輸出true，不相交輸出false

```
L01   If (r.MBR intersects entry.MBR)
L02     If (entry.Ptr n is a leaf)
L03       for (each flood f of n)
L04         If(r.MBR intersect f && r intersect f)
L05           return true
L06         End If
L07       End for
L08       return false
L09     Else If (entry.Ptr n is not a leaf)
L10       for (each entry e of n)
L11         If(SearchFloodIndex(e, r))
L12           return true
L13         End If
L14       End for
L15       return false
L16     End If
L17   Else
L18     return false
L19   End If
```

圖 3-8　SearchFloodIndex 演算法

首先，L01 判斷道路 MBR 是否與 FloodIndex 節點相交，L03-L08 判斷 leaf 節點裡淹水區塊是否與道路 MBR 相交，如果節點裡所有淹水區塊與道路 MBR 沒有相交，那麼不會與道路本身相交，就回傳 false，如果節點裡其中一個淹水區塊與道路 MBR 相交，需進一步判斷是否與道路本身相交，如果相交就回傳 true，同圖 3-6，此處會依道路間的轉折點一段一段判斷。L10-L15 如果道路與 RoadIndex Nonleaf 節點相交，判斷其所有小孩節點，如果與所有小孩節點不相交，就回傳 false，反之回傳 true，直到該節點為 leaf 節點。

第 4 章 基於雲端服務之方法

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，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，快速輸出迴避淹水區域的路徑。我們並針對鄰近點篩選設計不同的索引，稱作 CloudFRI 和 CloudFPI 方法。

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，當使用者輸入起迄點，主程式會呼叫『Google路徑查詢模組』查詢最短路徑。接著呼叫『淹水路口查詢模組』，判斷這條路徑是否有跟任何一個淹水區域重疊，若有淹水的道路，輸出該條道路路口點，然後呼叫『鄰近點篩選模組』，查詢附近未淹水的路口點。我們將起點、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『Google路徑查詢模組』查詢起點經過waypoint到目的地的最短路徑，若還是不幸經過淹水區域，我們就須修正此路徑，也就是刪除waypoint，重新呼叫GSP演算法以及『Google路徑查詢模組』，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，我們將查詢結果以網頁方式呈現。

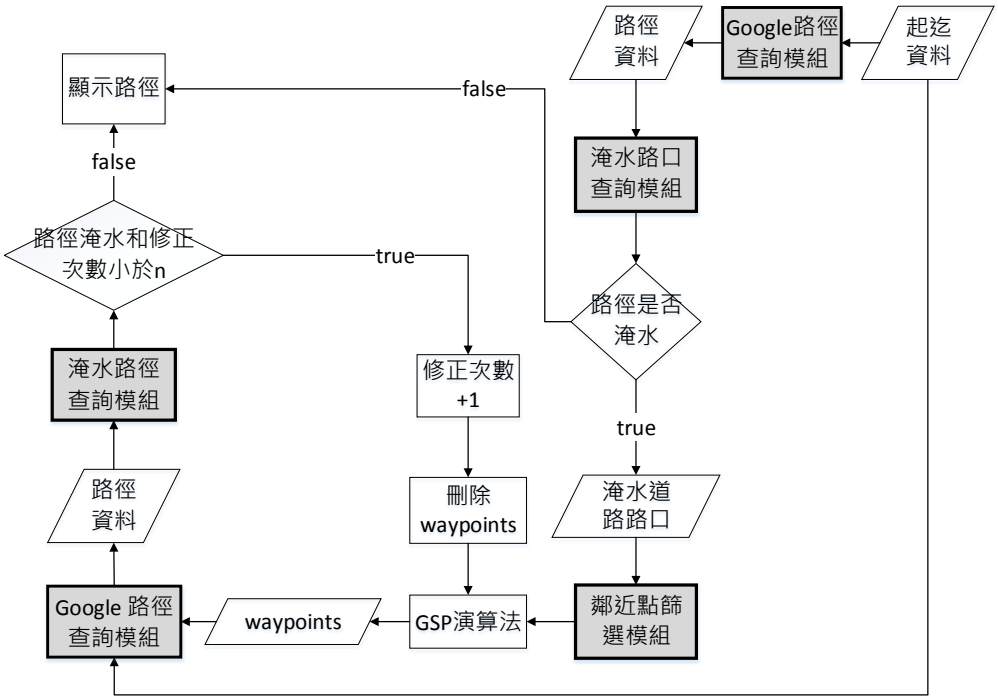


圖 4-1　Cloud 方法系統架構

4.2 主程式

完整的Cloud演算法如圖4-2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、淹水區塊FloodSet、鄰近點索引NPIndex以及order。首先，L01呼叫『Google路徑查詢模組』查詢V_s至V_t的最短路徑$P_{s,t}$，GetPathURL為雲端路徑規劃URL。接著，L02使用淹水區塊FloodSet建立索引。L03-07呼叫『淹水路口查詢模組』，透過SearchFloodIndex函數(如圖3-8)查詢$P_{s,t}$路徑是否與淹水區塊索引內的節點重疊，並將經過淹水區域道路的路口點儲存於FloodIntersectList集合。L09如果FloodIntersectSet為空集合，表示$P_{s,t}$路徑不與淹水區塊重疊，因此輸出此路徑。反之，L11呼叫『鄰近路口篩選模組』，查詢淹水路口點鄰近點，並將點分類為$C_0=\{V_s\}$、C_1~C_N為N個淹水路口鄰近點、$C_{N+1}=\{V_t\}$。L14經由GSP演算法選出每個種類代表的點當作waypoint，L15將V_s、V_t以及waypoint經由『Google路徑查詢模組』查詢最短路徑$P_{s,t}$，L16-L20判斷$P_{s,t}$是否經過淹水區域，如果經淹水區域，就刪除路徑waypoint，並將修正次數n加1。L22輸出最後路徑。

演算法名稱：Cloud_main

輸入：*Vs*, *Vt*, FloodSet, NPIndex, order

變數：FloodIntersectSet //儲存淹水路口點

GSPCategory//GSP 種類表，記錄每個種類所有點

Ci, Cj //Ci 表示種類個數，Cj 表示每個種類點個數

輸出：*Ps,t*

```
L01   Ps,t←FindGoogleMapsShortestpath(GetPathURL,Vs,Vt)
L02   FloodIndex←BuildRTree(FloodSet, order)
L03   For each road Ri,j ∈ Ps,t do
L04       If(SearchFloodIndex(FloodIndex.Root, Ri,j))
L05           FloodIntersectSet.insert(Vi, Vj)
L06       End If
L07   End for
L08   If(FloodIntersectSet.empty())
L09       return Ps,t
L10   Else
L11       GSPCategory←BuildNearPoint(FloodIntersectSet, NPIndex, FloodIndex, Vs, Vt)
L12       Initialize Time←0
L13       do
L14           Waypoint←GSP(GSPCategory, Ci, Cj)
L15           Ps,t←FindGoogleMapsShortestpath(GetPathURL,Vs,Vt,Waypoint)
L16           Flooded←SearchFloodIndex(FloodIndex.Root, Ps,t)
L17           If(Flooded)
L18               DeleteWaypoint(GSPCategory)
L19               Time++ //記錄修正次數
L20           End If
L21       while(Flooded & Time < n)
L22       return Ps,t
L23   End If
```

圖 4-2　Cloud 主程式

4.3 Google 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 、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 服務搭配 C#程式語言，並輸入 SearchURL 呼叫服務。演算法如圖 4-3 所示。

L02 是使用 WebRequest 通訊協定開啟查詢路徑的網頁 SearchURL，而 SearchURL 分為兩種，一種輸入 *Vs* 和 *Vt*，查詢 *Vs* 到 *Vt* 最短路徑[2]，另一種是輸入 *Vs*、*Vt* 以及 waypoint，查詢 *Vs* 經過 waypoints 到 Vt 的最短路徑[3]，waypoints 最多只能有 8 個點，回傳格式均為 JSON[4]。L03 接收回傳的 JSON 格式後進行剖析得到路徑經過的點集合。然後 L04 利用 Join_Point 演算法，輸入回傳點及合併距離值，將點合併成路徑，並存於 GPathSet 集合，目的是利用集合裡路徑判斷哪些經過淹水區域。最後，L05 輸出 GPathSet。

演算法名稱：FindGoogleMapsShortestPath 輸入：SearchURL、V_s、V_t、waypoint(查詢起點經 waypoint 至終點時輸入) 變數：GPoint //表示 GoogleMap 回傳的點 輸出：GPathList 代表 GoogleMaps 規劃 V_s 到 V_t 的最短路徑

L01 **Initialize** GPathList←NULL; //初始值為空
L02 **WebRequest** WRGM = WebRequest.Create(SearchURL)
L03 GPoint = parse(WRGM.GetResponse( ))
L04 GPathSet = Join_Point(GPoint, t) //t 為合併距離值
L05 **return** GPathSet

圖 4-3 FindGoogleMapsShortestPath 演算法

[2] SearchURL 為 https://maps.googleapis.com/maps/api/directions/json?origin= *Vs* &destination= *Vt* &mode=driving&language=zh-TW

[3] SearchURL 為 https://maps.googleapis.com/maps/api/directions/json?origin= *Vs* &destination= *Vt* &waypoints=points&mode=driving&language=zh-TW

[4] "routes": [{ "overview_polyline": {"points_decode"} }]

合併與未合併差異在於路徑數量，未合併的作法是將 Google 回傳點依序以每兩點形成的路徑，如表 4-2，每一列代表路徑，每一欄代表該路徑其中點，而合併的作法是先將前兩個點先合併成一條路徑 Path，接著從第二點開始，計算與下一點(第三點)距離是否小於合併距離值 t，如果小於 t，將下一點(第三點)新增至路徑 Path，反之將第二點與下一點(第三點)合併成另一條路徑(與路徑 Path 不同)，持續比對至最後一點，如表 4-3，每一列代表一條路徑，每一欄代表該路徑其中點，完整演算法如圖 4-4。我們發現合併的道路個數比未合併少，因此可增加查詢淹水道路之效率。

Join_Point 演算法是將 Google 回傳的點序列合併成路徑[5]。L02 先新增 GoogleMap 回傳的前兩個點。L03-L13 從第二點開始比對至最後的點。L05 判斷點與點大園距離是否小於合併距離值 t。L06 如果大園距離小於合併距離值 t，將點新增至 Path 路徑。L08 如果點與點大園距離大於等於 t，就將目前路徑 Path 新增至 PathList 集合。L09 清空路徑 Path 所有點。L10 將這兩點另外合併成一條路徑 Path。L15 回傳最後路徑集合 PathList。

演算法名稱：Join_Point 輸入：GPoint //表示 GoogleMap 回傳的點序列 　　t //合併距離值 變數：Path //表示暫存路徑 輸出：PathList //代表合併出來路徑集合

L01　**Initialize** PathList←NULL, i←1; //設定初始值
L02　Path.insert(GPoint[0], GPoint[1]) //新增回傳前兩個點
L03　**while**(i < GPoint.size() - 1)
L04　　j←i + 1;
L05　　**If**(Dist(GPoint[i],GPoint[j]) < t)
L06　　Path.insert(GPoint[j]) //新增點 GPoint[j]至 Path 路徑
L07　　**Else**

[5] 路徑可包含多個點。

```
L08             PathList.insert(Path) //將路徑 Path 新增至 PathList 集合
L09             Path.clear //清除暫存路徑所有的點
L10             Path.insert(GPoint[i], GPoint[j])
L11         End If
L12         i←i + 1;
L13     End while
L14     PathList.insert(Path) //將路徑 Path 新增至 PathList 集合
L15     return PathList
```

圖 4-4　Join_Point 演算法

舉例來說，我們利用 GoogleMap 路徑規劃服務查詢(121.52157 25.04983)至(121.51918 25.0478)最短路徑，結果如表 4-1，而未合併前的路徑如表 4-2，接著假設合併距離值 t 為 50m，利用 Join_Point 演算法得到合併後的路徑，結果如表 4-3，每一列代表一條路，每一欄代表該條道路的其中一點。此合併的結果會影響到圖 4-2 的 L03 行，也就是合併前該迴圈執行 10 次(表 4-2)，但合併後只會執行 4 次(表 4-3)。至於合併後還得留中間的轉折點，是因為圖 3-6 在判斷道路本身是否淹水時，會依照轉折點分段判斷。

表 4-1　Google 回傳點

順序	點	順序	點
1	121.52157,25.04983	7	121.51995,25.04736
2	121.52086,25.04828	8	121.51918,25.04749
3	121.52084,25.04823	9	121.51924,25.04773
4	121.5207,25.04825	10	121.51923,25.04776
5	121.52049,25.04783	11	121.51918,25.0478
6	121.52022,25.04731		

表 4-2　未合併路徑

順序	路徑	
1	121.52157,25.04983	121.52086,25.04828
2	121.52086,25.04828	121.52084,25.04823
3	121.52084,25.04823	121.5207,25.04825
4	121.5207,25.04825	121.52049,25.04783
5	121.52049,25.04783	121.52022,25.04731
6	121.52022,25.04731	121.51995,25.04736
7	121.51995,25.04736	121.51918,25.04749
8	121.51918,25.04749	121.51924,25.04773
9	121.51924,25.04773	121.51923,25.04776
10	121.51923,25.04776	121.51918,25.0478

表 4-3　Join_Point 得到合併路徑

順序	路徑				
1	121.52157, 25.04983	121.52086, 25.04828	121.52084, 25.04823	121.5207, 25.04825	
2	121.5207, 25.04825	121.52049, 25.04783			
3	121.52049, 25.04783	121.52022, 25.04731	121.51995, 25.04736		
4	121.51995, 25.04736	121.51918, 25.04749	121.51924, 25.04773	121.51923, 25.04776	121.51918, 25.0478

4.4 鄰近點篩選模組

本節說明圖 4-2 中 L11 行所呼叫的 BuildNearPoint 演算法。完整的演算法如圖 4-5 所示，L01 將點分類為 $C0$={*Vs*}，L03-L09 將 $C1$~CN 為 N 個淹水路口建立鄰近點，L10 將點分類為 $CN+1$={*Vt*}，而 PointCategory 為 GSP 點種類表，記錄每個種類(也就是 $C0$~$CN+1$)所有點，L04 呼叫 **FindNearPoint** 演算法查詢淹水路口鄰近點，L11 輸出建立點種類表。

接下來，我們說明 L04 所呼叫的 FindNearPoint 演算法。鄰近點查詢通常是 range query，給定一個 Query Location、臨近距離值及路網圖，找出以 Query Location 為中心，臨近距離值範圍內的路口點，也就是說 Query Location 與輸出路口點距離小於等於臨近距離範圍值。以下分節說明事前建立的索引和 Online 查詢的方法。

演算法名稱：BuildNearPoint 輸入：FloodIntersectList //為淹水路口點集合 NPIndex, FloodIndex,*Vs*,*Vt* 輸出：PointCategory //點的種類表，記錄每個種類的點
L01 PointCategory[0][0]←*Vs* L02 **Initialize** i←1, j←0 //i 表示某個種類，j 表示第 i 個種類某個點 L03 **For**(each **P** ∈ FloodIntersectList) L04 NPointList←**FindNearPoint**(**P**, NPIndex.root, FloodIndex.root, *Vs*, *Vt*) L05 **For(**each *v* of NPointList) L06 PointCategory[i][j++] = *v* L07 **End For** L08 i++ L09 **End For** L10 PointCategory[i][0]←*Vt* L11 **return** PointCategory

圖 4-5 BuildNearPoint 演算法

4.4.1 **FindNearPoint 資料結構**

為了避免與所有路口一一比對，我們使用兩種不同鄰近點索引，一種是擴充同 3.2 節的 RoadIndex，也就是 Non-Leaf 節點額外記錄 Center 和 radius 屬性來表示節點『**外接圓**』，而 Leaf 節點除了記錄 Center 和 radius 屬性，還額外記錄 NodePointList 屬性，表示該節點道路路口點，如圖 4-6。比較原始的索引(如圖 3-3)，我們可看出每個節點額外記錄 Center、radius 和 NodePointList 三個屬性，舉例來說 N_1 節點的 Center 為(4.5,3.5)，radius 為 3.8，NodePointList 為[V_1, V_7, V_8, V_9]，目的是尋找節點路口點是否在鄰近距離範圍內。利用此索引的作法稱為 **CloudFRI**。另一種是另外建立一個路口索引 PointIndex，主要是根據路網圖所有路口點建立的索引，並在節點裡記錄 Center、radius 和 NodePointList 屬性，如圖 4-7，利用此索引的作法稱為 **CloudFPI**。之所以設計這兩種索引，是因為利用點索引建立出來的節點 MBR 面積比擴充的 RoadIndex 小，MBR 重疊也較少，因此 CloudFRI 查詢效率較 CloudFRI 好。比較圖 4-6 和圖 4-7，可看出道路索引的作法樹高比路口索引作法高，節點個數也較多。本論文設計 **FindNodeCircle** 演算法尋找節點 Center 以及 radius 屬性，如圖 4-8。

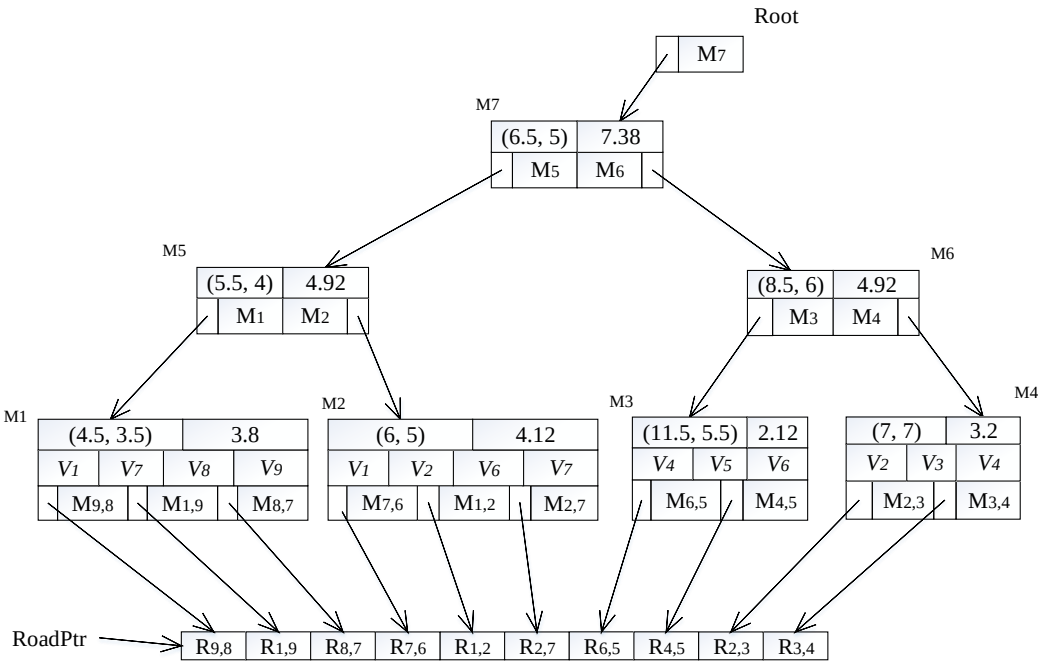


圖 4-6　擴充的道路索引示意圖

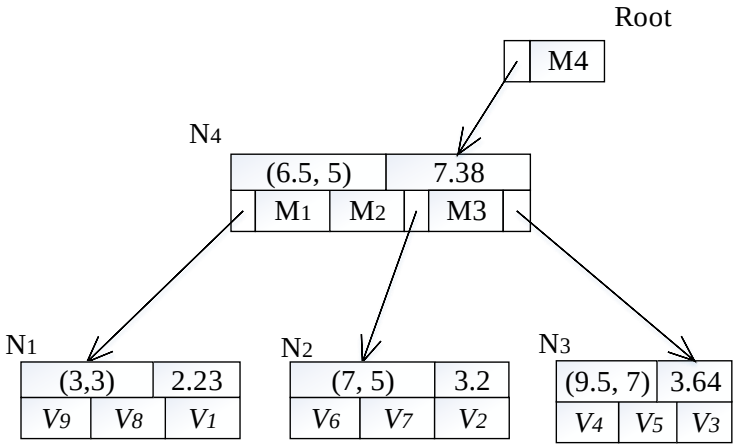


圖 4-7　路口點索引示意圖

演算法名稱：**FindNodeCircle**
輸入：Node //未設定 Center 以及 radius 屬性
變數：EndPoint, mbr
輸入：Node //指定完 Center 以及 radius 屬性
L01　mbr = FindMBR(Node) //計算節點MBR
L02　Node.Center.x = (mbr._xmin + mbr._xmax) / 2 //計算中心點X軸座標
L03　Node.Center.y = (mbr._ymin + mbr._ymax) / 2 //計算中心點Y軸座標
L04　EndPoint.x = mbr._xmin //左下點x軸座標
L05　EndPoint.y = mbr._ymin //左下點y軸座標
L06　Node.radius = **Dist**(Node.Center, EndPoint) //計算中心點與左下點
L07　**return** Node

圖 4-8　尋找外接圓半徑演算法

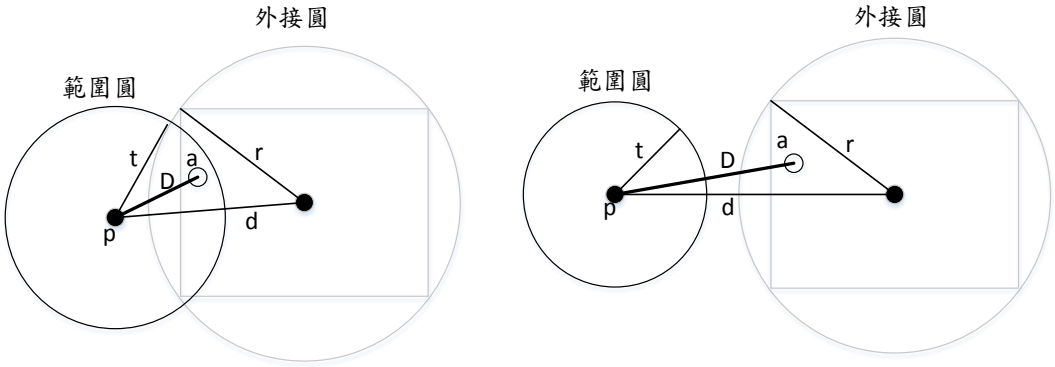
4.4.2　**FindNearPoint 作法**

為了尋找鄰近點，我們定義淹水路口的『**範圍圓**』，也就是以淹水路口為中心，threshold t 為圓半徑形成的圓，目的是要尋找鄰近點。我們利用淹水路口點與節點外接圓中心點距離扣掉半徑後是否小於淹水路口範圍圓的半徑，如果小於等於表示該節點某些路口在範圍圓內，外接圓與範圍圓相交，如圖 4-9(a)，反之在範圍圓外，外接圓與範圍圓不相交，如圖 4-9(b)。

以圖 4-9 為例，p 為淹水路口點，d 表示淹水路口 p 與節點外接圓中心的距離，

r 為外接圓半徑，a 為節點路口點，D 為淹水路口 p 與路口點 a 的距離，若 d-r<=t，表示 p 範圍圓與節點外接圓相交，節點的某些路口點可能在 p 的範圍園內，如圖 4-8(a)路口點 a，因此進一步尋找在範圍內的路口點，反之如圖 4-9(b)，p 範圍圓與節點外接圓不相交，節點路口點 a 在 p 的範圍園外，不是我們要尋找的替代點。

以圖 4-10 為例，我們使用路口索引，標示 Leaf 節點的外接圓，如虛線圓圈，*V2* 範圍圓，如實線圓圈，而白色圓圈表示節點路口點，黑色圓圈表示淹水路口點，灰色圓圈表示淹水路口點的鄰近點。我們以 *V2* 為例，尋找其鄰近點，假設鄰近距離值為 3.2，使用路口索引查詢，從根節點開始往下尋找，由圖得知 N4 節點外接圓與 *V2* 範圍圓相交，往小孩節點尋找，而 N1、N2、N3 節點外接圓與 *V2* 範圍圓相交，因此在 N1、N2、N3 內的路口點尋找 *V2* 的鄰近點，由於 *V2* 與 *V4*、*V5*、*V6*、*V7*、*V9* 距離大於鄰近距離值，與 *V1*、*V3* 和 *V8* 距離小於鄰近距離值，但 *V1* 為起點，不是我們所需找的鄰近點，因此 *V3*、*V8* 為 *V2* 鄰近點。以下介紹尋找鄰近點演算法，如圖 4-11。



(a) 範圍圓與外接圓相交　　(b) 範圍圓與外接圓不相交

圖 4-9　淹水點與某一索引節點的關係

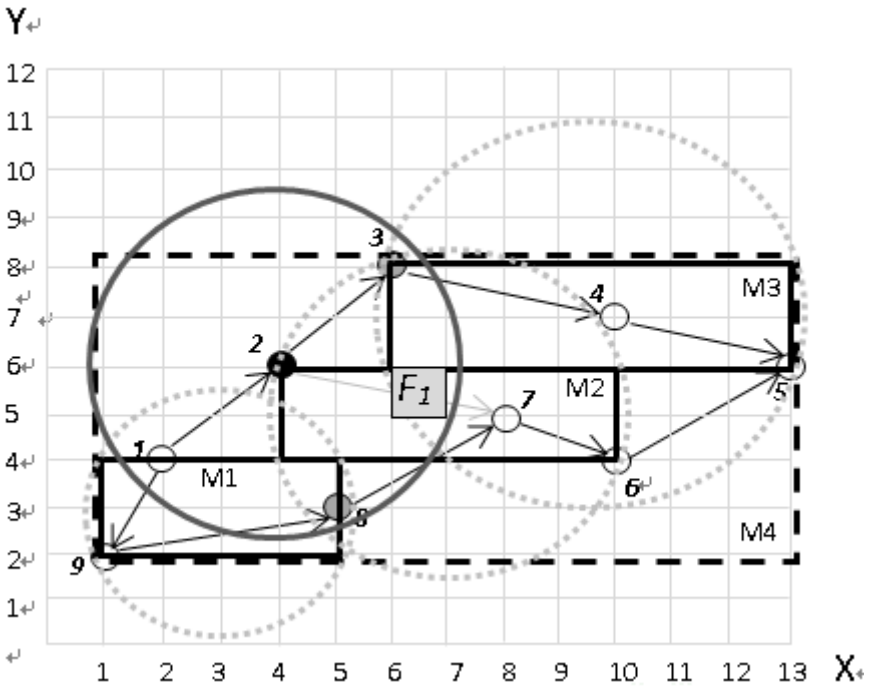


圖 4-10　尋找鄰近節點示意圖

演算法名稱：**FindNearPoint** 輸入：FIP, Node, FloodIndex, threshold, *Vs*, *Vt* 輸出：FIP.NearPoint
L01　**If(**Node is not a leaf) L02　　**If**(Dist(FIP.p, Node.Center) –Node.radius <= **threshold**) L03　　**For** (each ChildNode **c** of Node) L04　　**FindNearPoint**(FIP.p**, c**, FloodIndex.root) L05　　**End for** L06　　**End If** L07　**Else If(**Node is a leaf**)** L08　　**If**(Dist(FIP.p , Node.Center) – Node.radius <= **threshold**) L09　　**For** (each point **NP** of Node.NodePoint) L10　　**If**(**NP** != FIP.p) L11　　**If**(**Dist**(FIP.p , **NP**) <= threshold) L13　　**If**(**not WithinMBR**(**NP**, FloodIndex.root) && **NP** != *Vs* && **NP** != *Vt*) L14　　FIP.NearPoint.push(**NP**) L15　　**End If**

```
L16              End If
L17            End If
L18          End for
L19        End If
L20    End If
L21    return FIP.NearPoint
```

圖 4-11 **FindNearPoint** 演算法

首先,說明 FIP 資料結構,FIP 記錄淹水路口點 **p** 和鄰近點集合 **NearPoint**。而程式裡定義 **WithinMBR** 函數表示判斷候選鄰近點是否與淹水區域重疊,L01 判斷節點是否為 leaf,L02-L06 如果是 non-leaf,計算路口點 **p** 與目前節點外接圓大圓距離,判斷是否在 threshold 範圍內,如果在範圍內,計算路口點 **p** 與小孩節點外接圓大圓距離,判斷是否在 threshold 範圍內,直到該節點為 leaf 節點。L08-L19 如果是 leaf 節點,計算路口點 **p** 與目前 leaf 節點大圓距離,判斷是否在 threshold 範圍內,如果在範圍內,L10-L17 計算路口點 **p** 與節點所有路口點大圓距離,判斷是否在 threshold 範圍內、是否為淹水路口、不與淹水區塊重疊以及不為起迄點(L11-L16),並將結果新增至 FIP 的 NearPoint 屬性,L21 回傳 FIP 之鄰近節點。

4.5 GSP 演算法

在此節中，我們說明圖 4-1 中的「GSP 演算法」，該作法是根據論文[RT13]所提出之基於動態規劃的作法。其計算式如圖 4-12 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 代表兩個種類點之間的距離。與原論文之所不同的是，此處距離是直接計算兩點間的直線距離，而非經由道路距離[6]。我們會將起始點 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 的代表點，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 4-12　GSP 計算式

演算法名稱：GSP 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**

[6] 實作時是用大圓距離。

```
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
L18          X[i][j]←min_dis;
L19          VisitID[i][j]←tmp_vid;
L20          If Ci_min_dis>min_dis then
L21             Ci_min_dis←min_dis;
L22          End If
L23       End for
L24       Ci_min[i]←Ci_min_dis;
L25    End for
L26    For i←0 to Ci_min.length do
L27       for j←0 to C[i].length do
L28          min_dis←Ci_min[i];
L29          If min_dis is X[i][j] then
L30             final_road.insert(VisitID[i][j]);
L31               break;
L32          End If
L33       End for
L34    End for
```

```
L35      return final_road;
```

圖 4-13　GSP 演算法

圖 4-13 為 GSP 的演算法，L01-L24 的演算範圍是 C_i 個數 i ($0 \leq i \leq$ iMax)，L03-22 演算範圍是 C_i 內的項目個數 j ($0 \leq j \leq$ iMax)。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 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 即為每個種類內代表點。

最後針對圖 3-1 路網圖，給定起點為 V_1、目的地為 V_5，解釋如何使用 Cloud 方法，取得 $P_{1,5}$ 路徑。

首先，呼叫『Google 路徑查詢模組』查詢 V_1 至 V_5 的最短路徑 GPathList，如表 4-4。接著呼叫『淹水路徑查詢模組』，利用淹水索引判斷 GPathList 走訪的路徑是否有經過淹水區域，若有則將淹水道路路口點儲於 FloodIntersectList 集合，如 4-5，可得知道路 $R_{2,7}$ 的 MBR 與淹水區域重疊，所以呼叫『替代節點篩選模組』，使用路口索引查詢 V_2 和 V_7 淹水的路口點的鄰近點集合，一開始根據圖 4-10 來尋找替代節點。從根節點開始，假設 threshold 值為 3.2 公里，首先找出 V_2 鄰近點，由於 V_2 包含在根節點裡，因此判斷 V_2 範圍圓與 N_1、N_2 和 N_3 節點外接圓是否相交，由圖得知與 N_1、N_2 和 N_3 節點外接圓相交，因此在 N_1、N_2 和 N_3 節點尋找 V_2 的 threshold 範圍內的路口點，而路口點非起迄點以及不與淹水區塊重疊，結果如表 4-6。接著尋找 V_7 鄰近點，從根節點開始查詢，由於 V_7 包含在根節點裡，因此判斷 V_7 範圍圓與 N_1、N_2 和 N_3 節點接圓是否相交，由圖得知與 N_2 和 N_3 節點外接圓相交，因此在 N_2 和 N_3 節點裡尋找 V_7 的 threshold 範圍內的路口點，執行結果如表 4-7，然後將起點與目的地及兩個淹水路口點鄰近點建立 4 個種類表，$C_0=\{V_1\}$、$C_1=\{V_3、V_8\}$、$C_2=\{V_4、V_6\}$、$C_3=\{V_5\}$，利用

GSP 演算法選擇每一個種類走訪的點，如表 4-8。最後，我們個別從 V_2、V_7 之鄰近節點集合裡選擇 V_8、V_6 當作 waypoint，得到起點經 V_8、V_6 到終點的路徑 $P_{1,5}$= <V_1,V_9,V_8,V_7,V_6,V_5>，而此路徑不經過淹水區域，因此輸出此路徑。

表 4-4　GPathList 結果

走訪點
V_1-> V_2-> V_7-> V_6-> V_5

表 4-5　淹水的路口點

點
V_2
V_7

表 4-6　V_2 之鄰近節點集合

點
V_3
V_8

表 4-7　V_7 之鄰近節點集合

點
V_4
V_6

表 4-8　GSP 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=6	6	$C_{0,1}$
$X[1,2]$	X[0,1]+ Distance ($C_{0,1}$, $C_{1,2}$)= 0+3.2=3.2	3.2	
$X[2,1]$	X[1,1]+ Distance ($C_{1,1}$, $C_{2,1}$)=6+4.2=10.2 X[1,2]+ Distance ($C_{1,2}$, $C_{2,1}$)=3.2+6.6=9.8	9.8	$C_{1,2}$
$X[2,2]$	X[1,1]+ Distance ($C_{1,1}$, $C_{2,2}$)=6+5.7=11.7 X[1,2]+ Distance ($C_{1,2}$, $C_{2,2}$)=3.2+5.1=8.3	8.3	
$X[3,1]$	X[2,1]+ Distance ($C_{2,1}$, $C_{3,1}$)= 9.8+3.2=13 X[2,2]+ Distance ($C_{2,2}$, $C_{3,1}$)= 8.3+3.6=11.9	11.9	$C_{2,2}$

接下來，我們以新竹市起迄資料為例，起點 V_s(120.97805,24.8235)、目的地 V_t(120.98487,24.81597)，首先呼叫『Google 路徑查詢模組』查詢起點至終點最短路徑，使用合併距離值為 100m，利用 Join_Point 演算法將回傳點合併成路徑，如表 4-9，接著呼叫『淹水路口查詢模組』，利用淹水索引判斷表 4-9 的道路是否經淹水區域，舉例來說順序 1 的路徑經淹水區域，因此輸出其淹水路口點，如表

4-10，並且利用路口索引以及臨近距離值為 0.13km 方式尋找其臨近點，如表 4-11 和表 4-12，然後將起點、目的地及淹水路口鄰近點建立 4 個種類表，C_0={V_s}、C_1={NP_1、NP_2、NP_3、NP_4、NP_5、NP_6、NP_7}、C_3={ NP_8、NP_9、NP_{10}}、C_4={V_t}，利用 GSP 演算法選擇每一個種類走訪的點，如表 4-13，此範例則使用大圓距離計算點間距離。最後，我們分別從 FIP_1、FIP_2 之鄰近節點集合裡選擇 NP_4 和 NP_{10} 當作 waypoint，將此四點<V_s,NP_4,NP_{10},V_t>，送交 google 路徑查詢模組規畫路徑，由於此路徑不經過淹水區域，因此顯示此路徑，如圖 4-14。

表 4-9　起點至終點最短路徑

順序	路徑			
1	120.97805 24.8235	120.97815 24.82348	120.97852 24.82364	120.97871 24.82352
2	120.97871 24.82352	120.97956 24.82258	120.97969 24.82251	
3	120.97969 24.82251	120.98064 24.82146		
4	120.98064 24.82146	120.98173 24.82025		
5	120.98173 24.82025	120.9824 24.81935	120.98288 24.8187	
6	120.98288 24.8187	120.98351 24.81785		
7	120.98351 24.81785	120.98438 24.81668	120.98462 24.8163	120.98487 24.81597

表 4-10　淹水的路口點

	點
FIP_1	120.97805,24.8235
FIP_2	120.97871,24.82352

表 4-11　FIP_1 淹水的路口鄰近點

	點		點
NP_1	120.976944,24.823023	NP_5	120.978141,24.822378
NP_2	120.977304,24.823178	NP_6	120.978246,24.822508
NP_3	120.977676,24.822752	NP_7	120.978306,24.822581
NP_4	120.977739,24.822802		

表 4-12　FIP_2 淹水的路口鄰近點

	點
NP_8	120.977739,24.822802
NP_9	120.978246,24.822508
NP_{10}	120.978306,24.8225818

表 4-13　真實範例 GSP 演算法過程

Matrix	Step	min_dis	VisitID
X[0,1]	-	0	-
X[1,1]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,1}$)= 0+0.1237=0.1237	0.1237	$C_{0,1}$
X[1,2]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,2}$)= 0+0.0834=0.0834	0.0834	
X[1,3]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,3}$)= 0+0.0914=0.0914	0.0914	
X[1,4]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,4}$)= 0+0.0838=0.0838	0.0838	
X[1,5]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,5}$)= 0+0.1252=0.1252	0.1252	
X[1,6]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,6}$)= 0+0.1121=0.1121	0.1121	
X[1,7]	X[0,1]+ Distance ($C_{0,1}$, $C_{1,7}$)= 0+0.1055=0.1055	0.1055	
X[2,1]	X[1,1]+ Distance ($C_{1,1}$, $C_{2,1}$)= 0.1237+0.084=0.2077 X[1,2]+ Distance ($C_{1,2}$, $C_{2,1}$)= 0.0834+0.0606=0.144 X[1,3]+ Distance ($C_{1,3}$, $C_{2,1}$)= 0.0914+0.0084=0.0998 X[1,4]+ Distance ($C_{1,4}$, $C_{2,1}$)= 0.0838+0=0.0838 X[1,5]+ Distance ($C_{1,5}$, $C_{2,1}$)= 0.1252+0.0622=0.1874 X[1,6]+ Distance ($C_{1,6}$, $C_{2,1}$)= 0.1121+0.0607=0.1728 X[1,7]+ Distance ($C_{1,7}$, $C_{2,1}$)= 0.1055+0.0623=0.1678	0.0838	$C_{1,4}$

X[2,2]	*X*[1,1]+ Distance ($C_{1,1}$, $C_{2,2}$)= 0.1237+0.1434=0.2671 *X*[1,2]+ Distance ($C_{1,2}$, $C_{2,2}$)= 0.0834+0.1209=0.2043 *X*[1,3]+ Distance ($C_{1,3}$, $C_{2,2}$)= 0.0914+0.0636=0.155 *X*[1,4]+ Distance ($C_{1,4}$, $C_{2,2}$)= 0.0838+0.0607=0.1445 *X*[1,5]+ Distance ($C_{1,5}$, $C_{2,2}$)= 0.1252+0.0179=0.1431 *X*[1,6]+ Distance ($C_{1,6}$, $C_{2,2}$)= 0.1121+0=0.1121 *X*[1,7]+ Distance ($C_{1,7}$, $C_{2,2}$)= 0.1055+0.0101=0.1156	0.1121	
X[2,3]	*X*[1,1]+ Distance ($C_{1,1}$, $C_{2,3}$)= 0.1237+0.1461=0.2698 *X*[1,2]+ Distance ($C_{1,2}$, $C_{2,3}$)= 0.0834+0.121=0.2044 *X*[1,3]+ Distance ($C_{1,3}$, $C_{2,3}$)= 0.0914+0.0664=0.1578 *X*[1,4]+ Distance ($C_{1,4}$, $C_{2,3}$)= 0.0838+0.0623=0.1461 *X*[1,5]+ Distance ($C_{1,5}$, $C_{2,3}$)= 0.1252+0.0281=0.1533 *X*[1,6]+ Distance ($C_{1,6}$, $C_{2,3}$)= 0.1121+0.0102=0.1223 *X*[1,7]+ Distance ($C_{1,7}$, $C_{2,3}$)= 0.1055+0.00008=0.10558	0.10558	
X[3,1]	*X*[2,1]+ Distance ($C_{2,1}$, $C_{3,1}$)= 0.0838+1.0298=1.1136 *X*[2,2]+ Distance ($C_{2,2}$, $C_{3,1}$)= 0.1121+0.9707=1.0828 *X*[2,3]+ Distance ($C_{2,3}$, $C_{3,1}$)= 0.10558+0.9725=1.07808	1.07808	$C_{2,3}$

圖 4-14　真實範例執行之結果

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 BaseRI、BaseFI、CloudFRI 及 CloudFPI 四種方法的差異。其中 BaseRI 方法是以道路建立索引，淹水區塊與道路做空間相交，找出淹水道路，BaseFI 方法是以淹水建立索引，道路與淹水索引節點做空間相交，找出淹水道路。兩個方法皆從路網中移除淹水道路，然後由 Dijkstra 演算法規劃出最短路徑。CloudFRI、CloudFPI 方法皆利用雲端路徑規劃服務查詢最短路徑，利用淹水區塊建立索引，判斷這條路徑是否經過淹水區域，將經過淹水區域道路的路口點查詢鄰近點，然後透過 GSP 演算法選出必經點(waypoint)，再將起點、必經點(waypoint)與終點交由 Google Maps 進行最後路徑之規劃與呈現。兩種方法差別在於尋找鄰近鄰近點的方式，CloudFRI 是利用 BaseRI 方法所建立的道路索引，如圖 4-6，CloudFPI 則利用路口點資料所建立的索引，如圖 4-7。

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 i5-3570 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。

5.1 輸入檔案格式

四種方法皆需輸入道路與淹水資料，因此介紹兩種資料輸入檔案格式，道路輸入檔案如表 5-1 所示，其中一行代表一條路的起點、轉折點、終點的經緯度座標和起迄經轉折點至終點的距離，中間以空白隔開，順序為起點、轉折點、終點和道路長度，如圖 5-1。我們利用一條道路的起點、轉折點以及終點計算道路 MBR 與淹水區塊相交，如圖 3-6 第 4 行的第一個判斷式，起點至轉折點路段以及轉折點至終點路段所形成道路與淹水區塊作空間相交(實作時則使用 boost Libraries)，如圖 3-6 第 4 行的第二個判斷式。淹水區域輸入檔案如表 5-2 所示，其中一列代表一個淹水區域，一個淹水區域有四個點，每個點為一組經緯度座標。

表 5-1 輸入的路網圖檔案(RoadFile)

點個數	內容
3	121.546218,25.044973 121.546629,25.044977 121.546935,25.044955 0.072409

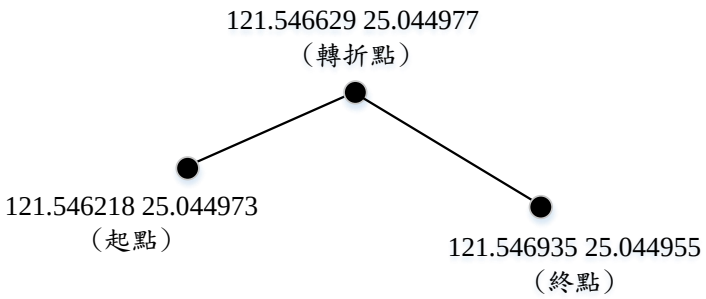


圖 5-1 道路示意圖

表 5-2 輸入的淹水區域檔案(FloodFile)

行 數	內 容							
1	120.5328	22.4236	120.5381	22.4236	120.5381	22.4253	120.5328	22.4253
2	120.5288	22.4237	120.5291	22.4237	120.5291	22.4254	120.5288	22.4254

5.2 系統實作方法與輸出範例

BaseRI、BaseFI 兩個方法以 Microsoft Visual C++ 2012 進行實作。而 CloudFRI 和 CloudFPI 方法除了主程式以及替代節點篩選模組以 C++實作外，雲端路徑查詢模組是利用 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。

我們會將四個方法回傳的路徑輸出於不同的網頁中，BaseRI 和 BaseFI 兩個方法是將得到的路徑以線段方式(Polyline)呈現在 Google Maps 上，而 CloudFRI 和 CloudFPI 兩個方法則是將得到的路口(必經點)、輸入起點和終點利用 Google Maps Direction API 進行路徑規劃與呈現。

以起點(120.9143,24.7747)、終點(120.9329,24.7847)做為範例顯示四個作法的不同，如圖 5-2 深色線段為 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 5-3 為 BaseRI 和 BaseFI 迴避淹水區域的路徑規劃。而如圖 5-4 為初次 Google Maps 規劃的最短路徑(尚未判斷淹水區域)、圖 5-5 為 Cloud 迴避淹水區域透過替代點的規劃路徑。我們可看到這些方法成功規劃出迴避淹水路徑，但路徑不盡相同，後續我們將做進一步的實驗與比較。

圖 5-2 Dijkstra 最短路徑規劃

圖 5-3　BaseRI 與 BaseFI 方法迴避淹水區域

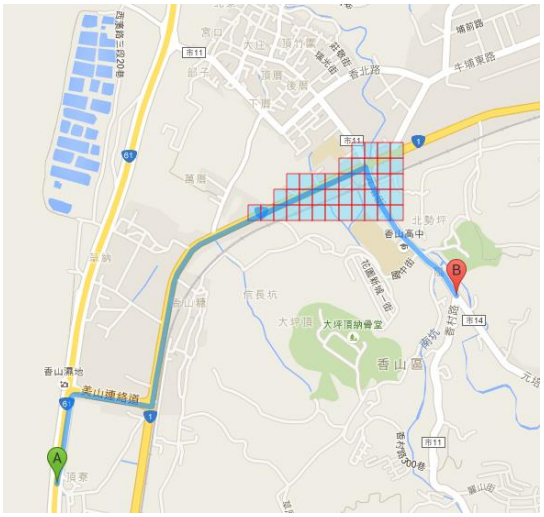


圖 5-4　GoogleMaps 最短路徑的路線

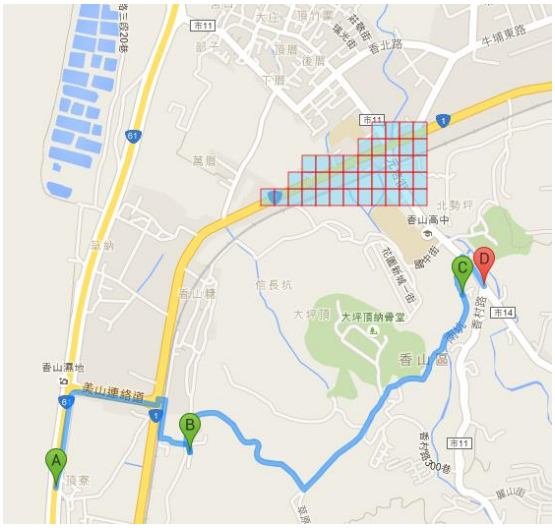


圖 5-5　CloudRI、CloudPI 方法迴避淹水的路線

5.3 資料集

本實驗所使用的資料集包括交通部路 102 年出版的路網數值圖、國家災害防救科技中心[7]提供的各縣市『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，我們挑選三個縣市路網圖，分別為基隆市、新竹市以及台北市路網圖資料，意義代表著各種不同大小資料集，大小是依據道路和路口點個數判斷。

淹水潛勢圖分別挑選基隆市、台北市及新竹市。時間為 104 年 6 月。我們選擇深度達 1~2m 的灰色淹水區域，如圖 5-6，然後取該區域最小邊界矩形，接著判斷矩形面積是否夠小，如果面積小於等於 $0.0049km^2$就不切割，反之切割為 $0.0049km^2$(長寬皆 0.07km)的淹水區塊，選擇 $0.0049km^2$矩形主要原因是約略為最小的淹水矩形面積，且希望切割出來的矩形面積盡可能一致。最後將未與淹水區塊重疊的矩形去除。

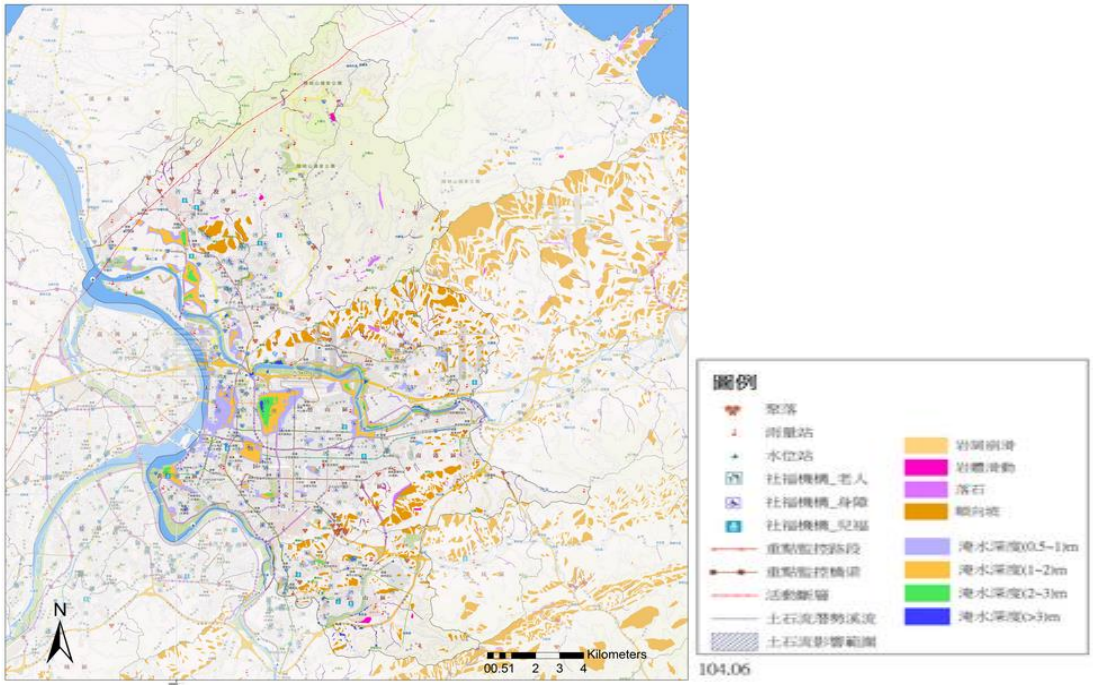


圖 5-6　台北市淹水潛勢圖範例

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

人工所產生出來的淹水區域資料範圍整個台北地區，且根據高斯、隨機與群聚分佈不重複的產生 0.01km^2(長寬約 0.1km)的淹水區塊。以下是不同縣市路網參數比較，如表 5-3 所示。表 5-4 表示 10 組台北市起迄點之測試資料。

表 5-3　資料集之各項參數

	路口點 (不含轉折點)	道路	真實 淹水	人工淹水	threshold 範圍值	threshold 合併值	索引 order
基隆(Klu)	1464	1600	148		0.5km 1km(預設) 1.5km	50m 100m 150m 200m(預設)	16 32 64(預設)
新竹(Hchc)	10151	11526	330				
台北(Tpe)	18872	21652	655	個數 100、150、200 分布 Gaussian、 Cluster、Random			

表 5-4　10 組台北市起迄點之記錄表

	Q1	Q2	Q3	Q4	Q5
V_s	121.526526 25.029662	121.526526 25.029662	121.526526 25.029662	121.526526 25.029662	121.538077 25.038036
V_t	121.584596 25.044142	121.545926 25.048064	121.616744 25.05812	121.578907 25.058233	121.578907 25.058233
距離	6.074km	2.832km	9.634km	6.166km	4.691km
	Q6	Q7	Q8	Q9	Q10
V_s	121.616744 25.05812	121.545926 25.048064	121.616744 25.05812	121.538077 25.038036	121.517748 25.039436
V_t	121.506016 25.041403	121.513682 25.050453	121.513682 25.050453	121.544088 25.049281	121.616452 25.043474
距離	11.32km	3.262km	10.428km	1.39km	9.964km

5.4 改變索引建立方式和 order 之實驗

本實驗路網採用台北市(12419 個路口點及 15874 個路)，淹水區域採用真實資料的 655 個淹水區域。我們比較使用不同索引以及 order 建立、查詢以及整體效率。而實驗方式執行索引建立、查詢以及輸入資料至查詢各十次，然後取這十次平均。

(1) 這次實驗，我們使用 order 值為 32，針對 BaseRI 和 BaseFI 搭配不同索引建立方式，也就是 Sorted、Split 和 Hilbert 三種進行比較，分別稱為 SortedRI、SplitRI、SortedFI、SplitFI、HilbertRI 和 HilbertFI。注意到，Sorted 方式是先依 MBR 右上點 y 軸座標值，再依 x 軸座標值排序，Hilbert 是將 MBR 中心點換算成 Hilbert 值在排序，Split 則是依據 R*-tree 的建立方式。結果如圖 5-7。由圖可觀察出以下幾種現象

- SortedRI、SortedFI 索引整體時間比 HilbertRI、HilbertFI 好，主要是因為 HilbertRI、HilbertFI 花較多查詢時間。
- SplitRI、SplitFI 索引查詢效率比 SortedRI、SortedFI 索引好，主要是因為 overlap 程度降低，判斷相交次數減少，因而減少查詢時間。
- SplitFI 索引整體時間比 SortedFI 好，主要是因為 SortedFI 花較多查詢時間。SortedRI 索引整體時間比 SplitRI 好，主要是 SplitRI 花較多時間建立，但 SplitRI 有較好的查詢效率。
- 本實驗使用道路個數遠多於淹水區塊，SplitRI 方法需花大部分的時間建立索引，相對於 SplitFI 方法只花較少時間建立索引，而 SplitRI 查詢時間與 SplitFI 接近，因而 SplitFI 整體時間比 SplitRI 好。

整體來看，SplitFI 的總時間最佳，而 SplitRI 的查詢時間最短。

表 5-5　真實資料集中改變索引建立方式之實驗紀錄

	SortedRI	SplitRI	HilbertRI	SortedFI	SplitFI	HilbertFI
Build	3.524	8.085	1.868	0.003	0.15	0.003
Query	7.87	4.985	11.154	3.856	2.882	3.854
Total	11.394	13.07	13.022	3.859	3.032	3.696

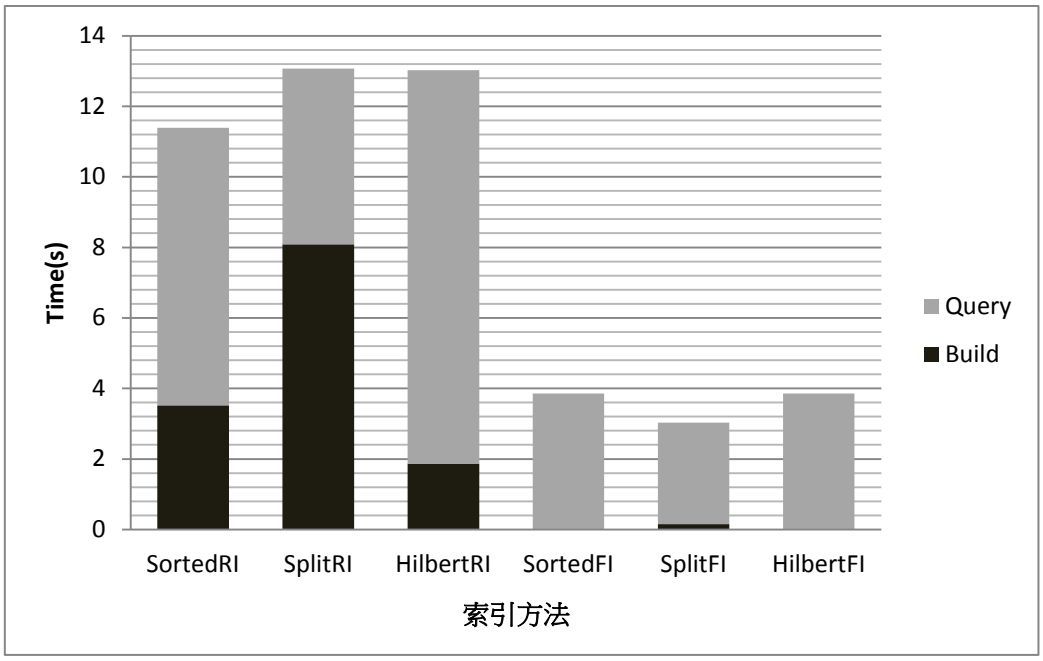


圖 5-7　真實資料集中改變索引方法之影響

(2) 這次實驗，而針對 SplitRI 和 SplitFI 方法使用不同 order 數值進行比較。結果如圖 5-8、圖 5-9，圖 5-8 記錄兩個方法改變 order 值之查詢時間，圖 5-9 記錄兩個方法改變 order 值之建立、查詢和建立至查詢總時間。由圖 5-8 可觀察出 SplitRI 方法的查詢時間不隨著 order 數值增加而明顯上升，主要是節點 MBR 面積雖然變大，但 overlap 程度不會明顯增加，因為 SplitRI 方法會新增物件至增加面積最小的節點，讓面積不至增加太多，這樣可以減少 overlap 情形發生，如此一來判斷相交次數沒有變多，查詢時間也沒太大變化。SplitFI 方法的查詢時間隨著 order 數值增加而趨近線性下降，因為 order 變大，索引樹的高度降低，使判斷道路與淹水區塊次數變少。我們嘗試不建立索引，查詢淹水道路，發現執行時間為 5310.6 秒，比使用索引慢許多，因此本論文均使用索引。

表 5-6　真實資料集中改變 order 值之查詢實驗紀錄

	16	32	64
SplitRI 執行時間	4.978	4.985	4.995
SplitFI 執行時間	3.55	2.882	2.504
SplitFI 樹高	4	4	3

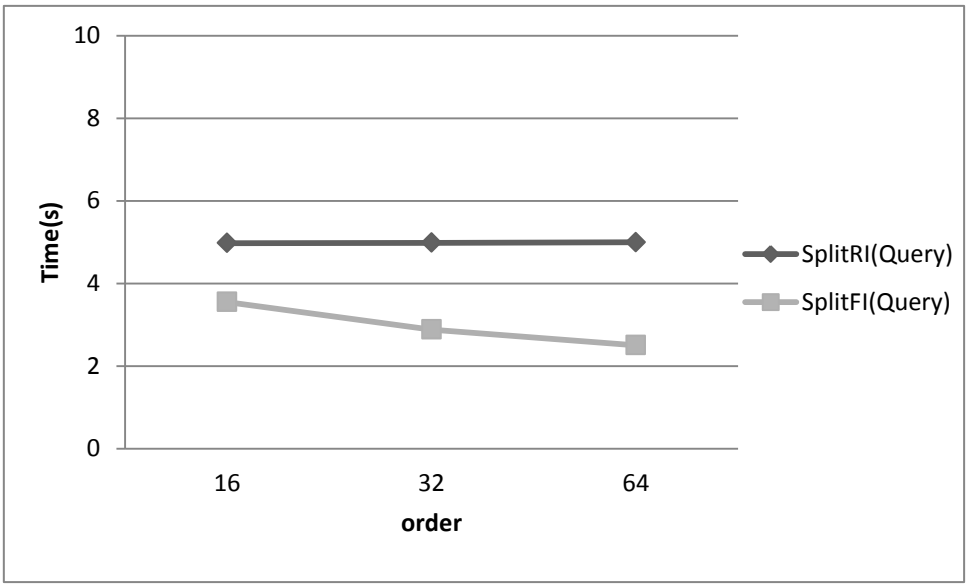


圖 5-8　真實資料集中改變 order 值之查詢影響

由圖 5-9 可觀察出 SplitFI 方法的總時間隨著 order 數值增加而趨近線性下降，SplitRI 方法的總時間隨著 order 數值增加而線性上升，主要是因為 SplitFI 方法建立時間變化不大，查詢時間而趨近線性下降，而 SplitRI 方法建立時間隨著 order 數值增加而線性上升，查詢時間變化不大，導致總時間變化很大。

表 5-7　真實資料集中改變 order 值之整體實驗紀錄

	16	32	64
SplitRI(Total)	11.819	13.07	14.605
SplitRI(Build)	6.841	8.085	9.61
SplitRI(Query)	4.978	4.985	4.995
SplitFI(Total)	3.684	3.032	2.622
SplitFI(Query)	3.55	2.882	2.504
SplitFI(Build)	0.134	0.15	0.118

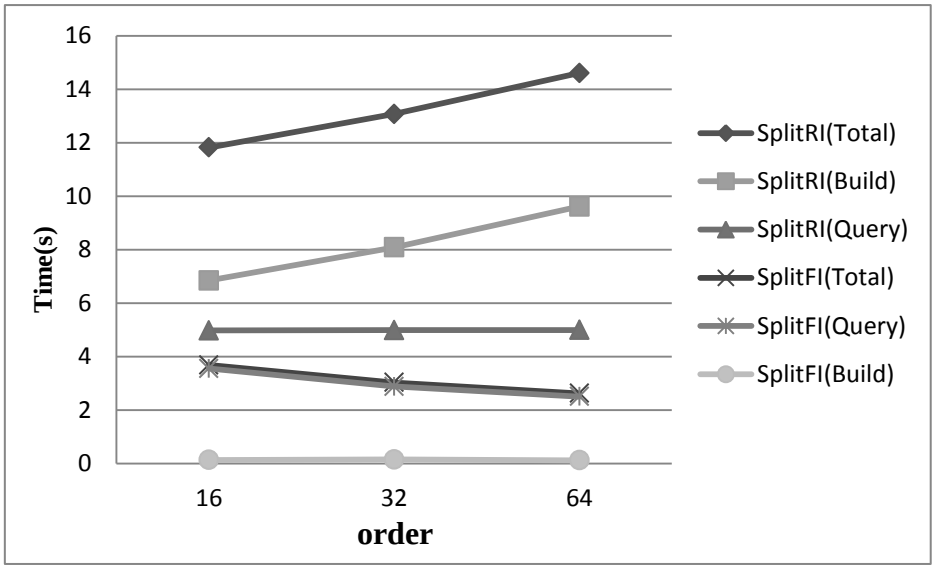


圖 5-9　真實資料集中改變 order 值之整體影響

最後，我們根據實驗結果，說明四種方法搭配的索引以及 order，BaseFI 方法索引建立與查詢在 Online 階段執行，因此選擇整體時間較少的 SplitFI 索引，而 BaseRI 方法索引建立可以在 Offline 階段執行，但查詢需在 Online 階段，因此選擇查詢較好的 SplitRI 實作。SplitFI 整體時間比 SplitRI 好，而 CloudFRI 和 CloudFPI 作法的淹水路口查詢模組需在 Online 階段執行，因此選擇整體時間較少的 SplitFI 索引。鄰近點查詢模組需使用道路或路口索引尋找，該索引可在 Offline 建立，而 Split 方式查詢時間比 Sorted 方式好，因此採用 Split 方式建立索引。至於 order 部分，SplitRI 查詢時間不隨 order 增加而明顯上升，因此 BaseRI 以及 CloudFRI、CloudFPI 的鄰近點查詢就隨機選擇 order 值 64。而 BaseFI 建立時間不隨著 order 數值增加而明顯改變，查詢時間則隨著 order 數值增加而趨近線性下降，因此 BaseFI 以及 CloudFRI、CloudFPI 的淹水路口查詢就選擇查詢效率最好的 order，也就是 order 值 64。

5.5 threshold 效率和成功率之實驗

本實驗，固定路網資料為台北市，探討利用不同 threshold 值，CloudFPI 方法 NearPoint (對應至鄰近點篩選模組)、GSP(GSP 演算法)、整體執行時間以及路徑規劃成功率。在此使用 CloudFPI 而非 CloudFRI 是因為 CloudFPI 方法整體時間較 CloudFRI 方法佳，因此利用 CloudFPI 方法比較。為了能夠比較成功規劃出路徑，我們從 655 個淹水區域真實資料當中選出分布在左下、中間以及右上的 150 個淹水區域。成功率的實驗方式為選擇 25 組起迄資料，記錄每組是否成功規畫出路徑，如果一開始就成功規劃出路徑，用'1'表示，否則需修正路徑，但次數不能超過 3 次，如果超過，表示不成功，用'00'表示，反之成功，用'01'表示，最後將一開始就能成功規劃路徑的個數以及修正後能成功規劃路徑的個數皆除以 25，分別得到修正前、修正後的成功率。效率的實驗方式為針對上述 25 組起迄資料，記錄每組 NearPoint、GSP 以及整體執行時間，然後取這 25 組平均，得到 CloudFPI 方法 NearPoint、GSP 以及整體平均執行時間。注意到，若無法成功規劃出路徑，整體時間為修正 3 次總共所需之時間。以下將 threshold 區分為臨近距離值和合併距離值進行實驗。

5.5.1 臨近距離值

(1) 這次實驗，我們探討不同臨近距離值，也就是圖 4-11 中的參數 threshold，比較 CloudFPI 方法 NearPoint(鄰近點篩選模組)、GSP(GSP 演算法)以及整體執行時間，結果如圖 5-10。我們可以發現隨著 threshold 增加，NearPoint 呈線性增加，因為 threshold 值越大，涵蓋範圍越多，因此需要比對更多路口點，而 GSP 成非線性增加。因此整體時間成非線性增加。

表 5-8　CloudFPI 時間之改變臨近距離值實驗紀錄

	0.5km	1km	1.5km
Total	0.9928	1.594	3.853
GSP	0.6078	1.16	3.456
NearPoint	0.0174	0.064	0.141

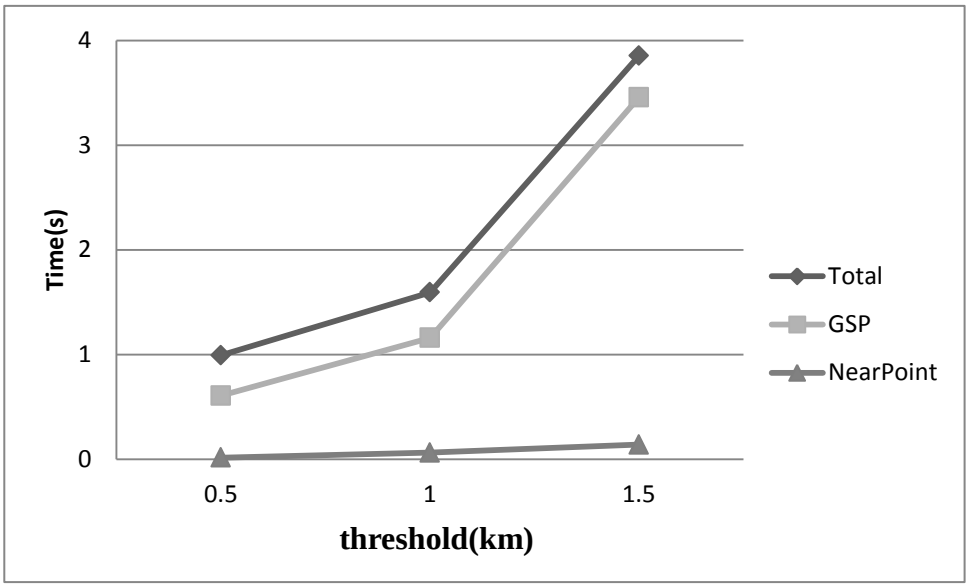


圖 5-10　CloudFPI 時間之改變 threshold 實驗紀錄

(2) 這次實驗，我們探討不同臨近距離值，比較 CloudFPI 方法迴避淹水區域路徑的成功率，結果如圖 5-11，其中 **CloudFPI** 表示修正後成功率，**CloudFPI-** 表示修正前成功率。由圖我們得到幾個結論

- **CloudFPI** 成功率比 **CloudFPI-**高。
- 隨著 threshold 值增加，成功率也增加。

綜合圖 5-10 和圖 5-11，臨近距離值為 1 與臨近距離值為 1.5 的成功率(修正後)都很高，但在整體效率方面，臨近距離值為 1 卻比臨近距離值為 1.5 高許多，因此最後臨近距離值採用值為 1 進行比較。

表 5-9　規劃路徑成功率之改變 threshold 實驗紀錄

	0.5km	1km	1.5km
CloudFPI	0.68	0.8	0.96
CloudFPI-	0.56	0.68	0.72

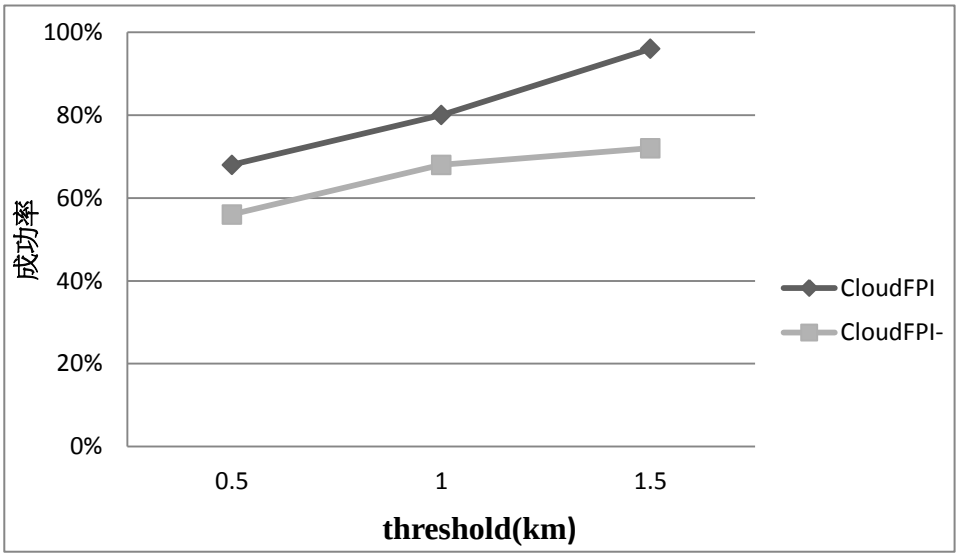


圖 5-11　規劃路徑成功率之改變 threshold 實驗紀錄

5.5.2　合併距離值

(1) 這次實驗，探討不同合併距離值之影響，也就是圖 4-4 中的參數 t，比較 CloudFPI 方法 NearPoint(鄰近點篩選模組)、GSP(GSP 演算法)以及整體時間，結果如圖 5-12。我們可以發現隨著 threshold 增加，NearPoint 以及 GSP 非線性減少，因此整體時間非線性減少，因為 threshold 值越大，點被合併的越多，合併出來的道路、淹水路口個數越少，需尋找鄰近點的淹水路口越少。

表 5-10　查詢鄰近點效率之改變 threshold 值合併實驗紀錄

	50m	100m	150m	200m
Total	2.746	2.687	2.193	1.594
GSP	2.241	2.21	1.741	1.16
NearPoint	0.154	0.116	0.087	0.064

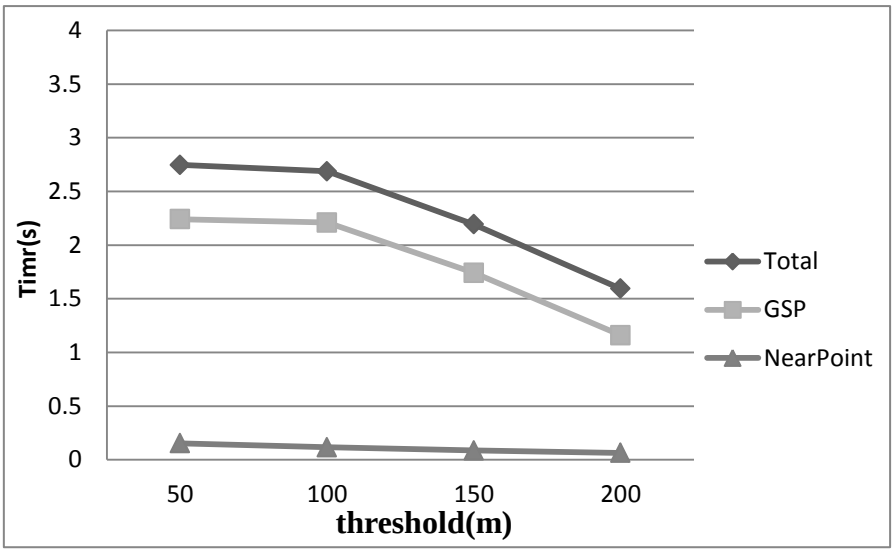


圖 5-12　查詢鄰近點效率之改變 threshold 影響

(2) 這次實驗，探討不同合併距離值之影響，比較 CloudFPI 方法迴避淹水區域路徑的成功率，結果如圖 5-13，由圖得知，一開始成功率隨著 threshold 增加而降低，但是到 threshold 值為 150、200 時增加，主要是因為這 25 組起迄資料當中有幾組在 threshold 值為 100 時反而無法成功規劃出路徑，但 threshold 值為 150、200 時卻可以一次就成功規劃出路徑，導致整體成功率又上升，如表 5-11。

表 5-11　起迄資料成功率之改變合併值實驗紀錄

	50m	100m	150m	200m
121.522439,25.031594、121.578907,25.058233	01	00	1	1
121.517748,25.039436、121.532955,25.078093	1	00	1	1

綜合而言，因為合併值的大小對於成功率沒有絕對影響，但合併距離值為 200 的執行效率最好，因此最後合併距離值採用值為 200 進行比較。

表 5-12　規劃路徑成功率之改變合併值實驗紀錄

	50m	100m	150m	200m
CloudFPI	0.96	0.8	0.8	0.8
CloudFPI-	0.6	0.56	0.6	0.68

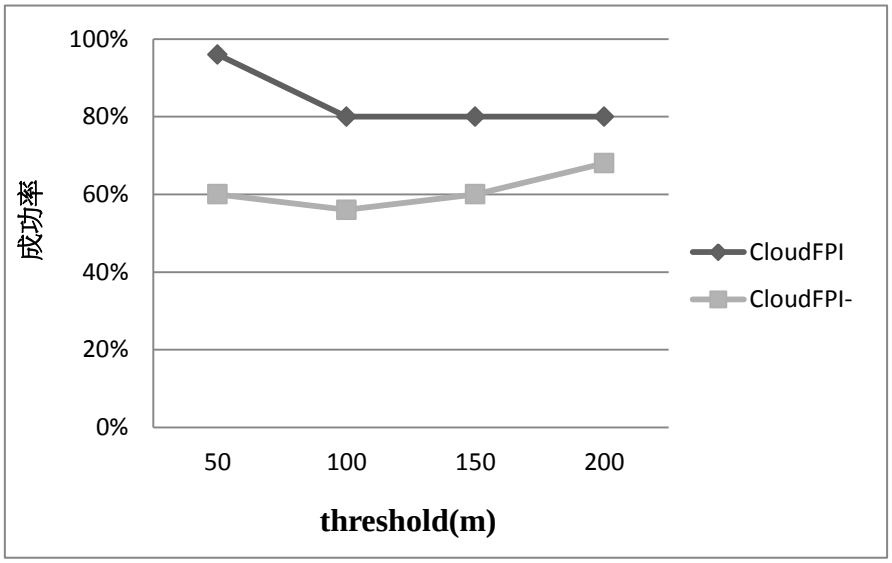


圖 5-13　規劃路徑成功率之改變 threshold 值合併實驗紀錄

5.6 路網圖大小之效率實驗

本實驗，我們探討不同路網圖大小之影響，包含基隆市(1600 條道路)、新竹市(11526 條道路)和台北市(21652 條道路)，為了盡可能讓每個路網圖淹水個數一致，固定真實資料為基隆 148 個，從新竹 330 個以及台北 655 個淹水區域分別挑選分布在左下角、右上角以及中間的 150 個淹水區域，比較 BaseRI、BaseFI、CloudFRI、CloudFPI 四個方法不同模組以及整體時間，實驗方式為表 5-4 的 10 組起迄資料，包含距離較遠、中和較近，記錄每組起迄執行時間，然後取這 10 組平均。

(1) 這次實驗，我們探討 BaseRI 載入道路索引、BaseFI 建立淹水索引以及 BaseRI、BaseFI 查詢與路徑規劃效率，BaseRI 之所以用 load 路索引而不直接建立路索引，主要原因是路索引可以先在 Offline 先建立，Online 直接 load 建立好的路索引，這樣避免 Online 花過多時間建立，實驗結果如圖 5-14。由圖可觀察出 BaseRI 方法查詢時間隨著路網圖數量增加而線性上升。BaseFI 方法查詢時間隨著路網圖數量增加而非線性緩慢（sublinaer）上升，主要原因是受道路與淹水判斷次數影響(也就是根據圖 3-8 演算法，只要得知道路與其中一塊淹水相交後，就不需要再與其它淹水區塊判斷是否相交)。路徑規畫時間隨著路網圖數量增加而趨近線性上升，而這兩個方法所花的時間基本上一樣。不過，雖然道路索引可以事先建立，但因為其結構龐大，所以其 load 的時間比淹水索引 online 建立所需之時間更長，而影響到整體時間。但整體來講還是路徑規劃所需時間最長。

表 5-13　BaseRI、BaseFI 模組實驗紀錄

	1600	11526	21652
BaseRIDijk	0.641	5.369	8.192
BaseFIDijk	0.636	5.296	8.046
BaseRIQuery	0.606	1.439	2.123
BaseFIQuery	0.255	0.991	1.047
BaseRILoad	0.085	0.665	1.094
BaseFIBuild	0.016	0.017	0.012

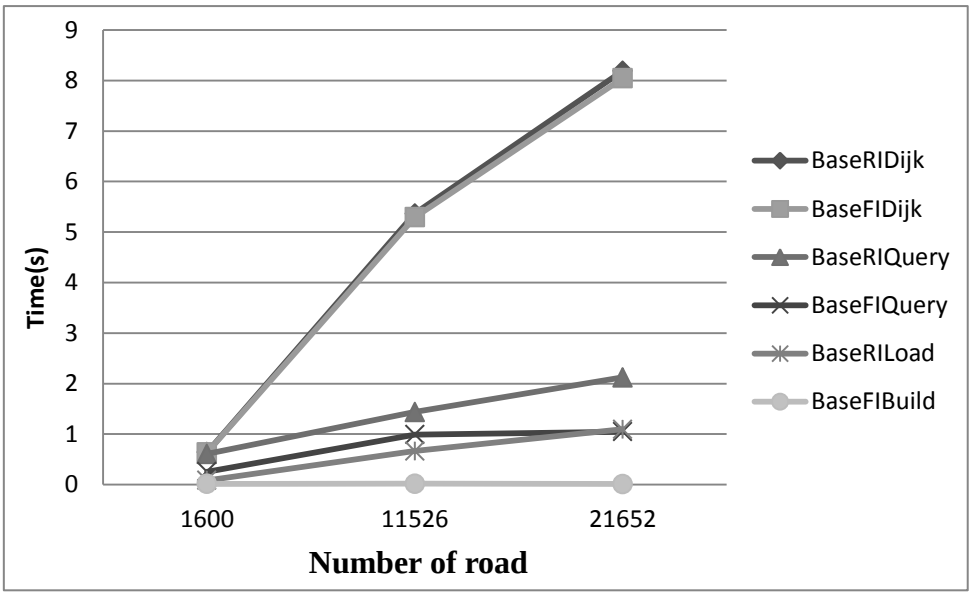


圖 5-14　Baseline 模組實驗之影響

(2) 這次實驗，我們固定臨近距離值為 1(km)，合併距離值為 200(m)，探討 CloudFRI、CloudFPI 方法「淹水路口查詢模組」、「鄰近點篩選模組」以及「GSP 演算法」的效率，結果如圖 5-15。其中 FIP 對應到「淹水路口查詢模組」，CloudFRINP 則使用 CloudFRI 方法執行「鄰近點篩選模組」，CloudFPINP 使用 CloudFRI 方法，GSP 則對應「GSP 演算法」。就 FIP 而言，其效率和道路個數沒有絕對關係。隨著道路個數增加，CloudFRINP 和 CloudFPINP 約略成線性增加，顯示我們的索引有效的控制查詢時間。另一方面，GSP 則快速增加，而且所需時間比 CloudFPINP 大約多出 12.44 倍。

表 5-14　真實資料集中改變路網大小之實驗紀錄

	1600	11526	21652
GSP	0.252	0.344	0.574
FIP	0.086	0.127	0.041
CloudFRINP	0.011	0.04	0.078
CloudFPINP	0.01	0.026	0.058

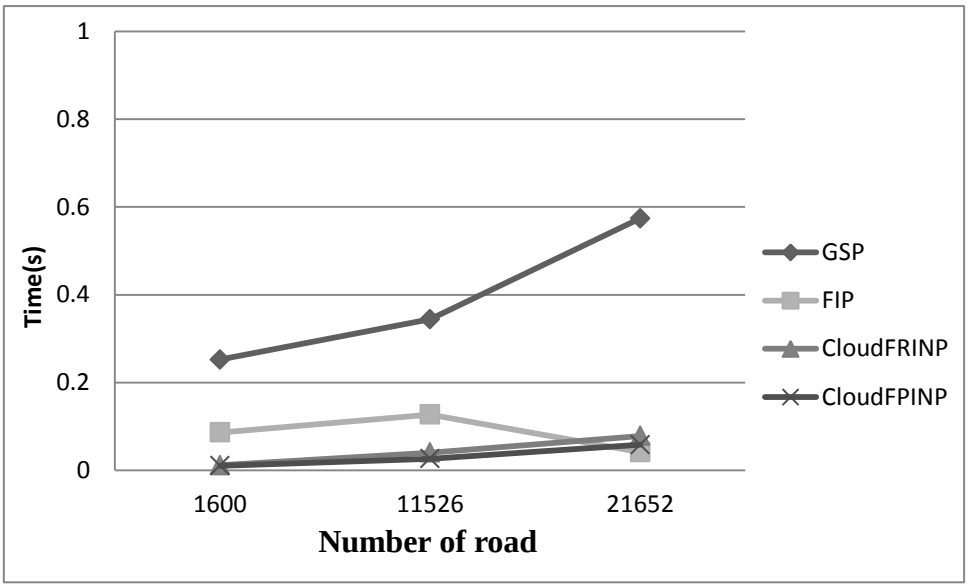


圖 5-15　真實資料集中改變路網大小之影響

(3) 最後，我們設定臨近距離值為 1(km)，合併距離值為 200(m)，比較 BaseRI、BaseFI、CloudFRI、CloudPI 四個方法整體時間。前二方法的整體時間為表 5-13 三個模組時間的總和，後二方法的時間除了如表 5-14 三個模組的時間，還加上「雲端路徑查詢模組」。結果如圖 5-16。由圖可觀察出以下幾種結論

- BaseRI、BaseFI 方法整體時間隨著路網圖數量增加而 sublinear 上升。
- BaseFI 方法的整體時間比 BaseRI 方法還快，主要是因為道路個數比淹水個數多，建立出來的道路索引節點個數比淹水索引多，因此 BaseRI 在 load 路索引時間比 BaseFI 建立索引慢，查詢時間差不多，所以整體時間 BaseRI 比 BaseFI 慢。
- CloudFRI、CloudFPI 方法整體時間隨著路網圖數量增加而線性上升，但增加時間不多。
- 在 CloudFPI 方法的整體時間比 CloudFRI 方法平均快 1.04 倍，主要是因為點索引矩形重疊程度比路索引小，在尋找鄰近節點效率點索引比路索引好，因而影響整體時間。
- CloudFRI、CloudPI 兩個方法明顯比 BaseRI、BaseFI 兩個方法快，例如

當道路數量 1600 筆、148 個淹水區域，CloudFRI 整體的效率更是 BaseRI 的 2.24 倍，而道路數量 21652 筆、150 個淹水區域，CloudFRI 整體的效率更是 BaseRI 的 10.81 倍。

表 5-15　真實資料集中不同大小路網圖整體時間之實驗紀錄

	1600	11526	21652
BaseRI	1.333	7.472	11.409
BaseFI	0.907	6.305	9.104
CloudFRI	0.594	0.81	1.055
CloudFPI	0.575	0.79	0.991

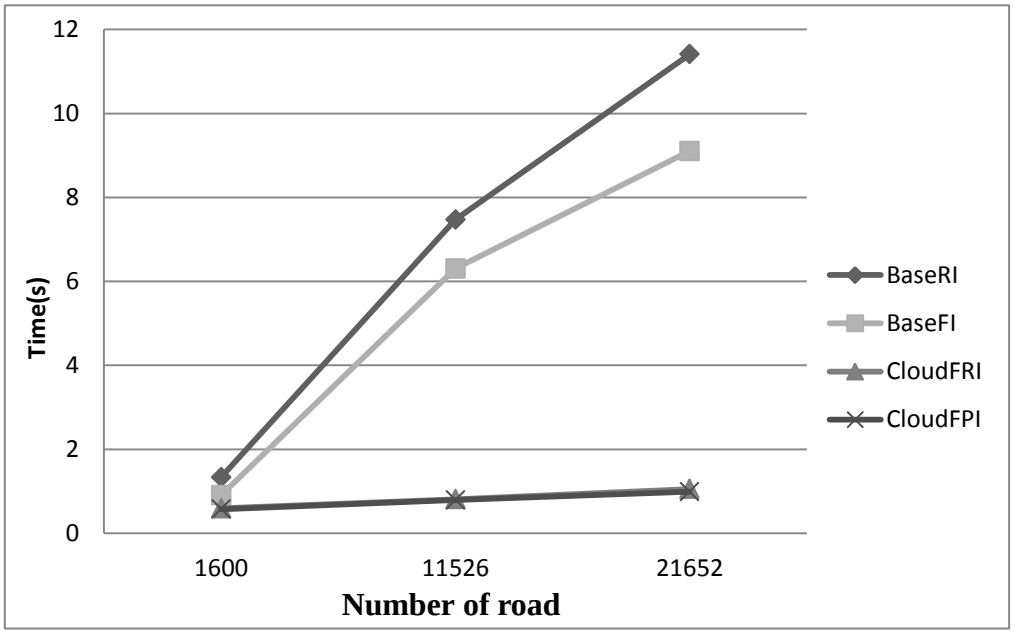


圖 5-16　真實資料集中不同大小路網圖整體時間之影響

5.7 淹水區域數量和分布之實驗

本實驗，固定路網為台北市，臨近距離值為 1(km)，合併距離值為 200(m)，探討著重於改變淹水區域資料集的數量，範圍限制在台北地區內產生 100、150、200 個不同分布淹水區域資料集。為了能成功規劃出路徑，我們針對隨機、高斯以及群聚三個分布個別選擇 10 組 OD(和表 5-4 不盡相同)，記錄每組起迄執行時間，然後取平均。

(1) 這次實驗，探討改變隨機分布淹水區域個數之影響，結果如圖 5-17。隨著淹水區域個數增加，四個方法的執行時間成線性增加，這是因為與道路做空間交及次數變多，查詢時間也會增加，因而影響整體時間。而以 CloudFPI 效率較佳，當達到 200 個淹水區域時，CloudFPI 方法的效率為 BaseFI 的 8.04 倍。

表 5-16　隨機分布資料集中改變淹水區域數量之實驗紀錄

	100	150	200
BaseRI	9.986	10.212	10.577
BaseFI	8.837	9.198	9.537
CloudFRI	1.09	1.123	1.325
CloudFPI	1.064	1.079	1.186

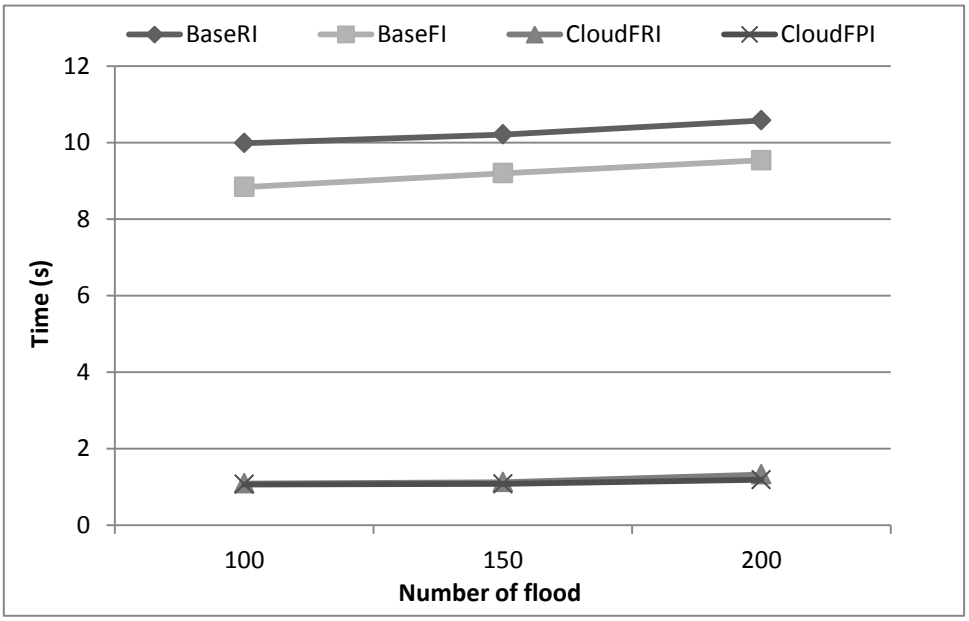


圖 5-17　隨機分布資料集中改變淹水區域數量之影響

(2) 這次實驗僅在於改變淹水區域分佈為高斯分佈的資料集。如圖 5-18，BaseRI、BaseFI 花費的時間隨著淹水區域個數增加而線性上升，CloudFRI、CloudFPI 花費時間隨著淹水區域個數增加而趨近線性上升，而以 CloudFPI 效率較佳，當達到 200 個淹水區域時，CloudFPI 方法的效率為 BaseFI 方法的 5.83 倍。

表 5-17　高斯分布資料集中改變淹水區域數量之實驗紀錄

	100	150	200
BaseRI	10.076	10.565	10.704
BaseFI	8.964	9.323	9.554
CloudFRI	1.177	1.194	1.696
CloudFPI	1.169	1.188	1.638

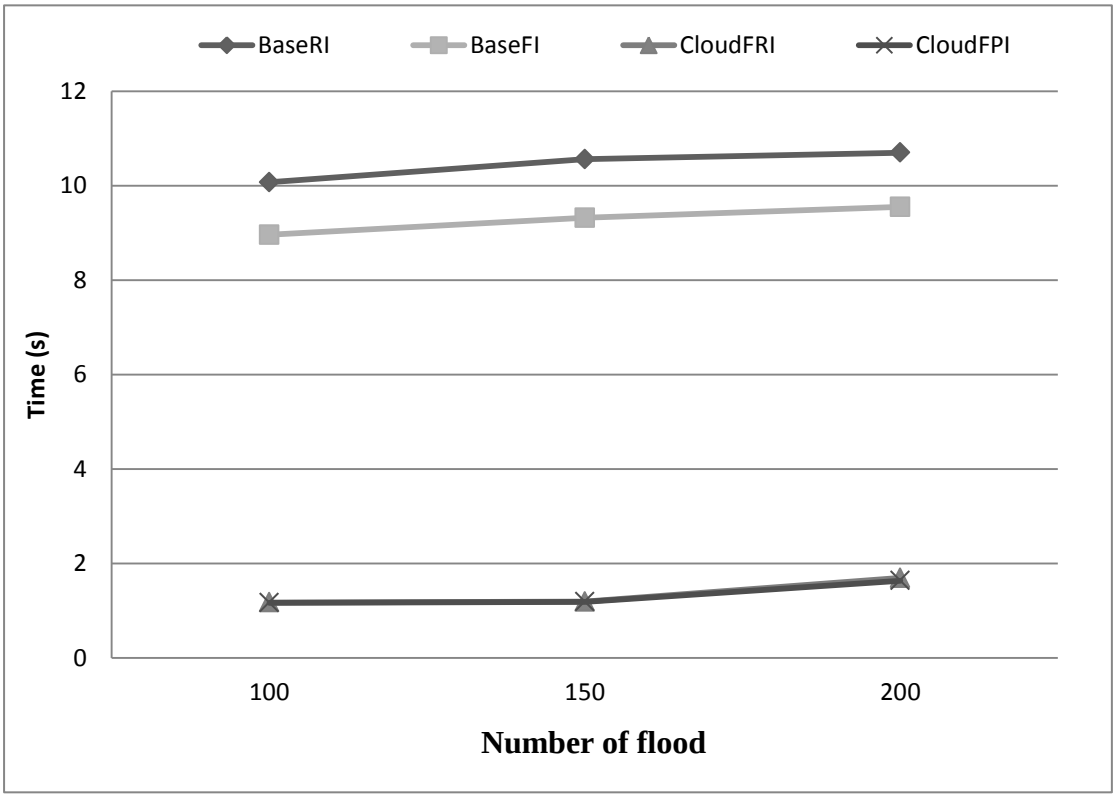


圖 5-18　高斯分布資料集中改變淹水區域數量之影響

(3) 這次實驗僅在於改變淹水區域分佈為群聚分佈的資料集。如圖 5-19，四個方法花費的時間隨著淹水區域個數增加而線性上升，而以 CloudFPI 效率較佳，當淹水區域達到 200 個時，CloudFPI 方法的效率為 BaseFI 方法的 7.2 倍。

表 5-18　群聚分布資料集中改變淹水區域數量之實驗紀錄

	100	150	200
BaseRI	10.312	10.542	10.805
BaseFI	8.926	9.253	9.507
CloudFRI	1.317	1.374	1.473
CloudFPI	1.241	1.278	1.319

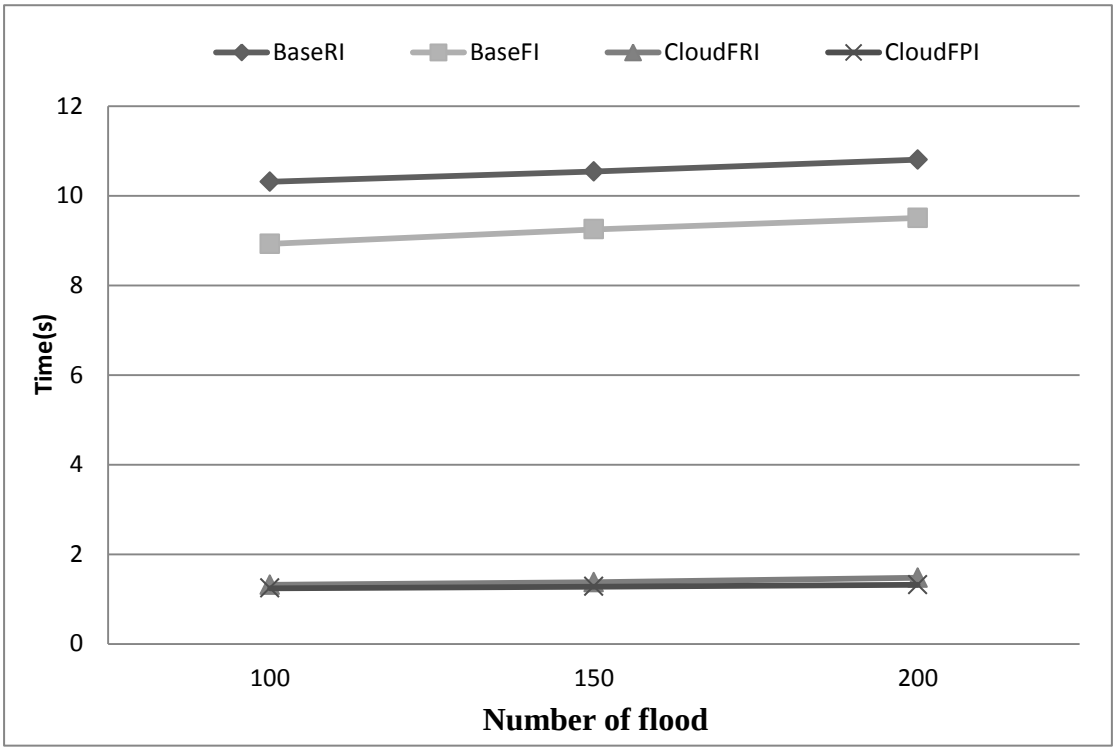


圖 5-19　群聚分布資料集中改變淹水區域數量之影響

從本節的實驗看來，淹水區域的數量和分布對所有方法執行時間只有微小的影響。

5.8 規劃路徑長度之實驗

本實驗，我們比較四方法最後輸出避開淹水區域的路徑長度。固定台北市 150 個淹水區域、選擇台北市的路網圖(12419 個路口點及 15874 個路)，記錄表 5-4 十組查詢經過四個方法規劃出來的長度，如表 5-19，並將十組查詢結果平均，如表 5-20。由表 5-19 得知 CloudFRI、CloudFPI 規劃出來平均長度比 BaseRI、BaseFI 還要長 34%，主要原因是 Google 做路徑規劃時會優先選擇時間較短的路徑，也就是越快抵達終點，接著才會考量到路徑長度，例如輸入 Q8 起迄，圖 5-20 表示 CloudFRI、CloudFPI 方法規劃出迴避淹水路徑，圖 5-21 表示 BaseRI、BaseFI 方法規劃出迴避淹水路徑，我們可以發現圖 5-20 經過快速道路(市民大道)的路徑長度比圖 5-21 經過平面道路長，但卻可以較快抵達目的地。

表 5-19　每組查詢長度實驗紀錄

	Q1	Q2	Q3	Q4	Q5
BaseRI、BaseFI	7.087km	3.464 km	11.01 km	7.976 km	6.4 km
CloudFRI、CloudFPI	11.654 km	5.044 km	15.517 km	9.761 km	10.513km
	Q6	Q7	Q8	Q9	Q10
BaseRI、BaseFI	12.7 km	4.61 km	11.367 km	1.85 km	12.345 km
CloudFRI、CloudFPI	15.558 km	3.681 km	14.334 km	4.322 km	15.241 km

表 5-20　平均長度實驗紀錄

	平均長度
BaseRI、BaseFI	7.8809 km
CloudFRI、CloudFPI	10.5625 km

圖 5-20　CloudFRI、CloudFPI 方法規劃出迴避淹水路徑

圖 5-21　BaseRI、BaseFI 方法規劃出迴避淹水路徑

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四個方法。首先，提出了利用道路資料建索引，淹水區域與節點作空間相交的 BaseRI 方法，其次提出利用淹水區塊建索引，道路資料與節點作空間相交的 BaseFI，兩者方法皆會剔除淹水的道路，並透過 Dijkstra 尋找最短路徑。為了避免因為資料量提升而導致效率下降，我們也提出了 CloudFRI、CloudFPI 方法，兩個方法皆透過『雲端路徑查詢模組』規劃一條最短路徑，並利用淹水區塊索引，將淹水區塊與路徑作空間相交，將經過淹水區域道路的路口點利用『鄰近點篩選模組』尋找鄰近點集合，最後，利用 GSP 演算法計算起點經過淹水路口鄰近點到終點的最短路徑，並從每個淹水路口鄰近點集合選出代表點當作 waypoint，然後將起點、waypoint 以及目的地再次透過『雲端路徑查詢模組』規劃一條最短路徑，若不幸經過淹水，則刪除 waypoint，重新規畫路徑，直到路徑未經過淹水或重新規畫超過 n，n 表示重新規劃次數，而兩個方法差別在於 CloudFRI 使用道路索引尋找，CloudFPI 則使用路口點索引。

在真實數據資料的實驗下，CloudFRI、CloudFPI 方法整體花費時間比 BaseRI、BaseFI 方法少。隨著道路數量增加，四個方法成線性成長，但 BaseRI、BaseFI 方法增加趨勢更明顯，例如根據表 5-15，在道路數量 1600 筆、150 個淹水區域中，CloudFPI 整體的效率是 BaseFI 的 1.57 倍，而道路數量 21652 筆、150 個淹水區域，CloudFPI 整體的效率是 BaseFI 的 9.18 倍。

在成功率方面，根據表 5-9，隨著鄰近距離值增加而非線性增加，而合併距離值為 200 時，修正前的成功率最高，但合併距離值為 50 時，修正後成功率最高。而路徑長度方面，根據表 5-20，CloudFRI 和 CloudFPI 平均長度比 BaseRI 和 BaseFI 多 34%。

本論文未來的研究方向，希望針對 CloudFRI 和 CloudFPI 方法，提升路徑規劃成功率，減少規劃出來路徑長度，並同時保持高效率。

附錄 A 建立道路索引程式

演算法名稱：**BuildRoadRStarTree** 輸入: Road // Road 表示道路集合，用於建立 R*-tree order //表示節點所能存放個數 變數: Leaf //表示 Leaf 節點 Root //表示根節點指標 輸出: RoadIndex //道路索引
L01 Root = Leaf L02 **for each** r of Road L03 Leaf = chooseLeaf(root, r) L04 Leaf.push(r) L05 **If**(Leaf.Entry.size() > order) L06 Split(leaf, newnode) L07 AdjustTree(leaf, newnode) L08 **Else** L09 AdjustContent(leaf) L10 **End If** L11 **End for** L12 **return** RoadIndex

L01 假設一開始 Leaf 為根節點，L02-L11 利用路的資料建立 RTree，其中 L03 選擇要新增的 Leaf 節點，L04 新增至選擇 leaf 節點，L05 判斷 leaf 節點 Entry 個數是否超過 order，L06 如果超過就 split 此節點，L07 由於 split 除了本身節點外還會有額外節點，因此利用 AdjustTree 函數將額外節點新增至索引，並調整樹的結構，L09 如果路個數不超過 order，只需利用 AdjustContent 函數調整節點內容，也就是說當道路新增至 leaf 時，leaf 節點 MBR 會改變，因此需修改 leaf 所有祖先 MBR，L12 最後回傳路的索引。

接下來，我們介紹 Split 演算法，L01 起始 axis、Distribute 為 null 以及計數 i 為 0，其中 axis 代表用以分割節點的軸，如果為'X'，表示根據 x 軸分割，如果

為'Y'，表示根據 y 軸分割。Distribute 為一分布，其表示方式為兩個節點內之 entry 個數，其格式為<Node1.entry.size, Node2.entry.size>，每個分布節點個數至少為 order 一半。L02 利用 chooseAxis 函數選擇要分割的軸，該函數先將 entry 按照 x 軸和 y 軸分別排序(以 entry.MBR 右上點)，然後每個軸分別計算每個分布的 margin-value（邊長總合），並將其結果相加，分別得到 Sx 和 Sy，若 Sx 小於 Sy，則分割軸為 X，反之為 Y。然後將 entry 按照分割軸排序。L03 利用 chooseDistribute 計算每個分布其兩個包圍矩形面積(area-value)的總和，選擇面積總和最小的分布。L04-L07 將該分布中，將分隔軸值較大的那些 entry，依序移至新節點中。

演算法名稱：**Split**
輸入: node //表示被 Split 的節點 newnode //表示從 node 節點 Split 出來的新節點，因此一開始沒有任何物件 order //表示節點最多所能存放個數
變數: axis //根據所選出來的軸 Split 節點 Distribute//面積總和最小的分布
L01 **Initialize** axis←NULL,Distribute←NULL, i←0
L02 axis←chooseAxis(node.entry, order)
L03 Distribute= chooseDistribute(node.entry, order)
L04 **while**(i < Distribute.second) //Distribute.second表示該分布後半段的entry個數
L05 newnode.object.push(node.entry.end())
L06 node.object.delete(node.entry.end())
L07 i←i+1
L07 **End while**

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

[BKSS90] Norbert Beckmann, Hans-Peter begel Ralf Schneider, Bernhard Seege, “The R*-tree: an efficient and robust access method for points and rectangles”, Proceedings of the ACM SIGMOD international conference on Management of data, 1990

[CC15] Dong-Wan Choi, Chin-Wan Chung, “Nearest neighborhood search in spatial databases”, Proceedings of the ICDE conference, 2015

[EW59] Edsger Wybe Dijkstra, “A note on two problems in connexion with graphs ”, Numerische Mathematik, Vol. 1, pp. 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

[李 15] 李承翰,“迴避淹水區域之快速路徑規劃研究”, 國立臺灣海洋大學資訊工程研究所碩士論文, 2015

[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

[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

[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

[國 13] 『國家災害防救科技中心』, “一日暴雨 600mm 淹水潛勢圖”, http://satis.ncdr.nat.gov.tw/, 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