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

淹水區間分割方法之研究

The Study on the Method of Partitioning a Flooded Spot

研究生：李文瀚 撰

中華民國 106 年 05 月

淹水區間分割方法之研究

The Study on the Method of Partitioning a Flooded Spot

研究生：李文瀚　　Student：Wen-Han Lee

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
May 2017
Keelung, Taiwan, Republic of China

中華民國 106 年 05 月

摘要

當大雨來臨，我們希望規劃一條從起點至目的地的路線，且能避開淹水的路段。之前已經有研究提出此類規劃路徑的系統，但該系統以手動的方式將一個淹水區間分割為固定大小的矩形，缺點在於分割出的淹水區塊佔有淹水區間程度極小仍會被輸出，可能會在規劃路徑時，造成可以行走的路徑被剔除，花費更多時間繞道而行。

為了降低淹水區塊與道路之間相交的誤判，本論文主要是根據其所提出的架構，討論分割淹水區間的三種方法及兩種對照方法，將淹水區間分割為大小適中的矩形。第一種是 **BaseFS** 方法，是最簡單直覺的作法，我們將淹水區間以固定邊長 CellEdge 的矩形來分割。第二種是 **BaseMIR** 方法，先透過軸上和淹水區間相交且最接近中心點的四個軸點來取矩形後，再依序將其餘區間分割為等分的區塊。第三種是 **Quad** 方法，透過門檻值來控制淹水區塊的分割或輸出，其分割方式是將淹水區間分割為四個象限，分別遞迴的作法。為了對照上述三種分割方法，我們進一步提出以下兩種對照方法。第一種是將淹水區間直接取最小包圍矩形的 **MBR** 方法，第二種是將淹水區間點序列鄰近的 2 個點連成一個線段來取 MBR 的 **SegmentRtree** 方法。

我們實作上述五種方法，並進行一系列的實驗下，比較在不同資料集中規劃出的路徑長度及效率。實驗結果顯示，**SegmentRtree** 方法規劃的迴避淹水路徑最短，但是在規劃路徑時需處理很多不能走的道路，對效率造成影響，在實務上不可行。而 **Quad** 方法在其餘方法中規劃迴避淹水的路徑最短，關鍵在於分割出的淹水區塊總面積趨近原始的淹水區間面積，降低了誤判的機率，而執行效率也在可接受的範圍，因此整體而言 **Quad** 方法為較好的淹水區間分割方法。

關鍵詞：圖形分割、淹水區間、最短路徑規劃。

Abstract

When the heavy rain comes, we wish to plan a path which avoids flooded spots. Although there exists such a system, it relies on humans to manually partition a flooded spot into many blocks of fixed sizes. The defect of such a system is that it might output many small-sized blocks. When we are planning the path, available paths might be removed due to intersecting with such small blocks, and we will spend more time on detouring.

In order to reduce the errors as described above, this thesis utilizes the existing system to plan a path, and mainly discusses how to partition flooded spots into proper-sized blocks. We propose three methods. The first method is the most simple and intuitive one, which is called BaseFS and partitions a flooded spot into a set of fixed-sized blocks. The second method is called BaseMIR. It first identifies the "minimum interpoint rectangle" by using the interpoints on the four axes, and then partitions the remaining areas based on the BaseFS method. The third method is called Quad. It will partition a flooded spot into four quadrants and use the thresholds to determine if the partitioning should proceed recursively or stop. In order to contrast the three partitioning methods above, we propose two additional methods. The first method is called MBR, which directly outputs the minimum bounding rectangle (MBR) of each flooded spot. The second method is called SegmentRtree, where two adjacent points on the border of a flooded spot will form an MBR.

We have implemented the above five methods and performed a series of experiments to compare the lengths of planned paths and efficiency based on different datasets. Experimental results show that the SegmentRtree method usually obtains the shortest path, but it needs to process a lot of roads which cannot be passed and requires a lot of time in path planning. This makes it infeasible in the real world. In contrast, the Quad method can usually get the second shortest path since the total area of its flooded blocks nears the original flooded area. It can also get the path within an acceptable period of time. Therefore, it is the best partitioning method overall.

Keywords：Graph Partitioning, Flooded Spot, Shortest Path Planning.

誌謝

首先，特別感謝指導教授張雅惠博士，給予本論文許多建議，且在實作與撰寫論文期間不時地共同討論，使學生能從老師給予的建議及督導中，學習到學術上謹慎的態度及精神，並順利的完成論文。

除此之外，感謝林川傑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，也感謝承翰學長和蔚齊學長提供本論文意見與幫助，使論文更趨近完善。

最後，也要感謝實驗室的韋錫學長、思彥學長、毓城學長、我的同學怡淵和詩盈、學弟妹們以及在海洋大學認識的朋友們。共同度過研究所的時期，無論是一同用餐、運動或準備論文，都是一生中美好的回憶。更要感謝我最愛的家人和女友瑜欣，總是在我身旁給予我無限的支持，讓我能無憂無慮的專心在論文研究上，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，我們首先說明本論文的研究動機與目的，接著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

台灣地區長年降雨豐沛，但分布極不均勻的降雨量，常造成大雨宣洩不及釀成災害。在之前的研究中[李 15]，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 1-1。[洪 16]接續前一論文之 Baseline 作法，提出以淹水區塊建立索引，然後利用該索引找出與淹水區塊相交的道路，將其從路網中移除，再將其餘道路由 Dijkstra 演算法進行最短路徑規劃。該系統所使用的淹水區間基本上來自[劉 14]洪氾系統模組，該模組利用氣象資料和水利公式等推算某塊矩形所在位置是否會淹水。雖然[洪 16]在真實淹水資料的部份有觀察到一些不規則的淹水區間，但是只以手動的方式將一個淹水區間分割為固定大小的矩形。此方法的缺點在於分割出的淹水區塊佔有淹水區間程度極小仍會被輸出，可能會在規劃路徑時，造成可以行走的路徑被剔除，花費更多時間繞道而行。

圖 1-1　避開淹水區域之路徑規劃

我們進一步討論所分割的淹水區塊與道路的相交情況為何會影響規劃出來的結果。針對不規則的淹水區間，最簡單直覺的做法就是求取其最小邊界矩形(MBR)，如下圖 1-2 所示，其中不規則的圖形代表淹水區間，粗外框代表淹水區間的 MBR，而粗曲線代表道路 A 和 B。假設從起點可以透過道路 A 或道路 B 到達迄點，可以看到道路 A 並沒有實際與淹水區間相交，但是若直接以淹水區間的 MBR 來判斷，道路 A 會被誤判為不能行走，因此所規劃的路徑長度會從 1 公里增加為 1.3 公里。

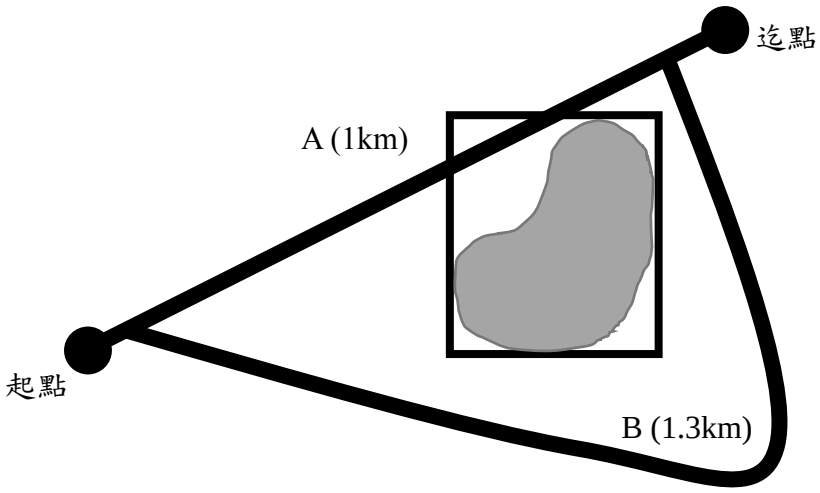


圖 1-2　以 MBR 做為淹水區塊示意圖

為了降低上述的誤判，本論文提出淹水區間分割方法之研究，也就是以將淹水區間分割為較小的區塊做為出發點，使分割出的淹水區塊總面積能貼近原始淹水區間的面積。並且希望能以程式自動化產生，不需要人工手動分割。

1.2　研究方法與貢獻

此研究討論如何將淹水區間分割為較精準的淹水區塊，而降低在規劃迴避淹水路徑時造成誤判的情況，以便輸出較短的路徑。本論文提出以下三種方法：第一種根據[洪 16]Baseline 的想法，將淹水區間分割為固定大小的區塊，只輸出與淹水區間相交的淹水區塊，稱作 **BaseFS** 方法。第二種是以最小交點矩形(MIR)來實作的作法，是利用軸上和淹水區間相交且距離中心點最近的 4 個點來取矩形，並將剩餘的淹水區間分為 4 個區塊後，以 **BaseFS** 方法分割，稱作 **BaseMIR** 方法。第三種是透過門檻值來控制淹水區塊的分割或輸出，其分割方式是將淹水區間分割為 4 個象限，分別遞迴的作法，稱作 **Quad** 方法。

我們完成上述三種方法的實作，並於實驗階段進行路徑長度及效率的實驗研究，比較其各自的優缺點。結論發現以上三種方法中，**BaseFS** 方法在 **Offline** 階段分割淹水區間模組的效率最佳，**BaseMIR** 方法在 **Online** 階段移除淹水道路模組的效率最佳，而 **Quad** 方法所規劃迴避淹水路徑的長度最短且效率僅略輸其他方法，因此整體而言 **Quad** 方法為最適合用來分割淹水區間的方法。

1.3　相關研究

我們首先討論有關分割圖形的相關議題。在圖學或影像處理也會因為光線追蹤(Ray Tracing)而需要分割圖形，而光線追蹤是計算機圖形學領域的經典演算法，它主要是模擬現實現世界中的成像原理，對於每條光線生成基本參數，然後利用

圖形分割結構與此條光線進行碰撞檢測，最後在碰撞點對分割區塊進行渲染。這些結構各有特點，需要根據所處理圖形的特點或繪製任務選擇對應的結構分割，目的皆是將圖形以階層化的樹狀結構分割，可使得空白區域能以較大的節點表示，進而減少碰撞檢測所花費的時間。

常見的分割結構包含 Kd-trees[ZHWG08]、octrees[WG92]及 grids[WIKKP06]。而光線追蹤普遍是以 Kd-trees 來實作，雖然效果很好但必須花很多時間來建立，也花費很多記憶體空間。因此論文[MVJ14]提出光線追蹤在近代的多核心 GPU(graphics processing unit)架構中使用 BVHs(Bounding Volume Hierarchies)及 Kd-trees 之間針對渲染時間和走訪特徵之間的效能比較。BVHs 的特徵是確保每個分割區塊都包含在一個葉節點中[EG07]。雖然發現走訪次數和交點測試為 BVHs 多於 Kd-trees，但光線追蹤的效能在 BVHs 結構使用在中小型景物時較 Kd-trees 快，對於在計算能力較低的行動裝置上進行渲染的過程非常重要。

接著討論空間資料及幾何圖形的相關議題。在本論文中使用到的淹水區間，其概念來自書籍[EK11]中提到單閉區間(Simply Connected Domain)的概念。在相關文獻[FS75]中，提出將凸多邊形取最小面積矩形(rectangle of minimum area)的方法，透過計算凸點之間的角度來找出最好的旋轉度數，來決定最小面積的矩形，其概念類似於[BKSS94]提到的 RMBR(rotated minimum bounding rectangle)，其包圍面積較 MBR 小。但由於 R-tree 無法存取歪斜的空間資料形式，因此旋轉後的矩形不適用。

空間連接(Spatial Join)是組合空間物件最重要的運算，該運算針對兩個空間物件的集合找出彼此之間具有某種特殊空間關係的配對。在相關論文[BKSS94]中，使用真實資料集並透過不同 bounding box 的取法，對二維的空間相交處理進行了詳細的研究。透過以下三步驟的查詢來增進空間連接的效能，第一步是利用空間存取方法 SAMs(Spatial Access Methods)來限制查詢範圍，找出候選的組合。第二步是利用幾何過濾器(Geometric Filter)來檢查是否符合空間連接的預期，分為完全符合(Hits)、非完全符合(False Hits)及剩餘可能符合的候選組合(Remaining Pairs of Candidate Possibly Fullfill)。第三步是針對空間物件的近似表示法來比較剩餘可能符合的候選組合，並輸出相對應的組合，而近似表示法包含最小包圍矩形(MBR)、最小包圍圓形(MBC)、旋轉的最小包圍矩形(RMBR)、最大封閉矩形(MER)、最大封閉圓形(MEC)、凸邊形(Convex Hull)及最小包圍多角形(MBm-c)。另外，論文[ZS98]也利用近似的表示法，透過過濾的方式找出需要做進一步檢查的組合來增進處理效能。該論文提出四色光柵法(Four-Colors Raster Signature, 4CRS)來表示區塊的型態，四種型態包含空(Empty)、弱相交(Weak)、強相交(Stong)及滿(Full)，交叉識別出的三項結果包含丟棄(Discard)、候選(Candidate)及保留(Accept)。四種顏色的型態依序定義，空代表與多邊形完全無相交的區塊，弱相交代表區塊與多邊形的相交程度小於等於 50%，強相交代表與多邊形的相交程度大於 50%，而滿即為被多邊形完全包含。四色光柵法與本論文 Quad 方法中的門檻值 MaxRatio 概念相似。

在資料庫的部分我們討論索引技術的相關議題。論文[AG84]使用 R-tree 來做Spatial join 查詢，而R-tree是一種階層式資料結構(Hierarchical Data Structure)，被用來儲存空間物件中的矩形，可以便於進行空間的重疊等查詢。至於論文[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。而分割方法是利用三種方法來找出最佳分割軸，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。

最後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），如 Dijkstra 演算法[EW59]是常被用來進行路徑規劃的演算法，該演算法保證找到最短路徑，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。[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。

1.4 論文架構

本論文其餘各章節的架構如下：第二章說明本論文的背景並給定相關的定義解釋及系統架構，藉以對本論文的研究有基礎的認識。第三章介紹基本之分割作法與改良之作法，並在說明演算法之後給定步驟範例。第四章介紹四分法分割，說明其性質、判斷方式及演算法，亦在說明演算法之後給定步驟範例。第五章介紹實驗資料的取得及使用方式，並說明門檻值的設定方式。最後透過路徑長度及效率比較的實驗，來比較所有方法的效率及路徑長度，並且透過不同資料集及參數設定來探討對路徑規劃結果的影響。最後，在第六章提出本論文的結論與未來方向。

第2章　背景說明與系統架構

我們在此章說明本論文欲解決的問題及其背景，並提出系統架構。

2.1　背景說明

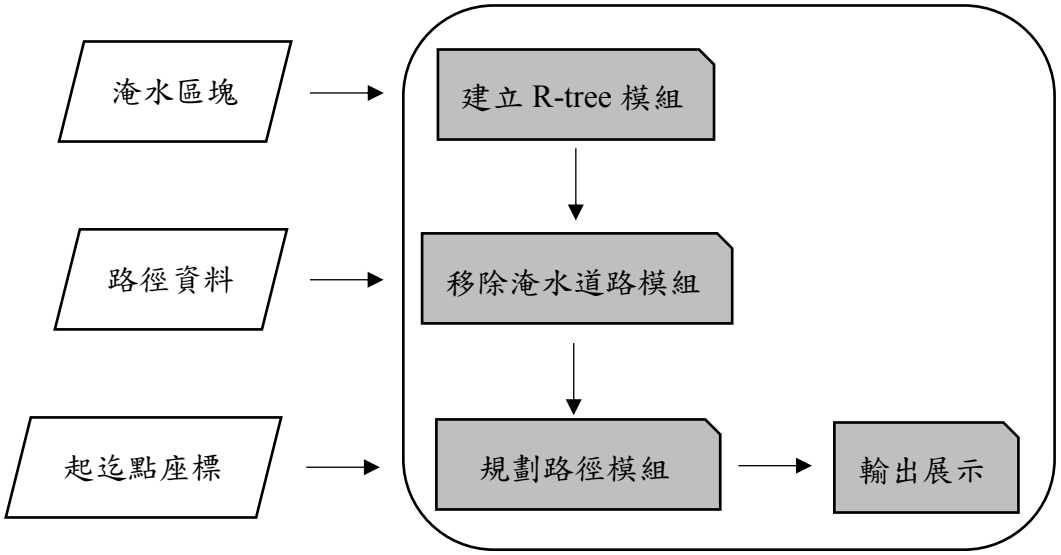


圖 2-1　[洪 16]系統架構圖

論文[洪 16]曾討論如何在淹水區域，「快速」規劃出一條迴避淹水區塊路徑的問題，其作法如上圖 2-1 所示，輸入為淹水區塊、路徑資料及起迄點座標。步驟為先以淹水區塊建立 R-tree 索引，接著以路徑資料走訪 R-tree，移除與淹水區塊有相交的道路，最後再透過起迄點來規劃最短路經，並輸出展示畫面。

我們進一步說明三個主要的模組。首先是建立 R-tree 模組，輸入的資料為矩形的淹水區塊，如下圖 2-2 中的 *F1* 到 *F5*。我們以 2 至 3 個淹水區塊形成 MBR，左邊三塊淹水區塊 *F1*、*F4*、*F5* 形成 *M1*，右邊兩塊 *F2*、*F3* 形成 *M2*，所建立的索引如圖 2-3 所示。接著，移除淹水道路模組，輸入的資料為道路資料，而輸出的資料是與淹水區塊沒有相交的道路。在此範例路徑資料中，*V1* 到 *V8* 代表各個路口點，起迄點分別為 *V1* 及 *V4*，原本從 *V1* 到 *V4* 的最短路徑是 *V1→V2→V5→V4*，但是路徑 *V2→V5* 與索引根節點中的 M1 相交，進而發現該路徑與淹水區塊 *F4* 相交，因此 *V2→V5* 這條路徑就會被移除，依此類推將淹水的道路移除，並將未淹水的道路輸出。最後，透過規劃路徑模組，使用 Dijkstra 演算法在所有未淹水道路中規劃出一條迴避淹水的最短路徑為 *V1→V2→V3→V4*，最後在 Google Map 上進行展示。

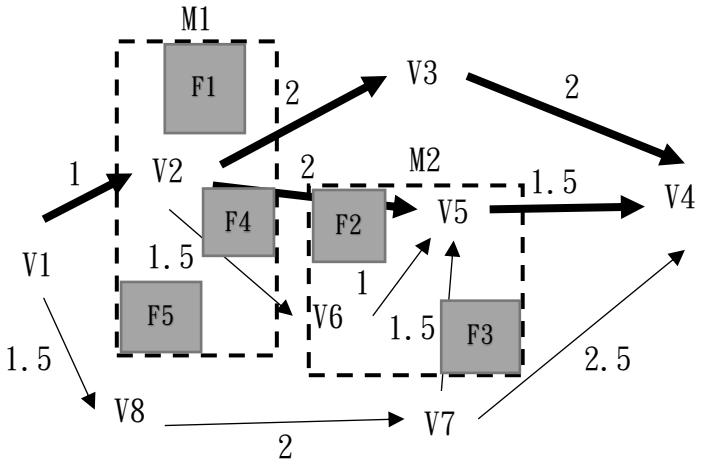


圖 2-2　淹水區塊與路網圖

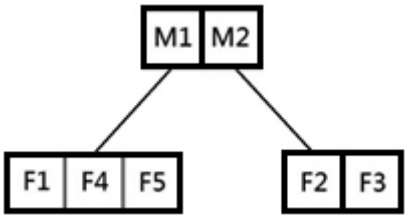


圖 2-3　R-tree 範例

2.2　基本定義

本論文希望分割任意形狀的淹水資料為一個個矩形，以配合圖 2-1 的系統架構圖，以下提出後續會使用的一些名詞定義：

[定義 2-1]　單閉區間(simply connected domain) [EK11][1]：當在定義域中隨意畫出封閉之曲線，並讓此封閉曲線逐漸收縮成一點，如果在收縮成一個點的過程當中，定義域裡面的每一個點都不會跑到收縮的曲線之外的話，則此定義域稱為單閉區間，如圖 2-4 之三個圖形所示。

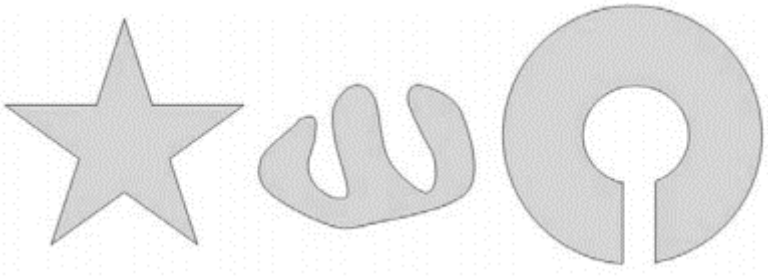


圖 2-4　單閉區間示意圖

[1] 此定義的原始英文敘述如下：A domain ***D*** is called simply connected if every closed curve in ***D*** can be continuously shrunk to any point in ***D*** without leaving ***D***.

[定義 2-2] 淹水區間(flooded spot)： 由原始淹水資料所構成的點序列，而且輸入的淹水資料所形成的圖型皆為單閉區間。序列中的點原始資料以經緯度表示，在論文中有時以序號及圖示實心圓●簡化之。如下圖 2-5 所示，是由點 1 至點 56 所構成的點序列，其中點 1 同時代表此點序列的起點和終點。

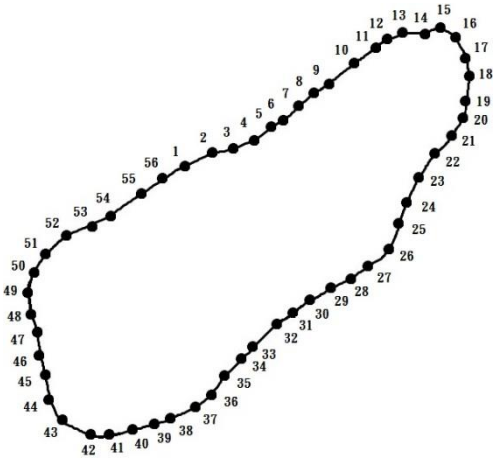


圖 2-5 點序列

[定義 2-3] MBR(minimum bounding rectangle)： 表示涵蓋物件(如一個淹水區間)的最小邊界矩形，如下圖 2-6 所示，其中Min_x為矩形範圍內最小 x 值，Min_y為最小 y 值，Max_x為最大 x 值，Max_y為最大 y 值。在後續的論文中會以 $(Min_x, Min_y, Max_x, Max_y)$表示一個特定的 MBR。

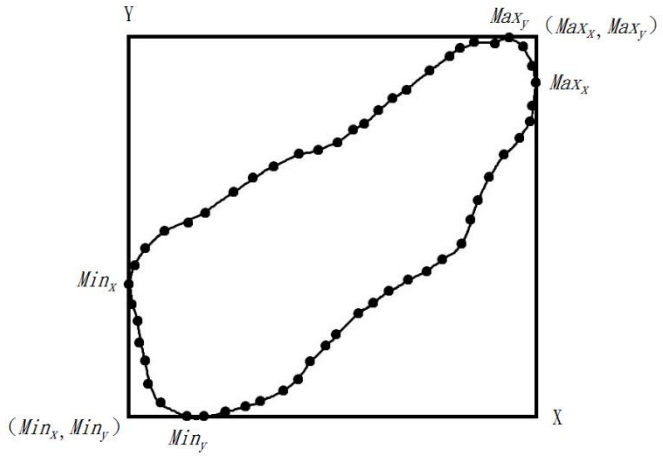


圖 2-6 淹水區間 MBR 範例

[定義 2-4] 分割區塊： 由淹水區間所分割出的矩形。

[定義 2-5] 淹水區塊： 與原始淹水區間有相交的分割區塊。

[定義 2-6] 點和點距離： 本論文的輸入資料點皆以經緯度表示，但為了方便起見，經緯度會分別以 X 軸和 Y 軸座標來標示，而兩點之間的距離皆為大圓距離（Great-circle distance）[GB97]。

2.3 系統架構

我們在此處介紹本論文的系統架構如下圖 2-7 所示。首先呼叫分割淹水區間模組，其輸入資料為淹水區間 ***S*** 的座標點序列，然後輸出淹水區塊，再接續使用如圖 2-1 的既有系統，包含建立 R-tree 模組、移除淹水道路模組以及規劃路徑模組。

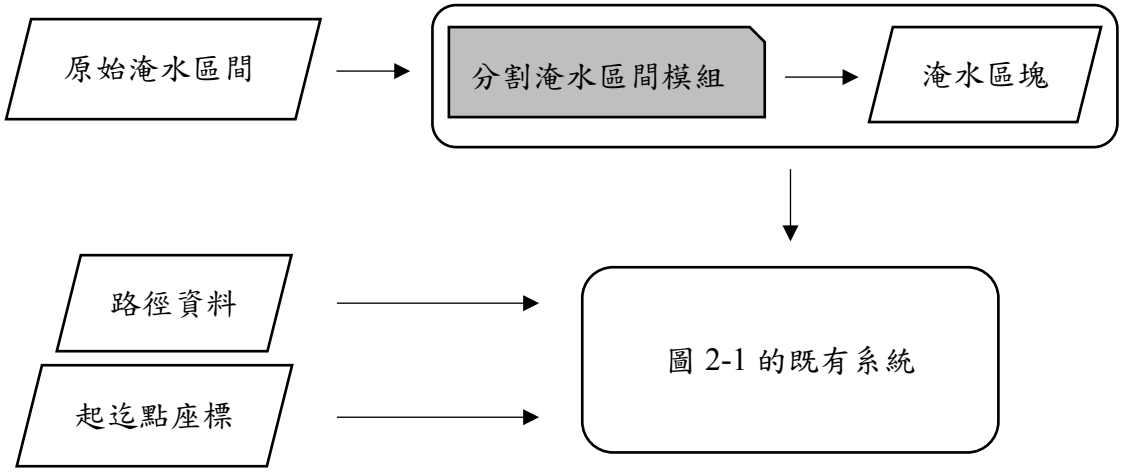


圖 2-7　系統架構圖

系統的主程式(Main)如下圖 2-8所示，首先L01從檔案中讀取所有原始淹水區間，L02從檔案中讀取路徑資料，L03-05會呼叫分割淹水區間模組(以Partition函數代表)，依序分割各個原始淹水區間***S***的點序列，將淹水區塊儲存於OutputList序列中。接著，L06利用分割出的淹水區塊和給定的索引order來建立R-tree，L07初始未淹水道路NoFloodRoad為空，L09-L10利用索引判斷一條道路是否與淹水區間相交，若不相交則將其加入到NoFloodRoad中。L13將起迄點及所有的未淹水道路NoFloodRoad交由Dijkstra做路徑規劃，規畫出最短路徑，最後將最短路徑輸出。

演算法名稱：**Main** 輸入：order, FloodFile, RoadFile, V_s, V_t //V_s和V_t表示起迄路口點 輸出：ShortPath //V_s到V_t的最短路徑
L01　FloodSet←ReadFlood(FloodFile) L02　RoadSet←ReadRoad(RoadFile) L03　**for** (each shape ***S*** of FloodSet) L04　　OutputList.add(**Partition**(*parameters…*))　//分割淹水區間模組 L05　**End for** L06　FloodIndex←BuildFloodIndex(OutputList, order)　//建立 R-tree 模組

```
L07   Initialize NoFloodRoad←NULL
L08   for (each Road r of RoadSet)
L09     If( not SearchRoadIndex(Root, r) )   //移除淹水道路模組
L10       NoFloodRoad.add(r)
L11     End If
L12   End for
L13   ShortPath←Dijkstra(V_s, V_t, NoFloodRoad)   //規劃路徑模組
```

圖 2-8　Main 演算法

第3章 基本作法與改良作法

分割淹水區間的方法，最直覺的做法就是將一個淹水區間依據給定的大小進行分割，而為了改善該方法中不必要的分割，我們進一步提出改良的作法。在本論文中，我們會比較不同作法所需之空間運算次數。空間運算分為三類，分別為空間相交判斷、點包含判斷及面積大小比較，以下詳細說明各方法與其所需之運算次數。

3.1 基本方法

此方法的精神是先取淹水區間的 MBR，透過端點 *P* 的移動將淹水區間分割成邊長為 CellEdge 的正方形分割區塊，如下圖 3-1。分割的順序為左下至右上，如圖 3-2，會依序分割編號為 1-2、3、4 的範圍。其中編號 1 代表 MBR 中 CellEdge 整數倍數的最大範圍，最後輸出與原始淹水區間有相交的分割區塊。

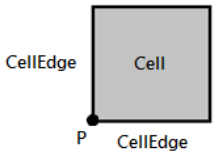


圖 3-1 分割區塊示意圖

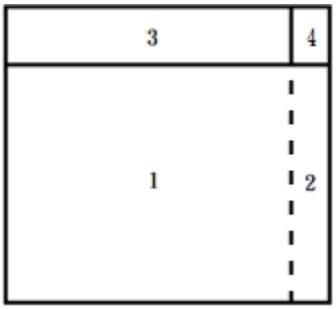


圖 3-2 分割區塊的順序編號

此 Baseline 方法，以下稱作 **BaseFS**，如下圖 3-3。輸入的淹水資料為一個原始淹水區間 ***S*** 的點序列以及欲分割的區塊邊長 CellEdge，其單位為公尺。變數的部分，端點 ***P*** 表示目前處理分割區塊的左下端點，Lng 和 Lat 表示每個分割區塊(Cell)的長與寬的距離(單位分別為經度和緯度)，remainX 及 remainY 分別表示剩餘的長與寬的距離(單位分別為經度和緯度)。和原始淹水區間有相交的分割區塊則記錄於 OutputList 中，做為輸出。

演算法首先於 L01 讀取淹水區間點序列，並將點序列設定 MBR。L02 及 L03 將 CellEdge 的公制距離(公尺) 透過函數 **m2Lng** 及 **m2Lat** 分別轉為經度度數 Lng 及緯度度數 Lat。計算的原理為利用赤道周長 40075.02 公里(相當於 40075020 公

尺)及子午線周長 20001.6 公里(相當於 20001600 公尺)，經度為東西經各 180°，經度共 360°；緯度為南北緯各 90°，緯度共 180°。以經度 **m2Lng** 函數來說明，透過赤道周長除以經度總度數，可以得知經線每 1°的距離為 111319.5 公尺，所以每 $\boldsymbol{m}$ 公尺對應的經度距離為 $\boldsymbol{m}$ 除以 111319.5。至於每 $\boldsymbol{m}$ 公尺緯度度數的計算方式也如同上所述。兩個轉換公式如下：

$$\text{每}\boldsymbol{m}\text{公尺的經度度數：}\mathbf{m2Lng}(\boldsymbol{m}) = \frac{\boldsymbol{m}}{\frac{40075020}{360}} = \frac{\boldsymbol{m}}{111319.5}$$

$$\text{每}\boldsymbol{m}\text{公尺的緯度度數：}\mathbf{m2Lat}(\boldsymbol{m}) = \frac{\boldsymbol{m}}{\frac{20001600}{180}} = \frac{\boldsymbol{m}}{111120}$$

接下來，L04-L07 判斷 MBR 的面積是否小於分割區塊(Cell)的面積，若是則將其直接輸出，反之，由左下往右上方分割。L08 初始化輸出序列、暫存序列、端點 $\boldsymbol{P}$ 座標，並給定剩餘距離為 0。

L09-L22 進行如圖 3-2 編號 1-2 的區塊分割，若端點 $\boldsymbol{P}$ 的 Y 座標加上緯度距離 Lat 沒有超過 MBR 的 Y 軸範圍，就將端點 $\boldsymbol{P}$ 的 X 座標設為 MBR 的最小 X 值，若同時端點 $\boldsymbol{P}$ 的 X 座標加上 Lng 沒有超過 MBR 的 X 軸範圍，L11 就以端點 $\boldsymbol{P}$ 為左下頂點且長寬分別以 Lng 及 Lat 的矩形設定該分割區塊，並加入到暫存序列中，並將端點 $\boldsymbol{P}$ 的 X 座標累加 Lng。若端點 $\boldsymbol{P}$ 的 X 座標加上 Lng 會超出 MBR 的範圍，就計算剩餘距離 remainX，L17-L20 若 remainX 不為 0，就依照 remainX 和給定的 Lat 設定分割區塊並加入到暫存序列中。完成分割一列後，L21 將端點 $\boldsymbol{P}$ 的 Y 座標累加 Lat，繼續往上進行分割。

接著進行編號 3-4 的區塊分割，當端點 $\boldsymbol{P}$ 的 Y 座標加上 Lat 超出 MBR 的範圍，L23 就計算剩餘的 Y 軸距離 remainY，L24 並將端點 $\boldsymbol{P}$ 的 X 座標設為 MBR 的最小 X 值。L25-L29 當端點 $\boldsymbol{P}$ 的 X 座標加上 Lng 沒有超出 MBR 的 X 軸範圍，就設定分割區塊並加入到暫存序列中，並將端點 $\boldsymbol{P}$ 的 X 座標累加 Lng。若剩餘的邊長皆不為 0，就以 remainX 及 remainY 設定分割區塊，並加入到暫存序列中，完成分割。最後 L34-L38 將跟原始淹水區間有相交的分割區塊加入輸出序列 OutputList 中並回傳。

BaseFS 方法會修正圖 2-8 Main 演算法如下：

L04 OutputList.add(**BaseFS**($\boldsymbol{S}$, CellEdge))

演算法名稱：**BaseFS**
輸入：$\boldsymbol{S}$, CellEdge
輸出：OutputList //儲存與原始淹水區間有相交的 Cell

```
L01  MBR(Min_x, Min_y, Max_x, Max_y)←SetMBR(S)
L02  Lng←m2Lng(CellEdge)  //CellEdge 的經度度數
L03  Lat←m2Lat(CellEdge)  //CellEdge 的緯度度數
L04  If(MBR.area < pow(CellEdge, 2) )
L05     OutputList←MBR
L06     return OutputList
L07  End If
L08  Initialize OutputList←null; TempList←null; P.x←Min_x; P.y←Min_y;
     remainX←0; remainY←0
L09  while(P.y + Lat ≦ Max_y)  //分割等分淹水區塊
L10     P.x←Min_x
L11     while(P.x + Lng ≦ Max_x)
L12        Cell.Set(P, Lng, Lat)
L13        TempList.add(Cell)
L14        P.x += Lng
L15     End while
L16     remainX←Max_x-P.x
L17     If(remainX)  //記錄右方分剩餘範圍
L18        Cell.Set(P, remainX, Lat)
L19        TempList.add(Cell)
L20     End If
L21     P.y += Lat
L22  End while
L23  remainY←Max_y- P.y
L24  P.x←Min_x
L25  while(P.x + Lng ≦ Max_x)  //處理上方剩餘範圍
L26     Cell.Set(P, Lng, remainY)
L27     TempList.add(Cell)
L28     P.x += Lng
L29  End while
L30  If(remainX != 0 && remainY != 0)  //處理右上方最後一塊剩餘區塊
```

```
L31      Cell.Set(P, remainX, remainY)
L32      TempList.add(Cell)
L33   End If
L34   For each Cell in TempList
L35      If(intersects(S, Cell) )
L36         OutputList.add(Cell)
L37      End If
L38   End For
L39   return OutputList
```

圖 3-3　**BaseFS** 演算法

[範例 3-1]　以下說明本演算法的分割結果如下圖 3-4 所示，假設參數 CellEdge 為一長度單位。L09-L22 進行編號 1 和 2 範圍之分割，所得之分割區塊為(0, 0, 1, 1)到(10, 8, 10.2, 9)的 99 個區塊。L25-L29 進行編號 3 範圍之分割，所得之分割區塊為(0, 9, 1, 9.5)到(9, 9, 10, 9.5)的 10 個區塊。L30-L33 分割出右上最後一塊剩餘的實線矩形區塊，座標為(10, 9, 10.2, 9.5)，L34-L38 將所得之 110 個分割區塊分別和原始淹水區間做空間相交判斷，最後得到以實線標示之淹水區塊共 63 個。最後，針對空間運算的次數，**BaseFS** 方法在 L04 進行面積大小比較次數 1 次，而每一個分割區塊都於 L35 和原始淹水區間進行空間相交判斷，次數為 110 次，因此空間計算總共為 111 次。

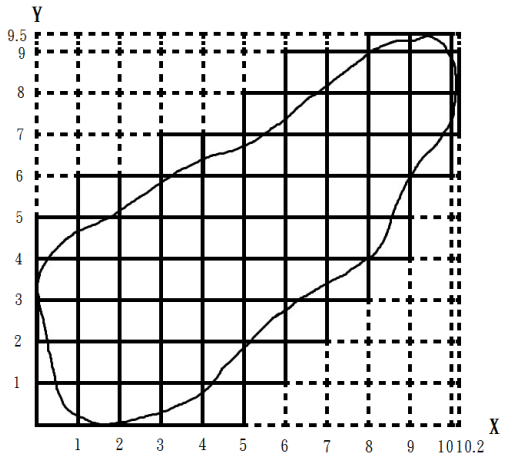


圖 3-4　**BaseFS** 方法之分割範例

3.2　降低相交運算的改進方法

3.1 節所描述的 **BaseFS** 方法適用於任意形狀的淹水區間，但是從圖 3-4 可以發現，(1, 1, 4, 4)對應到一個連續淹水的大範圍，但內部被分割成很多小塊，

而每塊都要做一次空間相交的運算，所以本節討論改進的方法。

在文獻[TB94]中可看到 MER(maximum enclosed rectangle)定義為一個在多邊型內的最大封閉的矩形，參閱相關的論文發現該方法需要很複雜的計算，而且求出來的矩形其四邊不一定平行 X 軸或 Y 軸，不符合建立 R-tree 的要求，所以我們提出 MIR(minimum interPoints rectangle)的概念。也就是基於 **BaseFS** 的分割方法，但是透過先求取 MIR 來減少不必要的分割及空間相交判斷，以下我們稱作 **BaseMIR** 方法。

首先說明本方法會使用到的基本定義：

[定義 3-1] 中心點(Center)： 淹水區間 MBR 或分割區塊的中心座標，後續以圖示實心菱形◆表示。

[定義 3-2] 方位： 依淹水區間 MBR 或分割區塊的中心點分為 8 個方位 NW、NE、SE、SW、W、E、N、S。方位 NW、NE、SE、SW 有時會分別以左上、右上、右下、左下象限稱呼之。而方位 W 或 E 又稱作 X 軸，方位 N 或 S 又稱作 Y 軸。計算方式是以中心點座標及查詢座標點的相對位置來決定方位。

[定義 3-3] 交點(InterPoint)、軸點: 以下將交點稱做為$\boldsymbol{P}_{Inpt}$，若點序列的連續兩點位於不同方位，且沒有任一點落在 X 軸或 Y 軸上，則利用內插法求取位於 X 軸或 Y 軸之點，如下圖 3-5 所圈選的座標點所示，後續會以圖示空心圓○表示。至於淹水點序列 $\boldsymbol{S}$ 中落在軸上的座標點，稱做軸點。

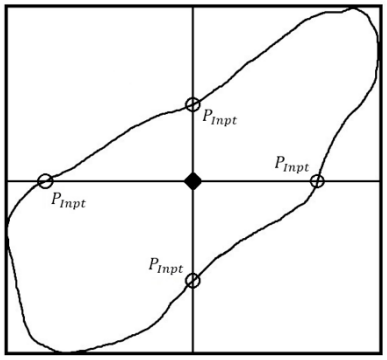
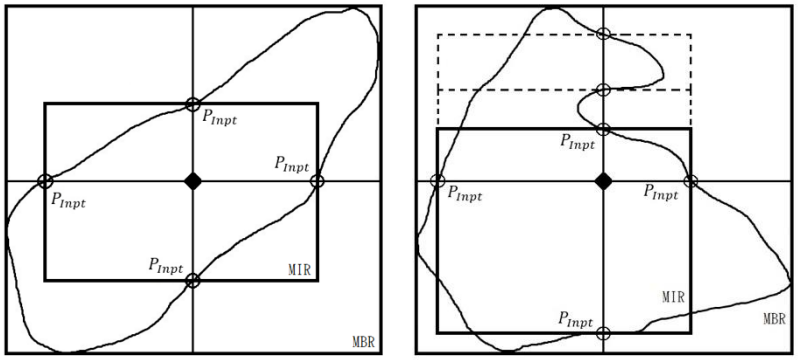


圖 3-5 交點示意圖

[定義 3-4] MIR(minimum interpoints rectangle)： 若淹水區間 $\boldsymbol{S}$ 的 MBR 的中心點位於 $\boldsymbol{S}$ 內部，則由 $\boldsymbol{S}$ 分別自東西南北方位求取最接近中心點的 4 個交點或軸點，則此 4 點所設定的矩形稱做 MIR。如下圖 3-6 (a)內框所示，而外框代表的是該淹水區間的 MBR。



(a) (b)

圖 3-6 淹水區間的 MIR 範例

另外注意到，**[定義 3-4]**中會求取東西南北四個方位最“接近”中心點的交點或軸點，所以若一個方位有兩個交點或軸點，如上圖 3-6 (b)的北方，則 MIR 會如實線框所示，而非虛線框所示。如此定義是為了避免 MIR 中包含太多沒有淹水的區域。

接著介紹 **BaseMIR** 演算法中所使用的資料結構 Coordinate、Bounding box 及 Direction。其中 Coordinate 包含經度 x 及緯度 y 用以宣告變數 Inpt 及點序列 ***S***，Bounding box 包含面積大小 area 及中心點 center 用以宣告變數 MBR 及 MIR，Direction 用以宣告變數 ***D***，表示座標點的方位，如**[定義 3-2]**所列之 8 個方位。

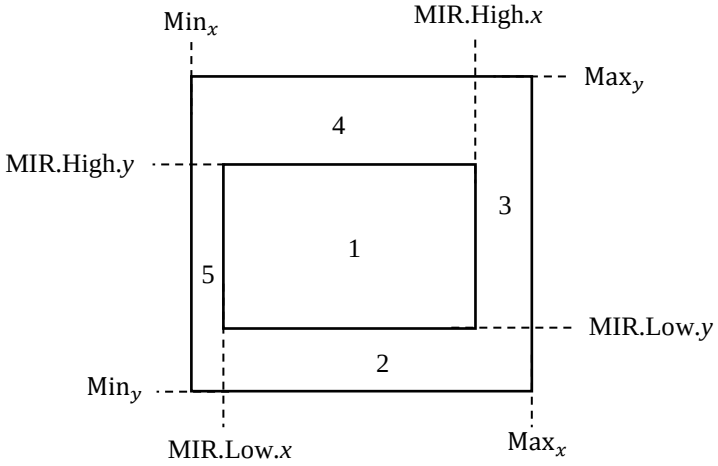


圖 3-7 分割順序的區塊編號

此作法的精神是先求取淹水區間的最小交點矩形如上圖 3-7 編號 1，之後給定編號 2-5 的 4 個區塊，依序呼叫 **BaseFS** 方法進行分割，最後輸出與原始淹水區間相交的淹水區塊，以下說明 **BaseMIR** 演算法如圖 3-8。

首先，以淹水區間 ***S*** 的點序列設定其 MBR，若 MBR 面積小於分割區塊面積，就直接整塊回傳不處理，否則就判斷 MBR 的中心點是否在淹水區間外，若中心點在淹水區間外，則直接以 **BaseFS** 的方式分割，回傳的淹水區塊即為與原

始淹水區間有相交的區塊。

以下說明本方法與前一方法不同之處。首先初始化座標點註標 i，交點序列、暫存序列、輸出序列、圖 3-7 編號 2-5 的四個區塊點序列(M2-M5)為空，並將東南西北 4 個方位上最接近中心點的座標點初始為 MBR 的極值，也就是離中心點最遠的點。L10-L38 逐點處理點序列 ***S***，以取出其 MIR。L11-23 首先處理軸點，處理方式為逐點呼叫 **IFOnAxis 函數**判斷點 i 是否為軸點，若是則呼叫 **SetAxisPoint** 函數更新該方位最接近中心點的點。此處使用 while 迴圈是為了處理如上圖 3-6(b)有連續軸點的狀況。

反之，點 i 不在軸上，則 L24 呼叫 **IFCross** 函數判斷下一座標點是否和點 i 不在同一個方位，當下一點跨越方位時，L25-35 呼叫 **GetInterPoint** 函數計算兩點之間和 X 軸或 Y 軸的交點，並呼叫 **FindDirection** 函數計算交點是落在軸上的哪個方位，然後更新離中心點最接近的點。L27-L35 和上一段落中 L13-L21 相似，不同之處在於，L13-L21 是處理軸點，傳給 **SetAxisPoint** 函數的值是點 i，L27-L35 是處理交點，傳給 **SetAxisPoint** 函數的值是 L25 計算出的交點 Inpt。在 L38while 迴圈結束時，已計算出 4 個最接近中心點的座標 InptN、InptE、InptS 及 InptW，依序加入到交點序列中，並透過交點序列來設定最小交點矩形 MIR。

L42-L49 以 MIR 及 MBR 的邊界分別設定其餘 4 個區塊點序列 M2 至 M5 如上圖 3-7，並呼叫 **BaseFS** 方法傳入區塊點序列及 CellEdge 來分割剩餘區塊，再將分割出的區塊存入暫存序列中，最後 L50-L54 呼叫 **intersects** 函數將與原始淹水區間有相交的區塊加入到輸出序列並輸出。

BaseMIR 方法會修正圖 2-8　Main 演算法如下：

L04　　OutputList.add(**BaseMIR**(***S***, CellEdge))

演算法名稱：**BaseMIR** 輸入：***S***, CellEdge 輸出：OutputList
L01　MBR(Min_x, Min_y, Max_x, Max_y)←SetMBR(***S***)
L02　**If**(not **Within**(MBR.center, ***S***))　//判斷中心點在淹水區間內或外
L03　　OutputList←**BaseFS**(***S***, CellEdge)　//直接以 **BaseFS** 方法分割
L04　　**return** OutputList
L05　**Else If**(MBR.area < pow(CellEdge, 2))
L06　　OutputList←MBR
L07　　**return** OutputList
L08　**End If**
L09　**Initialize** i←0; (InptList, TempList, OutputList, M2, M3, M4, M5)←null;

```
                InptN←Max_y; InptE←Max_x; InptS←Min_y; InptW←Min_x;
L10   while( i < S.size() )
L11     while( IFOnAxis(S[i], MBR.center) )
L12       D←FindDirection(S[i], MBR.center)
L13       If(D = "N")
L14         InptN←SetAxisPoint(S[i], InptN, D)
L15       Else If(D = "S")
L16         InptS←SetAxisPoint(S[i], InptS, D)
L17       Else If(D = "E")
L18         InptE←SetAxisPoint(S[i], InptE, D)
L19       Else If(D = "W")
L20         InptW←SetAxisPoint(S[i], InptW, D)
L21       End If
L22       i++
L23     End while   //End while (IFOnAxis())
L24     If(IFCross(S[i], S[i+1], MBR.center))   //下一點跨越方位
L25       Inpt←GetInterPoint(S[i], S[i+1], MBR.center)   //求交點
L26       D←FindDirection(Inpt, MBR.center)
L27       If(D = "N")
L28         InptN←SetAxisPoint(Inpt, InptN, D)
L29       Else If(D = "S")
L30         InptS←SetAxisPoint(Inpt, InptS, D)
L31       Else If(D = "E")
L32         InptE←SetAxisPoint(Inpt, InptE, D)
L33       Else If(D = "W")
L34         InptW←SetAxisPoint(Inpt, InptW, D)
L35       End If
L36     End If   //End if (IFCross)
L37     i++
L38   End While
L39   InptList.add(InptN); InptList.add(InptE);
```

```
        InptList.add(InptS); InptList.add(InptW)
L40   MIR.Set(InptList)   //設定 MIR
L41   OutputList.add(MIR)
L42   M2.add( (MIR.Low.x , Min_y), (MIR.Low.x , MIR.Low.y) ,
        (Max_x , MIR.Low.y), (Max_x , Min_y) )   //設定 M2
L43   TempList.add( BaseFS(M2, CellEdge) )   //暫存分割出的區塊
L44   M3.add( (MIR.High.x , MIR. Low. y), (MIR.High.x , Max_y) ,
        (Max_x , Max_y), (Max_x , MIR. Low. y) )   //設定 M3
L45   TempList.add( BaseFS(M3, CellEdge) )
L46   M4.add( (Min_x , MIR. High. y), (Min_x , Max_y) ,
        (MIR. High. x , Max_y), (MIR. High. x , MIR. High. y) )   //設定 M4
L47   TempList.add( BaseFS(M4, CellEdge) )
L48   M5.add( (Min_x , Min_y), (Min_x , MIR. High. y) ,
        (MIR. Low. x , MIR. High. y), (MIR. Low. x , Min_y) )   //設定 M5
L49   TempList.add( BaseFS(M5, CellEdge) )
L50   For each MBR in TempList
L51     If(intersects(S, MBR) )
L52       OutputList.add(MBR)
L53     End If
L54   End For
L55   return OutputList
```

圖 3-8　BaswMIR 演算法

[範例 3-2]　以下使用圖 3-4 的淹水區間為範例說明本演算法的分割結果如下圖 3-9 所示，同樣假設 CellEdge 為一長度單位。L39-L40 設定 MIR 並輸出，所得之分割區塊為(1, 1.9, 8.5, 6.8)。L42-L43 進行編號 2 範圍(1, 0, 10.2, 1.9)之分割，所得之分割區塊為(1, 0, 2, 1)到(10, 1, 10.2, 1.9)的 20 個區塊。L44-L45 進行編號 3 範圍(8.5, 1.9, 10.2, 9.5)之分割，所得之分割區塊為(8.5, 1.9, 9.5, 2.9)到(9.5, 8.9, 10.2, 9.5)的 16 個區塊。L46-L47 進行編號 4 範圍(0, 6.8, 8.5, 9.5)之分割，所得之分割區塊為(0, 6.8, 1, 7.8)到(8, 8.8, 8.5, 9.5)的 27 個區塊。L48-L49 進行編號 5 範圍(0, 0, 1, 6.8)之分割，所得之分割區塊為(0, 0, 1, 1)到(0, 6, 1, 6.8)的 7 個區塊。L50-L54 將 MIR 之外所得之 70 個分割區塊分別和原始淹水區間進行空間相交判斷，最後得到以實線標示之淹水區塊共 34 個。

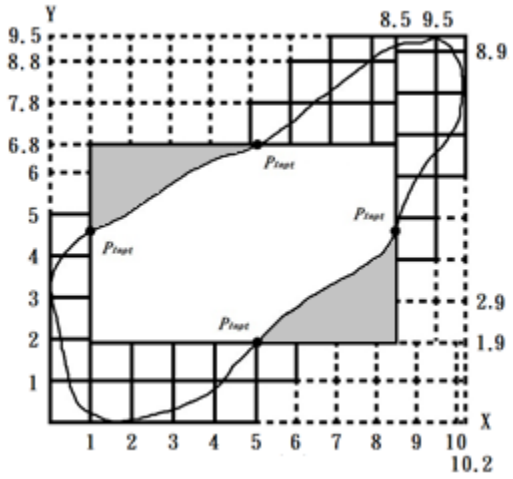


圖 3-9　BaseMIR 分割範例

與**[範例 3-1]**的 **BaseFS** 方法相比，**BaseFS** 方法空間計算的次數為 111 次，而 **BaseMIR** 方法點包含判斷的次數為 1 次，面積大小比較的次數為 5 次[2]，空間相交判斷的次數為 70 次，因此空間計算的次數總共為 76 次，較 **BaseFS** 方法減少 35 次。雖然透過 MIR 可以大幅降低空間計算次數，但 MIR 可能分割出的淹水區塊過大，而包含大量未與原始淹水區間相交之面積，如圖 3-9 灰色的淹水區間所示，因此在第四章我們提出門檻值方法來改善此問題。

最後說明本方法呼叫的幾個輔助函數，查詢點方位的演算法 **FindDirection** 如下圖 3-10，L01-04 判斷座標 ***P*** 的 *Y* 值大於中心點 *Y* 值，若座標 ***P*** 的 *X* 值小於中心點 *X* 值，則方位為西北，反之為東北。L05-08 判斷座標 ***P*** 的 *Y* 值小於中心點 *Y* 值，若座標 ***P*** 的 *X* 值大於中心點 *X* 值，則方位為東南，反之為西南。L09-L16，判斷座標 ***P*** 的 *X* 值等於中心點的 *X* 值，若座標 ***P*** 的 *Y* 值大於中心點 *X* 值，則方位為北，反之為南。判斷座標 ***P*** 的 *Y* 值等於中心點的 *Y* 值，若座標 ***P*** 的 *X* 值大於中心點 *X* 值，則方位為東，反之為西，L17 回傳該點的方位 ***D***。

演算法名稱：**FindDirection** 輸入：***P***, Center 輸出：***D*** //表示該點的方位
L01　**If**(***P***.*x* < Center.*x* & ***P***.*y* > Center.*y*) L02　　***D***←"NW" L03　**Else If**(***P***.*x* > Center.*x* & ***P***.*y* > Center.*y*) L04　　***D***←"NE" L05　**Else If**(***P***.*x* > Center.*x* & ***P***.*y* < Center.*y*) L06　　***D***←"SE"

[2] 原始 MBR 於 L5 比較一次，M2-M5 於呼叫 **BaseFS** 方法分割時各一次。

```
L07   Else If(P.x < Center.x & P.y < Center.y)
L08      D←"SW"
L09   Else If(P.x = Center.x & P.y > Center.y)
L10      D←"N"
L11   Else If(P.x = Center.x & P.y < Center.y)
L12      D←"S"
L13   Else If(P.x > Center.x & P.y = Center.y)
L14      D←"E"
L15   Else If(P.x < Center.x & P.y = Center.y)
L16      D←"W"
L17   return D
```

圖 3-10　FindDirection 演算法

判斷座標點是否在軸上的演算法 **IFOnAxis** 如圖 3-11，L01 透過演算法 **FindDirection** 來找出座標點***P***的方位***D***，L02-L03 當方位***D***在軸上則代表該點為軸點，即回傳 TRUE，反之 L04-L06 則回傳 FALSE。

```
演算法名稱：IFOnAxis
輸入：P, MBR.center
變數：D  //表示座標點P的方位
輸出：bool
```

```
L01   D←FindDirection(P, Cell.center)
L02   If(D = "N" || D = "S" || D = "E" || D = "W")
L03      return TRUE
L04   Else
L05      return FALSE
L06   End If
```

圖 3-11　IFOnAxis 演算法

設定最接近中心點的 4 個軸上的交點或軸點是由演算法 **SetAxisPoint** 來處理如下圖 3-12。L01-L04 處理當點 ***P*** 落在 *Y* 軸上的方位 *N*，當點 ***P*** 的 *y* 座標較交點值 Inpt 的 *y* 座標小，表示點更接近中心點，就把交點值 Inpt 取代為點***P***的值，L05-L17 依此類推，最後將交點值 Inpt 回傳。

演算法名稱：**SetAxisPoint**

輸入：***P***, Inpt, ***D***

```
L01   If(D = "N")
L02      If(P.y < Inpt.y)
L03         Inpt←P
L04      End If
L05   Else If(D = "S")
L06      If(P.y > Inpt.y)
L07         Inpt←P
L08      End If
L09   Else If(D = "E")
L10      If(P.x < Inpt.x)
L11         Inpt←P
L12      End If
L13   Else If(D = "W")
L14      If(P.x > Inpt.x)
L15         Inpt←P
L16      End If
L17   End If   // End If (D)
L18   return Inpt
```

圖 3-12　SetAxisPoint 演算法

判斷下一點是否跨越方位的演算法 **IFCross** 如下圖 3-13。條件是$\boldsymbol{D}_1$及$\boldsymbol{D}_2$皆不在軸上才算是跨越方位。L01-L02 找出$\boldsymbol{P}_1$及$\boldsymbol{P}_2$座標對應的方位$\boldsymbol{D}_1$及$\boldsymbol{D}_2$，L03-L09 判斷方位是否相異，若是兩方位屬於 NW、NE、SE 或 SW 且相異，則表示跨越方位即回傳 TRUE，其餘狀況則為相同方位或其中一個方位在軸上，則回傳 FALSE。

演算法名稱：**IFCross**

輸入：$\boldsymbol{P}_1$, $\boldsymbol{P}_2$, Center

變數：$\boldsymbol{D}_1$, $\boldsymbol{D}_2$ //表示$\boldsymbol{P}_1$, $\boldsymbol{P}_2$的方位

輸出：bool

```
L01   D1←FindDirection(P1, Center)
L02   D2←FindDirection(P2, Center)
L03   If(D1 = "NW" || D1 = "NE" || D1 = "SE" || D1 = "SW")
L04     If(D2 = "NW" || D2 = "NE" || D2 = "SE" || D2 = "SW")
L05       If(D1 != D2)
L06         return TRUE   //跨越方位
L07       End If
L08     End If
L09   End If
L10   return FALSE
```

圖 3-13　IFCross 演算法

計算交點的演算法 **GetInterPoint** 如下圖 3-14。此演算法是基於數學中的座標系公式：以中心點座標聯立求解P_1P_2直線方程式上的座標P_{Inpt}，如下列公式所示：

$$\begin{cases} y_1 = a * x_1 + b \\ y_2 = a * x_2 + b \end{cases} \quad ， \text{求解 } a \text{ 及 } b$$

L01 先聯立解出 a，再利用其中一組(x, y)座標解出 b。L02 找出$\boldsymbol{P}_1$及$\boldsymbol{P}_2$座標的方位$\boldsymbol{D}_1$及$\boldsymbol{D}_2$。L03-L04 判斷若$\boldsymbol{P}_1$及$\boldsymbol{P}_2$兩座標是在左右兩側如下圖 3-15 所示，$\boldsymbol{P}_{Inpt}$的 X 座標即為中心點的 X 座標，需求解$\boldsymbol{P}_{Inpt}$的 Y 座標。L05-L06 判斷若$\boldsymbol{P}_1$及$\boldsymbol{P}_2$兩座標是在上下兩側，需求解$\boldsymbol{P}_{Inpt}$的 X 座標，$\boldsymbol{P}_{Inpt}$的 Y 座標即為中心點的 Y 座標。L07-L08 判斷若是以對角線方式跨越方位，則交點$\boldsymbol{P}_{Inpt}$的座標就設為 Center 座標。L09 將計算出的交點座標回傳。

```
演算法名稱：GetInterPoint
輸入：P1, P2, Center
變數：D1, D2
輸出：P_Inpt  //表示計算出的交點
```

```
L01   a←(P1.y−P2.y) / (P1.x−P2.x); b←P1.y - (a*P1.x)
L02   D1←FindDirection(P1, Center), D2←FindDirection(P2, Center)
L03   If( (D1= NW && D2= NE) || (D1= NE && D2= NW) ||
        (D1= SW && D2= SE) || (D1= SE && D2= SW) )
L04     P_Inpt.x←Center.x; P_Inpt.y←(a * Center.x) + b
```

L05	**Else If**(($\boldsymbol{D}_1$= NW && D_2= SW) || ($\boldsymbol{D}_1$= SW && $\boldsymbol{D}_2$= NW) || ($\boldsymbol{D}_1$= NE && $\boldsymbol{D}_2$= SE) || (D_1= SE && D_2= NE))
L06	$\boldsymbol{P}_{Inpt}.x \leftarrow$*(Center.y - b) / a;* $\boldsymbol{P}_{Inpt}.y \leftarrow$*Center.y*
L07	**Else If**(($\boldsymbol{D}_1$= NW && $\boldsymbol{D}_2$= SE) || ($\boldsymbol{D}_1$= SE && $\boldsymbol{D}_2$= NW) || ($\boldsymbol{D}_1$= NE && $\boldsymbol{D}_2$= SW) || ($\boldsymbol{D}_1$= SW && $\boldsymbol{D}_2$= NE))
L08	$\boldsymbol{P}_{Inpt}.x \leftarrow$*Center.x;* $\boldsymbol{P}_{Inpt}.y \leftarrow$*Center.y*
L09	**return** $\boldsymbol{P}_{Inpt}$

圖 3-14　GetInterPoint 演算法

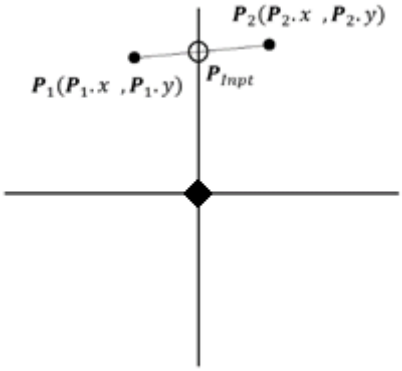


圖 3-15　座標系聯立求解交點示意圖

第4章 四分法分割的作法

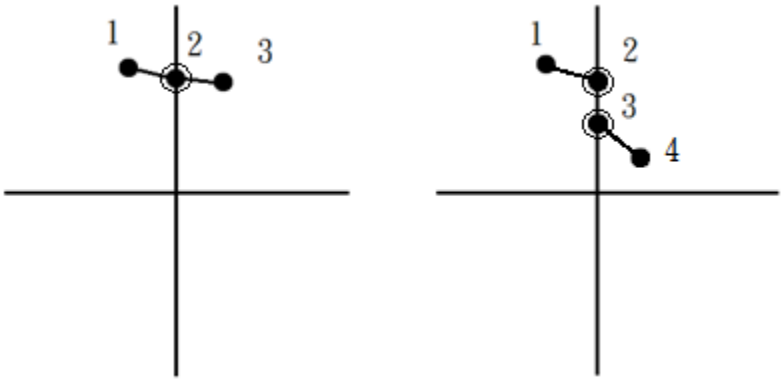
第三章提出兩種方法，首先是 **BaseFS** 方法，雖然較直覺簡單，但是在空間相交的判斷次數相當多。因此我們提出第二種改良的方法，稱作 **BaseMIR**，是透過取出最小交點矩形來降低空間相交的判斷次數，從[範例 3-2]可以看出，判斷空間相交的次數減少將近 1/3。但是若是處理細長的淹水空間，MIR 可能會過大而造成包含許多未淹水的面積，或是 MIR 過小會達不到減少連續判斷空間相交的效果。因此本章提出第三種分割的作法叫作 **QuadThreshold**，以下簡稱為 **Quad** 作法。

Quad 作法是透過調整門檻值，將淹水區間分割為大小不等的區塊，使輸出的淹水區塊總面積趨近原始淹水區間面積。輸入的淹水資料是一個淹水區間點序列，計算淹水區間面積及分割區塊面積後，將未滿足處理門檻值(面積比率或面積大小)的淹水區間點序列以中心點再分為四個方位，各別遞迴繼續分割，直到所得之分割區塊符合門檻值才停止。

4.1 補齊點序列的作法

在 **Quad** 作法中，我們會沿用[定義 3-1]至[定義 3-3]的中心點、方位、象限、交點及軸點的定義，另外會使用到的兩個定義如下 ：

[定義 4-1] 軸交點: 若一個軸點之前後點跨越象限，則稱做軸交點，後續以圖示空心圓包圍實心圓⊙表示。如下圖 4-1 中的圖(a)點 2 為軸交點，圖(b)之點 2、點 3 為連續兩個軸交點。判斷的方式以圖(a)來說明，是透過前一個不在軸上的點 1 的象限一個不在軸上的點 3 的象限比較，若兩點的象限不同，則代表點 2 為軸交點。

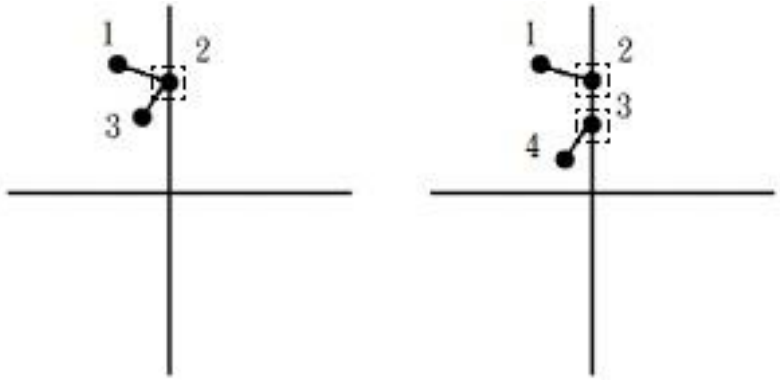


(a) (b)

圖 4-1 軸交點示意圖

[定義 4-2] 軸切點： 若軸點的前後點位於相同象限，則稱做軸切點，後續以圖示虛線方框包圍實心圓表示，其判斷以下圖 4-2 (a)來說明。前一個不在軸上的點 1 的象限與下一個不在軸上的點 3 的象限相同，

代表沒有跨越象限，所以點 2 為軸切點。至於圖 4-2 (b)的點 2 和點 3 皆為軸切點。



(a) (b)

圖 4-2 軸切點示意圖

由於 **Quad** 作法會把淹水區間的點序列分割成數段，所以我們提出利用中心點和交點組合成合理的點序列，以便讓面積的計算更準確，避免面積計算上的誤差。在此處我們針對封閉不中空的淹水區間，分別對中心點落在淹水區間內或外來做圖解說明。此處理模式適用於各象限及階層遞迴，我們以左上象限來說明，而灰色區域表示該段點序列所涵蓋的面積。

Quad 作法是將一個淹水區間分割為 4 個象限，當中心點落在淹水區間中，則四個象限的點序列都必須加入中心點才能計算正確的面積如圖 4-3 所示。因為若是沒有將中心點加入點序列中，分割出的點序列面積會如圖 4-4 所示，缺少一塊直角三角形的面積，而造成很大的誤差。

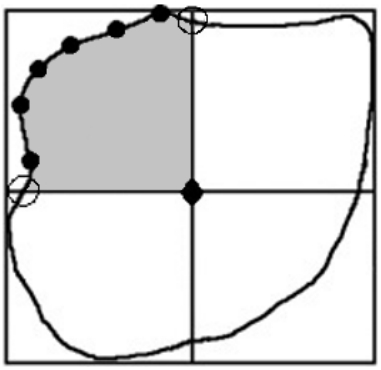


圖 4-3 中心點在淹水區間內且加入中心點的面積

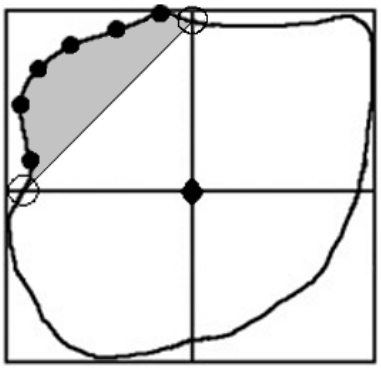


圖 4-4　中心點在淹水區間內未加入中心點的面積

當中心點落在淹水區間外，不需要加入中心點到點序列中，所形成的淹水區間面積如下圖 4-5 所示，呈現完整的點序列面積。

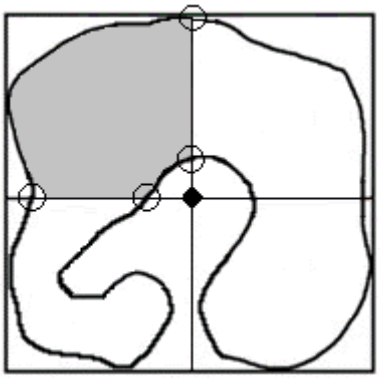


圖 4-5　中心點在淹水區間外的面積

至於交點的部分，觀察圖 4-3，若是只加入中心點而沒有將計算出的交點加入點序列中，如下圖 4-6 所示，以 *A* 點代表左上象限點序列的最後一點、*B* 點代表交點及 *C* 點代表中心點，則臨近 *Y* 軸的區域會缺少 *A*、*B* 及 *C* 點所形成的扇形面積，*X* 軸亦同，所以必須將交點 *B* 補入點序列中。

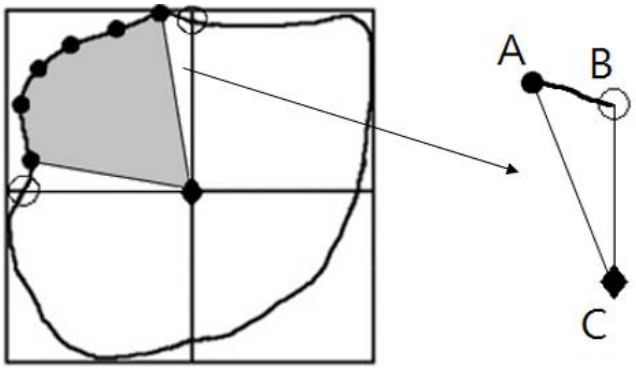


圖 4-6　未補入交點之面積示意圖

4.2 單閉區間的判斷

因為原始輸入的淹水區間為單閉區間，所以對於被四等分切割後所形成的 4 個子區間，我們希望也是單閉區間，以便遞迴處理，若已非單閉區間者會直接回傳其 MBR。由於後續的討論會根據子區間內象限轉換的情況，所以首先說明**[定義 3-3]**交點、**[定義 4-1]**軸交點與**[定義 4-2]**軸切點的相異之處。交點為象限轉換時所計算出的軸上點，而軸交點為原始圖形中象限轉換前，落在軸上的(連續)座標點，但是因為只有一次象限轉換，因此只有算一次交點數量。由於軸切點並不會跨越象限，因此不會讓淹水區間被斷開而造成影響，所以不需加入以下討論。以下圖 4-7 來舉例說明，假設點序列的走向為順時針，各象限(依序為左上、右上、右下、左下)的進入及離開次數為各 1 次，因此各象限的交點數量為(2, 2, 2, 2)。我們以圖示空心圓包圍實心圓⊙表示淹水空間的交點，以下分別針對兩種情況進行討論。

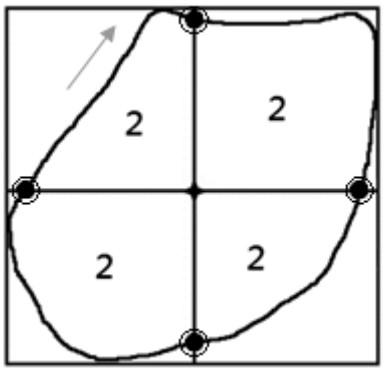


圖 4-7　中心點在淹水區間內，交點數量為(2, 2, 2, 2)

Case I 中心點在淹水區間內的情況：

因為中心點在淹水區間內的話，各象限皆會有非空的淹水子區間，且離開和進入的次數必成對，所以各象限的交點數量必為偶數。最基本的淹水區間圖形就如上圖 4-7 所示，交點數量為(2, 2, 2, 2)。而交點數量為(4, 2, 2, 2)時不合理，因為從交點數量為(2, 2, 2, 2)變為(4, 2, 2, 2)，僅有左上象限增加兩個交點數量，但其他象限的交點數量卻沒有改變，因此交點數量(4, 2, 2, 2)無法成像。

當其中一個象限交點增加，必定會影響相鄰象限的交點數量，如圖 4-8 在左上象限加入兩個交點後，交點數量會由(2, 2, 2, 2)增加為(4, 4, 2, 2)。也就是，交點數量為 2 的象限進出各 1 次，而交點數量為 4 的象限進出各 2 次，但是如此可觀察到在右上象限的點序列被斷開而非單閉區間，因此就不再進行處理，直接回傳整個 MBR。

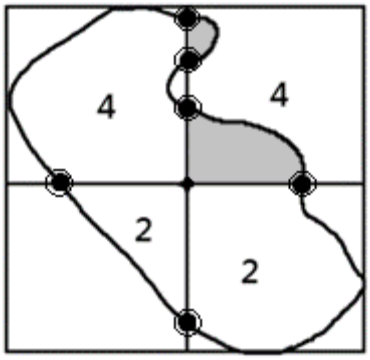


圖 4-8　圖形被斷開，交點數量為(4, 4, 2, 2)

由上述交點數量(2, 2, 2, 2)、(4, 2, 2, 2)、(4, 4, 2, 2)可以得知，任意象限交點增加數量必為 2 的倍數且加入交點後必會影響鄰近一個象限的交點數量，可以推論出合理的交點增加數量必為 4 的倍數，因此交點數量(4, 4, 4, 2)明顯可以得知不合理且無法成像。接下來，合理的交點數量(4, 4, 4, 4)如下圖 4-9 所示，可看出右上及左下象限的點序列皆被斷開而非單閉區間，因此就不再進行處理，直接回傳整個 MBR。交點數量(4, 4, 4, 4)以上的狀況與前述相似，可依此類推不用再討論。

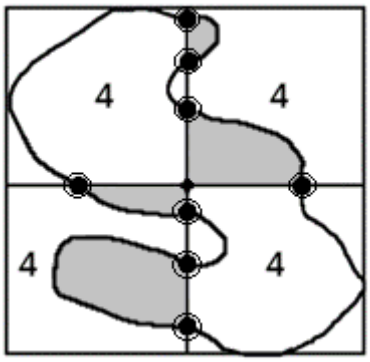


圖 4-9　圖形被斷開，交點數量為(4, 4, 4, 4)

Case II 中心點在淹水區間外的情況：

如 **Case I** 所述，各象限的交點數量必為偶數，但與 **Case I** 不同之處在於，**Case II** 有可能其中一個象限是不存在淹水區間如下圖 4-10，交點數量為(2, 4, 2, 0)，此時為單閉區間。

接下來增加 4 個交點數量，一個可能如下圖 4-11 所示，交點數量為(2, 4, 4, 2)，為單閉區間。另一個可能的方式，是將左上象限再加入 2 個交點的話，交點數量為(4, 6, 2, 0)，如下圖 4-12 所示，可以看出左上象限加入 2 個交點後，會造成淹水區間形成非單閉區間的情況。

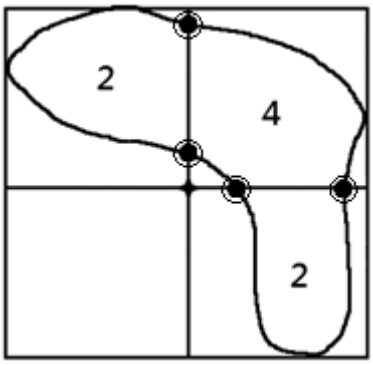


圖 4-10　交點數量為(2, 4, 2, 0)

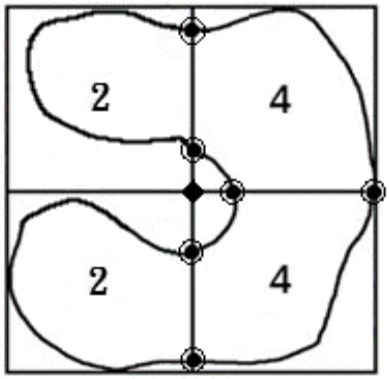


圖 4-11　交點數量為(2, 4, 4, 2)

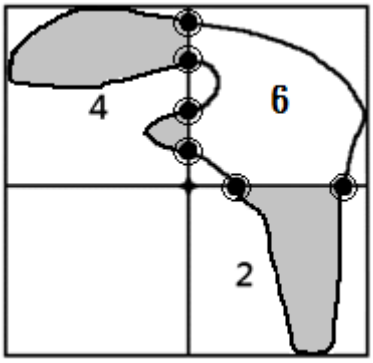


圖 4-12　圖形被斷開，交點數量為(4, 6, 2, 0)

接下來，若是由圖 4-11 使左上和左下象限同時增加交點，交點數量(4, 4, 4, 4)就如下圖 4-13 所示，左下象限的子區間不為單閉區間，所以不繼續分割。

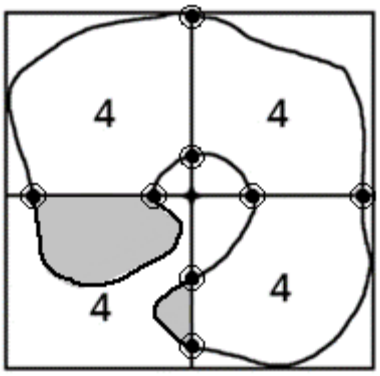


圖 4-13　圖形被斷開，交點數量為(4, 4, 4, 4)

至此，我們發現若再增加交點，都會形成非單閉區間的子區間，所以不再討論。

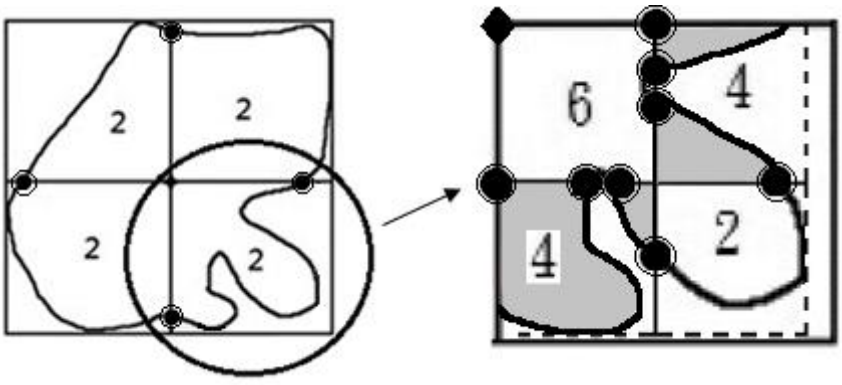
綜合以上兩個 Case 的討論，我們發現當中心點落在淹水區間內，可能的交點數量組合只能為(2, 2, 2, 2)，而當中心點落在淹水區間外，可能的交點數量組合只能為(2, 4, 2, 0)及(2, 4, 4, 2)，但是這些數目的順序可以對調。所以，我們提出以下兩個性質，此二性質在我們從頭依序走訪淹水序列時，會永遠成立：

[性質 4-1]　當中心點落在淹水區間內，各象限的交點數量必須≦2 個。

[性質 4-2]　當中心點在淹水區間外，交點數量為 4 個的象限數必須≦2 且各象限的交點數量必須≦4 個。

以上的討論是針對原始的淹水圖形，以下我們舉例說明，在遞迴處理才出現非單閉區間的情形。

第一種情況是淹水區間原本未斷開，其交點數量為(2, 2, 2, 2) 如下圖 4-14 (a)，而右下象限經過遞迴分割後如下圖 4-14 (b)所示，其交點數量為(6,4,2,4)，可以看出虛線框(亦即該象限內淹水點序列的 MBR)內的右上象限及左下象限造成斷開的情況，因此就不再進行分割處理，直接回傳整個右下象限的 MBR。

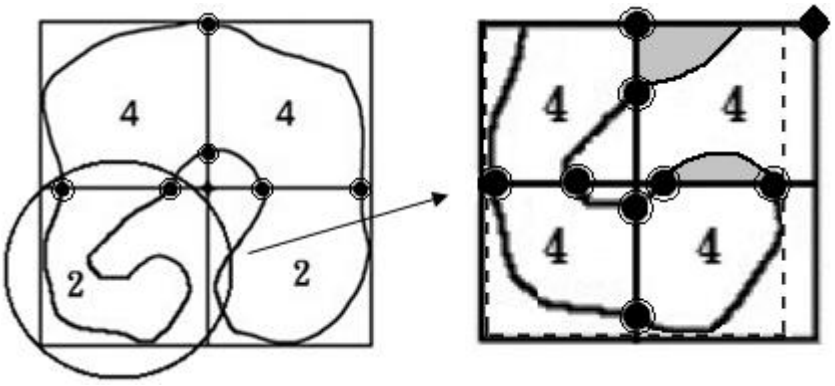


(a)　　(b)

圖 4-14　遞迴後被斷開(子象限的中心點在淹水區間內)

另一種情況是淹水區間原本未斷開，其交點數量為(2, 2, 4, 4)，如下圖 4-15 (a)，而左下象限經過遞迴分割後，其交點數量為(4, 4, 4, 4)，可以看出虛線框內

的右上象限造成斷開的情況如下圖 4-15 (b)所示，因此就不再進行分割處理，直接回傳整個左下象限的 MBR。



(a) (b)

圖 4-15　遞迴後被斷開(子象限的中心點在淹水區間外)

4.3 QuadThreshold 演算法

此作法的特色是給定兩個處理門檻值，首先 MaxRatio 代表的意義是淹水區間的面積已經很接近分割區塊的面積，而 MinArea 代表的意義是分割區塊的面積已經夠小，兩者皆不需再繼續處理。也就是，當淹水區間面積已經超過 MaxRatio，或者分割區塊面積已經小於 MinArea，就直接回傳該分割區塊。於圖 4-16 中，上方箭頭指向的粗框面積小於給定的最小面積，而下方箭頭指向的粗框內淹水區間面積已超過整個粗框的八成面積，所以兩者皆會直接回傳粗框。若尚未滿足上述兩個條件之一，就再將點序列分為四個象限再各別遞迴處理。

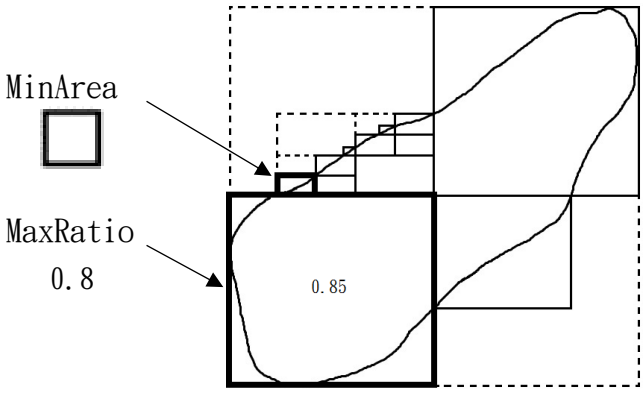


圖 4-16　符合門檻值示意圖

在此先說明程式使用到的變數。FO 代表一個淹水區間，其中包含**[定義 3-2]**中 8 個方位的結構，而每個象限又以 Inpts 記錄其交點數量及 MaxFour 對應到**[性質 4-2]**，是用來統計交點個數為 4 個的象限個數。注意，此處指的交點數量是跨越象限的次數，所以連續的軸點只算一次，另外會以變數 List 紀錄每個象限的子區間點序列。flagW 對應到**[性質 4-1]**及**[性質 4-2]**，是用來判斷中心點在淹水

區間內(其值為“in”)或外(其值為“out”)的旗標。至於 PonB 則是用來統計軸點數量，以處理連續的軸點。

本作法如同 **BaseMIR** 方法，會呼叫 **FindDirection** 函數來找出座標點對應的方位、**IFOnAxis** 函數來判斷座標點是否在軸上、**IFCross** 函數來判斷下一點是否跨越象限，及利用 **GetInterPoint** 函數來求取交點。與 **BaseMIR** 方法不同之處，在於必須配合**[性質 4-1]**和**[性質 4-2]**，記錄各象限的交點數量以決定是否要停止遞迴處理。另外，把軸點再細分為軸交點和軸切點，前者必須加入到兩個象限的點序列，而後者只需加入到一個象限中。

以下說明軸交點的判斷方式，以下圖 4-17 舉例說明。假設目前非軸點的註標 *i* 為 6，而軸點數量 PonB 為 2，包含點 4 及點 5，利用算式 *i*-PonB-1 計算出 6-2-1=3，則可得到前一個象限的最後一個座標點註標為 3。由於點 3 和點 6 兩點位於不同象限因此跨越成立，代表該軸點為軸交點，且跨越次數為 1 次。反之，若點 3 和點 6 兩點位於相同象限則跨越不成立，代表該軸點為軸切點，如下圖 4-18 所示。

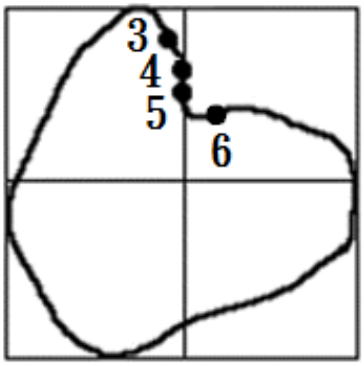


圖 4-17　軸上點判斷示意圖

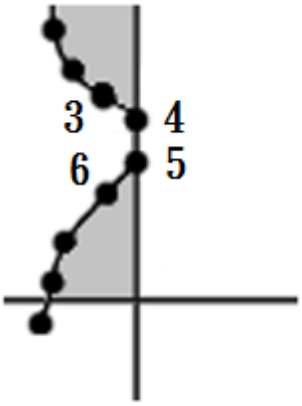


圖 4-18　軸切點判斷示意圖

QuadThreshold 演算法如圖 4-19。首先，以淹水區間 ***S*** 的點序列設定分割區塊 MBR，L02-L05 當分割出的淹水區間面積已經很接近分割區塊面積，或者分割區塊面積已經夠小，就將分割區塊加入輸出序列中並回傳。若繼續處理，在進行處理點序列前，L06 先指定旗標 flagW 的預設值為“in”、初始化輸出序列

OutputList 為空、點序列註標 i 為 0，L07-L09 會呼叫 Boost Libraries[BD98]的 Within 函數，依照中心點是在淹水區間的內或外，來確定旗標 flagW 的值並決定後續處理的方式。

L10-L60 依序處理淹水資料的點序列。L11 會將統計軸點數量 PonB 設為 0，避免再次統計軸點數量時數量錯誤。L12-L15 若點 i 是軸點，就累加落在軸上的座標點數量 PonB 如圖 4-17 和圖 4-18。L16 當有軸點存在時，必須判斷軸點為軸交點或軸切點。L17 呼叫 **IFCross** 函數，透過前一個象限的最後一個非軸點 $\boldsymbol{S}[i\text{-PonB-1}]$和目前象限的第一個非軸點 $\boldsymbol{S}[i]$，來判斷是否跨越象限，若是則代表為軸交點。首先 L18-L23 處理前一個象限，先呼叫 **CountInpts** 函數來累加該象限的交點數，L20-L23 再呼叫 **Property** 函數判斷是否符合**[性質 4-1]**或**[性質 4-2]**，若不符合則於 L21-L22 直接回傳 MBR。目前象限的處理於 L24-L29 同上。接下來，L30-L32 呼叫 **AddAxisPoint** 函數依序將所有軸交點加入到兩側象限中，加入軸點後已完成前一象限的處理，因此 L33 需要呼叫 **AddCenter** 函數來判斷是否需要將中心點加入到前一象限的點序列中。L34-L38 處理軸切點，由於軸點的前後兩點位於相同象限，直接依序將所有軸切點加入到對應的象限中，並不需要做**[性質 4-1]**和**[性質 4-2]**的判斷。

處理完軸交點或軸切點後，接下來處理點 i 不是軸點的情況，L40-L41 將點 i 加入對應象限的點序列中，L42-L58 呼叫 **IFCross** 函數以中心點來判斷點 i 到點 i+1 是否跨越象限，若兩點在不同象限，則跨越成立，進行交點的處理。首先，L43-L50 處理目前象限，呼叫 **CountInpts** 函數增加並統計交點交點數量，在計算及加入交點前，L44-L47 先呼叫 **Property** 函數判斷是否符合繼續處理的性質，若不符合性質就將整個 MBR 加入輸出序列並回傳。反之，L48-L49 呼叫 **GetInterPoint** 函數計算兩點之間的交點 $\boldsymbol{P}_{Inpt}$，並將交點加入到目前象限中，L50 呼叫 **AddCenter** 函數，判斷中心點是否在淹水區間內，若是的話也必須將中心點加入到點序列中。

下一個象限的處理於 L51-L58 同上所述，先找出其象限，統計交點數量後判斷性質，若符合性質才將交點加入對應象限的點序列中，即完成逐點判斷象限的步驟。L61-L62 判斷點序列的最後一點的象限並加入對應象限的點序列中。最後，L63-L67 將四個象限座標點數量大於 0 的點序列分別遞迴呼叫 **QuadThreshold** 演算法處理，並將 4 個象限回傳的淹水區塊加入輸出序列 OutputList 並輸出。

Quad 方法會修正圖 2-8 Main 演算法如下：

L04 OutputList.add(**QuadThreshold** (***S***, MaxRatio, MinArea))

演算法名稱：**QuadThreshold** 輸入：***S***, MaxRatio, MinArea 輸出：OutputList
L01 MBR←SetMBR(***S***)

```
L02   If( S.area / MBR.area > MaxRatio || MBR.area < MinArea ) //門檻值判斷
L03     OutputList←MBR
L04     return OutputList
L05   End If
L06   Initialize flagW← "out"; OutputList←null; i←0
L07   If( Within(MBR.center, S) ) //中心點在淹水區間內
L08     flagW← "in"
L09   End If
L10   while(i < S.size() - 1)   //逐點判斷方位，加入 4 個象限對應的點序列
L11     PonB←0
L12     while( IFOnAxis(S[i], MBR.center) )
L13       PonB++
L14       i++
L15     End while
L16     If(PonB)   //軸點數量大於 0
L17       If( IFCross(S[i-PonB-1], S[i] , MBR.center) ) //軸上跨越，軸交點
L18         D←FindDirection(S[i-PonB-1], MBR.center);
L19         FO←CountInpts(D, FO)
L20         If( not Property(flagW, FO.D.Inpts, FO.D.MaxFour) ) //判斷性質
L21           OutputList.add(MBR)
L22           return OutputList
L23         End If
L24         D←FindDirection(S[i], MBR.center);
L25         FO←CountInpts(D, FO)
L26         If( not Property(flagW, FO.D.Inpts, FO.D.MaxFour) ) //判斷性質
L27           OutputList.add(MBR)
L28           return OutputList
L29         End If
L30         For p ← i-PonB to i-1   //從第 1 個軸交點開始加入到兩邊象限
L31           FO←AddAxisPoint(FO, S[p] , MBR.center)
L32         End For
```

```
L33          FO←AddCenter(flagW, FO, S[i-PonB-1] , MBR.center)
L34        Else    //軸上未跨越，為軸切點
L35          For p ← i-PonB to i-1   //從第 1 個軸切點開始加入到目前象限
L36            FO.D.List.add(S[p])
L37          End For
L38        End If
L39      End If //End if (PonB)
L40      D←FindDirection(S[i], MBR.center)
L41      FO.D.List.add(S[i])   //加入目前座標點 S[i]
L42      If( IFCross(S[i], S[i+1] , MBR.center) )   //跨越
L43        FO←CountInpts(D, FO)   //先累加交點 1 次並統計數量
L44        If( not Property(flagW, FO.D.Inpts, FO.D.MaxFour) )   //判斷性質
L45          OutputList.add(MBR)
L46          return OutputList
L47        End If
L48        P_Inpt←GetInterPoint(S[i], S[i+1], MBR.center)   //找交點
L49        FO.D.List.add(P_Inpt)   //加入交點到目前象限
L50        FO←AddCenter(flagW, FO, S[i] , MBR.center)
L51        D←FindDirection(S[i+1], MBR.center)   //處理下一個象限
L52        FO←CountInpts(D, FO)
L53        If( not Property(flagW, FO.D.Inpts, FO.D.MaxFour) )   //判斷性質
L54          OutputList.add(MBR)
L55          return OutputList
L56        End If
L57        FO.D.List.add(P_Inpt)   //加入交點到下個象限
L58      End If   // End Else If( IFCross )
L59      i++
L60    End while   //while(i < S.size()-1)
L61    D←FindDirection(S[i-1], MBR.center)   //判斷最後一點的象限
L62    FO.D.List.add(S[i-1])   //加入最後一點
L63    For each D in (NW, NE, SE, SW)
```

```
L64     If(FO.D.List.size() > 0)
L65         OutputList.add(QuadThreshold(FO.D.List, MaxRatio, MinArea) )
L66     End If
L67   End For
L68   return OutputList
```

圖 4-19　QuadThreshold 演算法

[範例 4-1]　將**[範例 3-1]**的淹水區間依據 **Quad** 作法分割，設定 MaxRatio 為 0.8 及 MinArea 為一單位長度 CellEdge 平方的面積，如圖 4-20 (a)所示，其中淹水區間的 MBR 以粗框表示。由於原始淹水區間面積比例約為 0.4，且 MBR 面積大於最小處理面積 MinArea，兩者皆尚未達到門檻值，需要將淹水區間範圍(0, 0, 10.2, 9.5)分割為四個象限各別遞迴。以左上象限範圍(0, 4.75, 5.1, 9.5)為例，在 L01 會重新取 MBR，其範圍為(1.1, 4.75, 5.1, 6.75)，如圖 4-20 (b)粗框所示，而虛線框即為遞迴前左上象限的外框。由於此次淹水區間與其 MBR 的面積比例約為 0.55 且 MBR 面積大於 MinArea，尚未達到門檻值，因此需要再分割為 4 個子區間各別遞迴。此次遞迴分割後的左上象限範圍(1.1, 5.75, 3.1, 6.75)，如圖 4-20(c)虛線框所示，將其內的淹水區間重新取 MBR，範圍為(3, 5.75, 3.1, 6)，其面積比例約為 0.5 小於 MaxRatio，但 MBR 面積已經小於 MinArea，因此直接輸出停止遞迴。另一方面，當我們遞迴處理圖 4-20 (a)的左下象限時，其範圍(0, 0, 5.1, 4.75)，如圖 4-20 (d)粗框所示，由於其面積比例約為 0.85 大於 MaxRatio，因此也會直接輸出。如上所述，完成所有分割後，輸出的淹水區塊如圖 4-20 (e)粗框所示，共 27 個淹水區塊。

接下來我們討論此範例所需的空間計算次數。注意，根據圖 4-19 L02 的 shortcut 寫法，因為大於 MaxRatio 輸出的淹水區塊，面積大小比較次數為 1 次，而因為面積小於 MinArea 輸出的淹水區塊，面積大小比較次數則為 2 次；若需要遞迴處理者，如 L07 所示，還需要 1 次點包含判斷的次數。以圖 4-20 (b)的粗框為例，因為需要繼續分割，所以面積大小比較的次數為 2 次、點包含判斷的次數為 1 次；圖 4-20 (c)因面積達到 MinArea 輸出，面積大小比較的次數為 2 次；圖 4-20 (d)因面積比例達到 MaxRatio 輸出，面積大小比較的次數為 1 次。

我們將原始淹水區間以編號 1 代表，其四個象限由左上順時針到左下分別以 1.1、1.2、1.3 及 1.4 代表，依此類推。每個淹水區間所需要的空間計算次數，如下表 4-1 所示，**Quad 方法**的空間計算次數總共為 71 次。注意，L64 所代表的 size 為點序列中的座標點

數量，並非淹水區間的面積，因此不會列入面積大小比較次數中。

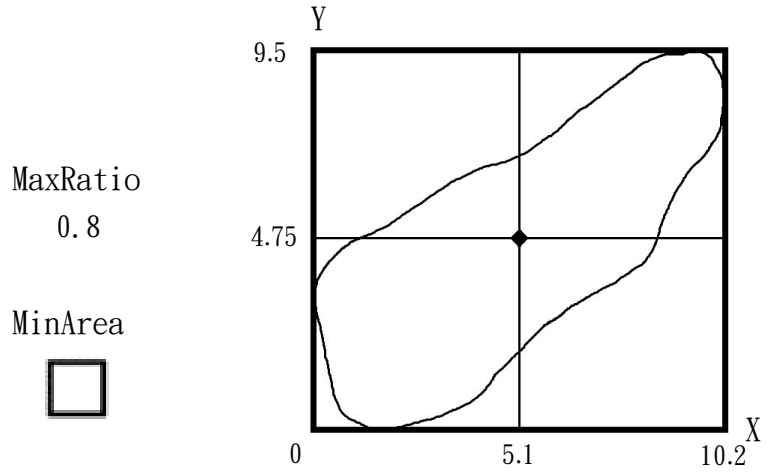
表 4-1　空間計算次數範例

編號	1	1.1	1.1.1	1.1.2	1.1.2.1	1.1.2.2	1.1.2.3	1.1.2.4	1.1.3	1.1.4	1.1.4.2
面積大小比較	2	2	2	2	2	2	1	1	1	2	2
點包含判斷	1	1	0	1	0	0	0	0	0	1	0

編號	1.1.4.3	1.1.4.4	1.2	1.2.1	1.2.1.2	1.2.1.3	1.2.1.4	1.2.2	1.2.3
面積大小比較	1	2	2	2	2	1	2	1	2
點包含判斷	0	0	1	1	0	0	0	0	1

編號	1.2.3.1	1.2.3.2	1.2.3.3	1.2.3.4	1.2.4	1.3	1.3.1	1.3.2	1.3.2.1
面積大小比較	1	2	2	1	1	2	1	2	1
點包含判斷	0	0	0	0	0	1	0	1	0

編號	1.3.2.2	1.3.2.3	1.3.2.3	1.3.4	1.3.4.1	1.3.4.2	1.3.4.4	1.4	總計
面積大小比較	2	2	2	2	1	2	2	1	61
點包含判斷	0	0	0	1	0	0	0	0	10



(a)

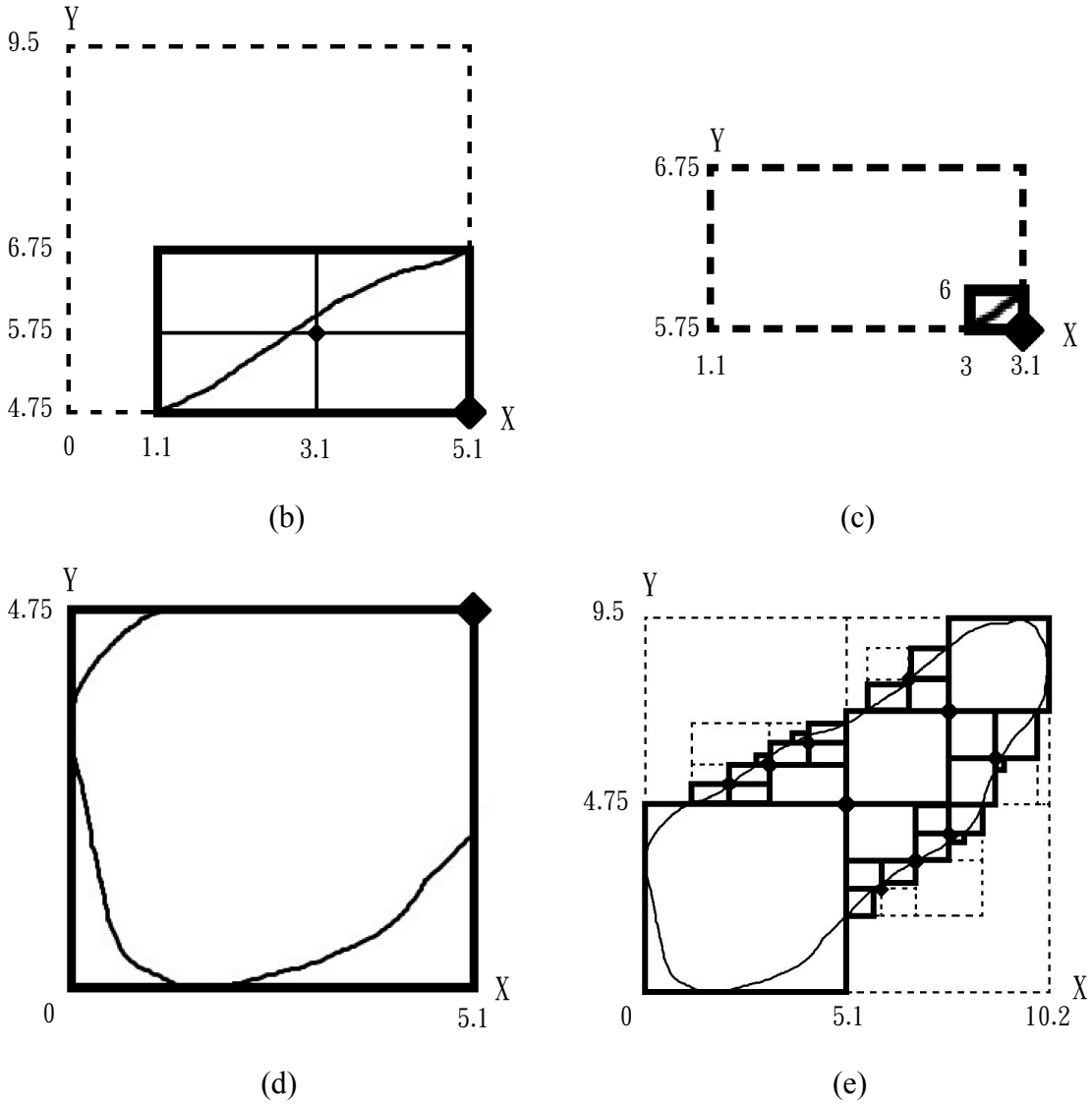


圖 4-20 Quad 方法分割範例

與**[範例 3-1]**及**[範例 3-2]**相比，**Quad** 作法優點是當分割區塊面積較大時可以大幅降低空間相交判斷，如圖 4-20 (b)的左下象限所示。而在空間計算上，**Quad** 方法是分割淹水區間點序列來計算面積，並透過面積大小比較來輸出淹水區塊。在空間計算的次數由 **BaseFS 方法**的 111 次及 **BaseMIR 方法**的 76 次降至 **Quad** 方法的 71 次，最多降低近五成，而輸出的淹水區塊數量由 **BaseFS 方法**的 63 個及 **BaseMIR 方法**的 34 個降至 **Quad** 方法的 27 個，也是最多降低近六成，由此可見在判斷空間相交次數及輸出淹水區塊數量皆優於前兩種分割方法。

以下列出本作法額外設計的幾個輔助程式。演算法 **CountInpts** 用來統計交點數量，如圖 4-21。透過淹水區間物件 FO，在對應的象限 ***D*** 增加一個交點數量，並將象限已達 4 個交點數量的變數 MaxFour 增加 1，最後回傳淹水區間物件 FO。

演算法名稱：**CountInpts** 輸入：***D***, FO 輸出：FO
L01　FO.***D***.Inpts++ L02　**If**(FO.***D***.Inpts = 4) L03　　FO.***D***.MaxFour++ L04　**End If** L05　**return** FO

圖 4-21　CountInpts 演算法

演算法 **Property** 是在更新交點數量後，用來判斷該象限的交點數量是否符合**[性質 4-1]**或**[性質 4-2]**，如下圖 4-22。透過輸入的旗標 flagW 分為 **CaseI** 及 **CaseII**，若符合性質則回傳 TRUE，反之則回傳 FALSE。

演算法名稱：**Property** 輸入：flagW, Inpts, MaxFour 輸出：bool
L01　**If**(flagW = "in" & Inpts≦2)　// **CaseI** L02　　**return** TRUE L03　**Else If**(flagW = "out" & (Inpts≦4 & MaxFour≦2))　// **CaseII** L04　　**return** TRUE L05　**End If** L06　**return** FALSE　//不符合 **CaseI** 及 **CaseII**

圖 4-22　Property 演算法

演算法 **AddAxisPoint** 是將落在軸上的座標點，同時加入到該軸兩側象限的序列中，如下圖 4-23。首先呼叫 **FindDirection** 函數找出點 ***P*** 的方位，例如回傳的方位為 N，代表為北方的 Y 軸，則將點 ***P*** 同時加到兩側象限 NW 及 NE 的點序列中，加入完成後回傳淹水區間物件 FO。

演算法名稱：**AddAxisPoint** 輸入：FO, ***P***, MBR.center 變數：Axis 輸出：FO //淹水區間物件

```
L01  Axis←FindDirection(P, MBR.center)
L02  If(Axis = "N")
L03      FO.NW.List.add(P)
L04      FO.NE.List.add(P)
L05  Else If(Axis = "S")
L06      FO.SW.List.add(P)
L07      FO.SE.List.add(P)
L08  Else If(Axis = "E")
L09      FO.NE.List.add(P)
L10      FO.SE.List.add(P)
L11  Else If(Axis = "W")
L12      FO.NW.List.add(P)
L13      FO.SW.List.add(P)
L14  End If
L15  return FO
```

圖 4-23　AddAxisPoint 演算法

演算法 **AddCenter** 是在加入交點後，將中心點補入對應的象限中，如下圖 4-24。透過傳入的值 flagW 來判斷，若其值為"in"，則代表中心點在淹水區間內，需要補入中心點，並將淹水區間物件 FO 回傳。

```
演算法名稱：AddCenter
輸入：flagW, FO, P, MBR.center
變數：D
輸出：FO
```

```
L01  If(flagW = "in")   //中心點在淹水區間內
L02      D←FindDirection(P, MBR.center)
L03      FO.D.List.add(MBR.center)   //加入中心點到前點的點序列
L04  End If
L05  return FO
```

圖 4-24　**AddCenter** 演算法

第5章 實驗

在本章，我們首先介紹本論文各個方法的實作方式，接著說明實驗使用到的資料集的來源，以及程式輸入的資料格式。接下來會進行一系列的實驗比較各個方法。至於進行實驗的環境，我們以個人電腦作為實驗的環境，i7-2600 四核心且核心時脈為 3.4GHz，而記憶體為 16GB，所採用的作業系統為 64 位元的 Windows 7 專業版。實驗方式是在分割淹水區間模組和建立 R-tree 模組各執行十次，然後分別取後九次執行時間的平均。在移除淹水道路模組及規劃路徑模組，每組起迄點執行十次，共使用三十組起迄點，將這三十組先分別取後九次執行結果的平均後，最後再取三十組執行結果的平均。

5.1 實作方式與各方法說明

本論文提出的淹水區間分割方法（包含 **BaseFS**、**BaseMIR** 及 **Quad**），以及延續使用[洪 16]的建立 R-tree 模組、移除淹水道路模組及規劃路徑等模組，皆是以 Microsoft Visual C++ 2015 進行實作，而輸出展示的 Google Map 畫面則是使用 HTML 搭配 JavaScrInpt 來進行實作。

如之前所討論，我們會需要進行空間計算，本論文是以 Boost C++ Libraries 的 Geometry 類別來實作空間計算，而空間計算在本論文中包含以下三種。第一種為空間相交，是使用 **intersects** 函數，利用兩個多邊型的封閉區間來判斷是否相交。第二種為點包含，是使用 **within** 函數，利用一個座標點與一個多邊型的封閉區間來判斷該點是否被多邊形的封閉區間完全包涵。第三種是面積比較，是使用 **area** 函數，來計算由座標點所形成封閉區間的面積。

以下說明三種分割方法的基本精神與其所需的空間計算次數。**BaseFS** 方法是根據固定長度 CellEdge 來分割淹水區間，使用到面積大小比較及空間相交判斷。**BaseMIR** 方法是當中心點在淹水區間中的情況，會先求取由距離中心點最近的 4 個交點或軸點所構成的 MIR 後，其餘的淹水區間再分為 4 個區塊來呼叫 **BaseFS** 方法來完成分割，使用到面積大小比較、點包含判斷及空間相交判斷；若中心點在淹水區間外的狀況，就會直接呼叫 **BaseFS** 方法來完成分割。**Quad** 方法是將未達門檻值的淹水區間再遞迴分割為 4 個子區間，使用到面積大小比較及點包含判斷。

另外，我們還會和其他 2 種方法比較。首先，**MBR** 方法是直接求取每個淹水區間的 MBR 不進行分割，直接傳送給建立 R-tree 模組。其次，**SegmentRtree** 方法是將每個淹水區間的點序列，將鄰近的 2 個點連成一個線段來取 MBR 以建立 R-tree。但 **SegmentRtree** 所輸出的淹水區塊僅包含淹水區間的周圍，會有道路落在淹水區間內的情況，如下圖 5-1 圈選的道路所示，造成該道路被判定為可以走的路，並將其加入到未淹水道路中使得未淹水道路數量增加，因此可能會影響到移除淹水道路模組及規劃路徑模組的效率。另外注意，此二種對照方法皆不

需要上述的空間計算。

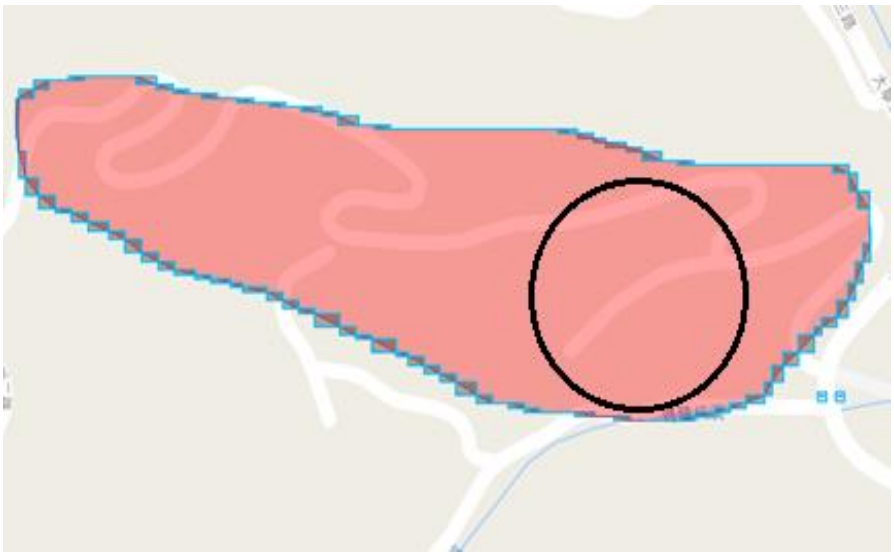


圖 5-1　**SegmentRtree** 方法之輸出結果展示

5.2　實驗資料說明

本節介紹實驗所使用到的資料集，淹水資料的來源是國家災害防救科技中心提供的各縣市『一日暴雨 600mm 淹水潛勢圖』[3]，以下使用基隆市的淹水潛勢圖來說明淹水區間的資料如何取得，如下圖 5-2。首先下載並安裝 GoogleEarth Pro[4] 後，開啟程式並新增一個資料夾來歸納以下所建立的物件。第一步是點選上方工具列中的第五個圖示【新增圖像疊加層】選項，接著在編輯圖像疊加層視窗中，點選【瀏覽】按鈕，將淹水潛勢圖疊加到 GoogleEarth 的地圖上，如下圖 5-3。接著，我們透過調整【清晰度】的拖曳選項，從不透明調整為半透明，並拖曳淹水潛勢圖的綠色邊角，使該圖片邊界與 GoogleEarth 的地圖完全切齊，與背景切齊後，再將【透明度】調整回不透明後按確定即完成圖像疊加的動作。

[3] 從國家災害防救科技中心網站: https://dmap.ncdr.nat.gov.tw/，進入「資料分析與下載」 的列表中，可以選擇「縣市等級」或「城鎮等級」來進入下載各類「高解析地圖籍」。

[4] GoogleEarth Pro 下載網站: https://www.google.com/earth/download/gep/agree.html

圖 5-2　基隆市淹水潛勢圖

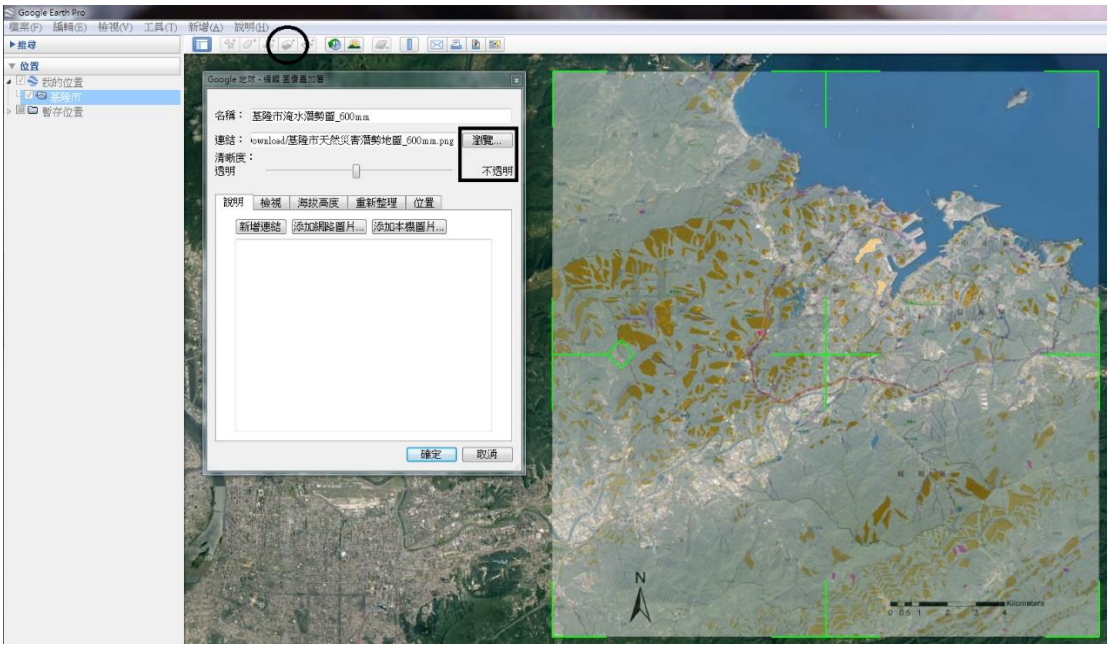


圖 5-3　圖像疊加範例

第二步是點選上方工具列中的第三個圖示【新增多邊形】選項來新增一個多邊形物件，然後以滑鼠左鍵沿著單一淹水區間拖曳，並完整將其包圍後，以數字命名後即完成圈選淹水區間的動作，如下圖 5-4。第三步是將整個基隆市資料夾另存為.kml 檔案。以 Notepad++開啟該檔案，可以看到資料夾對應到<Folder>標籤，包含<GroundOverlay>標籤及<Placemark>標籤。<GroundOverlay>即為我們所疊加的圖層，其子元素<LatLonBox>記錄該圖層的經緯度範圍。而每個多邊形各為一組<Placemark>，<name>標籤即為多邊形名稱，<coordinates>標籤裡的資料即為淹水區間的點序列，如下圖 5-5。此範例包含 2 個淹水區間，注意到，每

一個點是以「經度座標,緯度座標,0」表示。

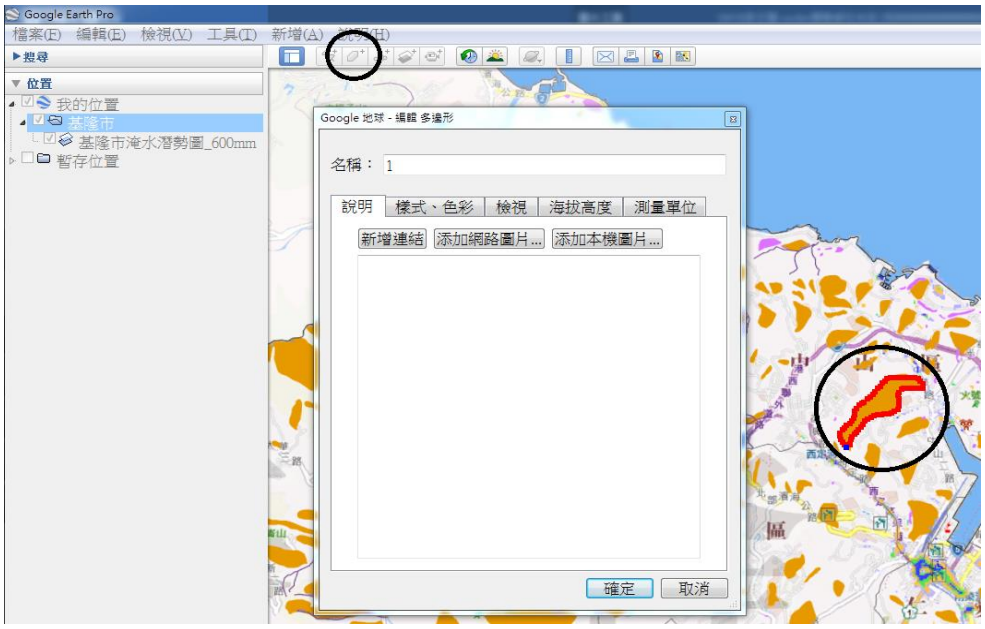


圖 5-4　以多邊形圈選淹水區間

```
<Folder>
    <name>基隆市</name>
    <open>1</open>
    <GroundOverlay>
        <name>基隆市淹水潛勢圖_600mm</name>
        <visibility>0</visibility>
        <Icon>
            <href>淹水潛勢圖600mm/基隆市/基隆市天然災害潛勢地圖_600mm.png</href>
            <viewBoundScale>0.75</viewBoundScale>
        </Icon>
        <LatLonBox>
            <north>25.20079105621199</north>
            <south>25.02485888451729</south>
            <east>121.8139938188922</east>
            <west>121.6230976542433</west>
        </LatLonBox>
    </GroundOverlay>
    <Placemark>
        <name>1</name>
        <styleUrl>#msn_ylw-pushpin</styleUrl>
        <Polygon>
            <tessellate>1</tessellate>
            <outerBoundaryIs>
                <LinearRing>
                    <coordinates>
                        121.7300657899805,25.14138792031877,0 121.7300620962289,25.14154667123231,0 121.7301402822339,25.14194368468331,0 121.7303
                    </coordinates>
                </LinearRing>
            </outerBoundaryIs>
        </Polygon>
    </Placemark>
    <Placemark>
        <name>2</name>
        <styleUrl>#m_ylw-pushpin</styleUrl>
        <Polygon>
            <tessellate>1</tessellate>
            <outerBoundaryIs>
                <LinearRing>
                    <coordinates>
                        121.7365430818867,25.13178034358138,0 121.7365991815162,25.13172606969085,0 121.7367655200941,25.13151035790844,0 121.7369
                    </coordinates>
                </LinearRing>
            </outerBoundaryIs>
        </Polygon>
    </Placemark>
</Folder>
```

圖 5-5　淹水資料.kml

在本論文中我們透過 C#所提供剖析 XML 的套件來將.kml 檔案中所有<coordinates>標籤內的資料取出，並輸出為.txt 的淹水資料檔案，其中每一行代表一個淹水區間的經緯度座標點序列，中間皆以空白隔開。資料集能以縣市或鄉鎮為單位，而一個淹水資料集可以包含多個淹水區間，依照實驗需求做調整。以基隆市為例，基隆市邊界內包含 436 個淹水區間，因此淹水資料檔案就會有 436 行，如下表 5-1 所示。

表 5-1　淹水資料檔案

行 數	內 容
1	121.7300657899805 25.14138792031877 121.7300620962289 25.14154667123231 …
2	121.7365430818867 25.13178034358138 121.7365991815162 25.13172606969085 …
n	P_1.Lng P_1.Lat P_2.Lng P_2.Lat … P_n.Lng P_n.Lat

實驗所使用的資料集為臺北市及基隆市，依序列出各項參數如表 5-2。淹水密度是使用淹水區間面積總和除以市區的面積。平均淹水空間面積為淹水區間面積總和除以淹水區間總個數。淹水區間總個數，代表該市區包含的淹水區間個數。淹水區間面積總和，是將該市區的所有淹水區間面積加總。而道路資料，是由交通部路網數值圖服務網[5]提供的路網圖取得，臺北市包含 42838 條道路，基隆市包含 6619 條道路。

選擇這兩個資料集的主要原因，在於我們希望市區的淹水密度不超過市區面積的 2 成，在規劃迴避淹水路徑時的情況會比較理想。雖然在淹水區間總個數上沒有明顯的差距，但是在市區淹水密度、淹水區間面積總和及道路個數上都有顯著差距，因此我們選擇這兩個較極端的資料集來作為實驗資料集。

表 5-2　資料集之各項參數

資料集名稱	市區 淹水密度		平均淹水區間面積 (km^2)		淹水區間 總個數	淹水區間 面積總和 (km^2)		市區面積(km^2)		道路個數	
臺北市	0.038	小	0.025	小	411	10.298	小	270.212	大	42838	多
基隆市	0.143	大	0.043	大	436	19.02	大	132.98	小	6619	少

5.3　門檻值的設定說明

本節說明門檻值參數的訂定方法。首先要決定參數 CellEdge，其為 MinArea 的邊長。我們透過程式來計算資料集臺北市及基隆市中所有淹水區間的面積，並且找出最小的面積，對其開根號以求取最小 MBR 的邊長。計算結果如下，臺北市為 48 公尺，基隆市為 55 公尺，因此後續的實驗，我們就將參數值 CellEdge 取一個中間的整數 50 公尺來作為預設值。接下來，要決定 **Quad** 方法中的門檻值 MaxRatio，也就是面積比例的門檻值。我們透過淹水覆蓋率來決定 MaxRatio，淹水覆蓋率的分母為淹水區間總面積，分子為分割出的淹水區塊面積總和，如公式(1)所示：

$$\text{淹水覆蓋率} = \frac{\Sigma\text{分割出的淹水區塊面積}}{\Sigma\text{淹水區間面積}} \qquad (1)$$

注意到淹水覆蓋率越大，則代表分割出的淹水區塊總面積大於原始淹水區間

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

面積越多，代表不能行走的面積越大，因此會希望所選擇的 MaxRatio 值讓淹水覆蓋率越接近 1 越好。

實驗結果如下圖 5-6 所示，選擇 0.55 及 0.95 作為 MaxRatio 值範圍的精神在於，MaxRatio 代表淹水區間面積已經佔有足夠的 MBR 面積，因此 MaxRatio 的設定至少要大於 0.5 且小於 1，才符合 MaxRatio 的概念。由該圖可以看出兩個資料集，當門檻值 MaxRatio 設為 0.55 時，因為在淹水區塊會僅包含約一半的淹水區間即不再繼續分割，所以淹水覆蓋率如預期中為最高的。相反的，當門檻值 MaxRatio 設為 0.95 時，會分割出較小的淹水區塊所以淹水覆蓋率最接近 1，因此在後續實驗我們將 MaxRatio 設為 0.95 來作為預設值。

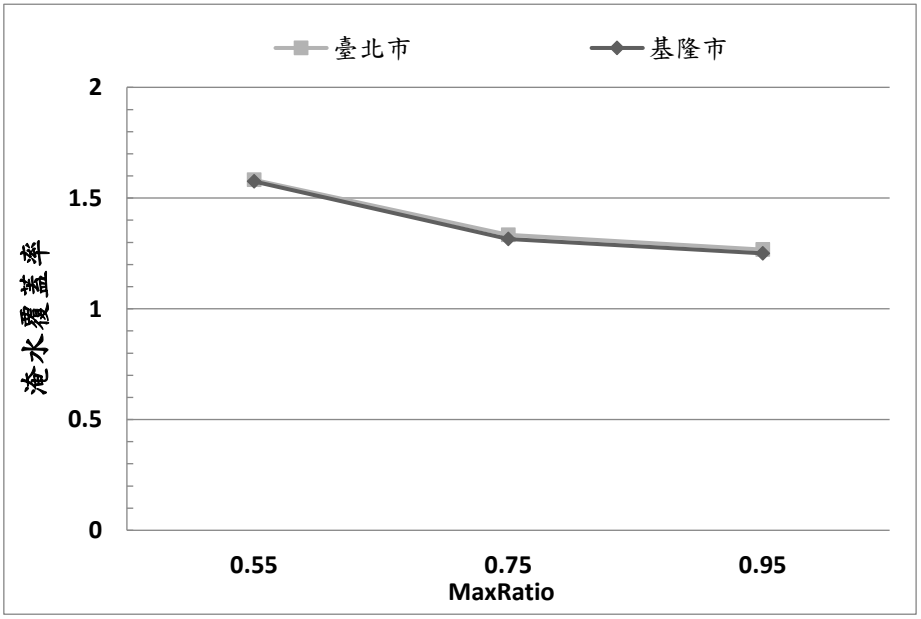


圖 5-6 **Quad** 方法改變 MaxRatio 對淹水覆蓋率之影響

表 5-3 **Quad** 方法改變 MaxRatio 對淹水覆蓋率之影響

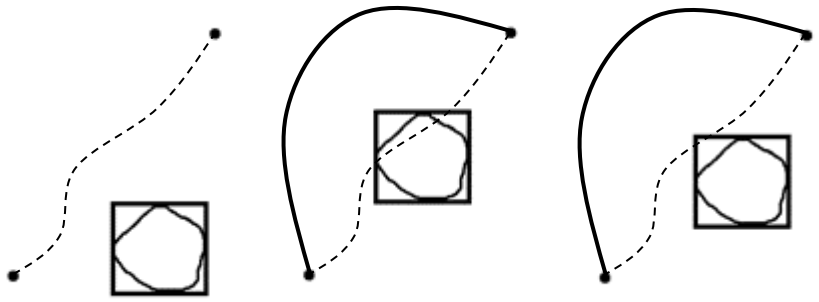
MaxRatio 值	**0.55**	**0.75**	**0.95**
臺北市	1.582	1.334	1.267
基隆市	1.575	1.315	1.25

5.4 路徑長度實驗

本節根據資料集臺北市及基隆市，以隨機的起迄點來比較 **BaseFS**、**BaseMIR**、**Quad**、**SementRtree** 及 **MBR** 五種方法規劃出的路徑。參數值 CellEdge 設為預設值 50 公尺，門檻值 MaxRatio 設為預設值 0.95，order 設為預設值 64。

路徑長度實驗中，我們比較迴避淹水的路徑長度，因此要選用有經過或是在淹水區間附近的起迄點。首先，在臺北市隨機產生 150 組起迄點及基隆市隨機產生 100 組起迄點。接著，透過五種方法其中一種，評估該組起迄點是否規劃前後路徑長度不同且規劃成功，然後從其中取出 30 組。

詳細說明所取出的 30 組起迄點的特性如下圖 5-7，以實心圓圖形●代表起迄點、虛曲線代表起迄點規劃出的原始最短路徑、粗曲線代表起迄點規劃出迴避淹水路徑、不規則圖形代表淹水區間、粗框代表淹水區間 MBR。圖 5-7 (a)代表該組起迄點規劃出的原始最短路徑沒有與淹水區間相交，造成五種方法所規劃的路徑長度皆相同，所以不予取用。圖 5-7 (b)代表淹水區間落在該組起迄點的最短路徑上，淹水區間及 MBR 皆與其相交，因此規劃前後的路徑長度會不同，列入本實驗所使用的起迄點種類之一，但是以不同分割方法，其規劃的路徑長度一定都比原始最短路徑長。圖 5-7 (c)代表淹水區間在該組起迄點的最短路徑附近，雖然淹水區間與最短路徑並沒有直接相交，但淹水區間的 MBR 與最短路徑有相交，因此規劃前後的路徑長度會不同，列入實驗所使用的起迄點種類之一。與圖 5-7 (b)不同之處，在於使用不同分割方法，輸出的淹水區塊大小不同，有可能有些方法規劃的最短路徑會與原始最短路徑相同。



(a)　　(b)　　(c)

圖 5-7　起迄點選擇示意圖

針對每個分割方法，我們計算其平均迴避淹水長度比率如下：

$$\text{平均迴避淹水長度比率} = \frac{\Sigma \frac{\text{迴避淹水路徑長度} - \text{原始最短路徑長度}}{\text{原始最短路徑長度}}}{\text{起迄點組數}} \qquad (2)$$

假設原始最短路徑長度為 10 公里，迴避淹水路徑長度為 15 公里，迴避淹水長度比率的計算結果即為(15-10) / 10 = 0.5。比率代表規劃後的路徑比原始最短路徑多出幾成，因此比率越低顯示規劃出迴避淹水的路徑越短。

臺北市及基隆市的平均迴避淹水路徑長度比率如圖 5-8 所示，依照不同方法列出平均迴避淹水長度比率。首先，可以看到 **MBR** 方法規劃的結果如預期，在臺北市及基隆市中，皆是所有方法中比率最高的，而 **SegmentRtree** 方法規劃的結果，在兩個資料集中，皆是所有方法中比率最低的，略勝於我們所提出 3 種方法的最佳者 **Quad** 方法。仔細分析該三種方法，在臺北市使用 **Quad** 方法分割，僅略勝於其他兩種方法。而在基隆市使用 **Quad** 方法的長度比率雖然和 **BaseFS** 相近，但卻是 **BaseMIR** 的 0.4 倍。因此 **Quad** 方法可以說是三種分割方法中較

為精準的方法。

接下來，比較不同資料集，可以看到基隆市使用 **BaseMIR** 及 **MBR** 方法來分割，其平均迴避淹水長度比率較高，分別為 **SegmentRtree** 方法的 2.7 倍和 2.8 倍，而臺北市五種方法的結果差距不大，其原因在於臺北市的道路資料較多且市區淹水密度較低，因此當道路碰到淹水時，有較多替代道路可以規劃，造成五種方法長度比率差異較小。

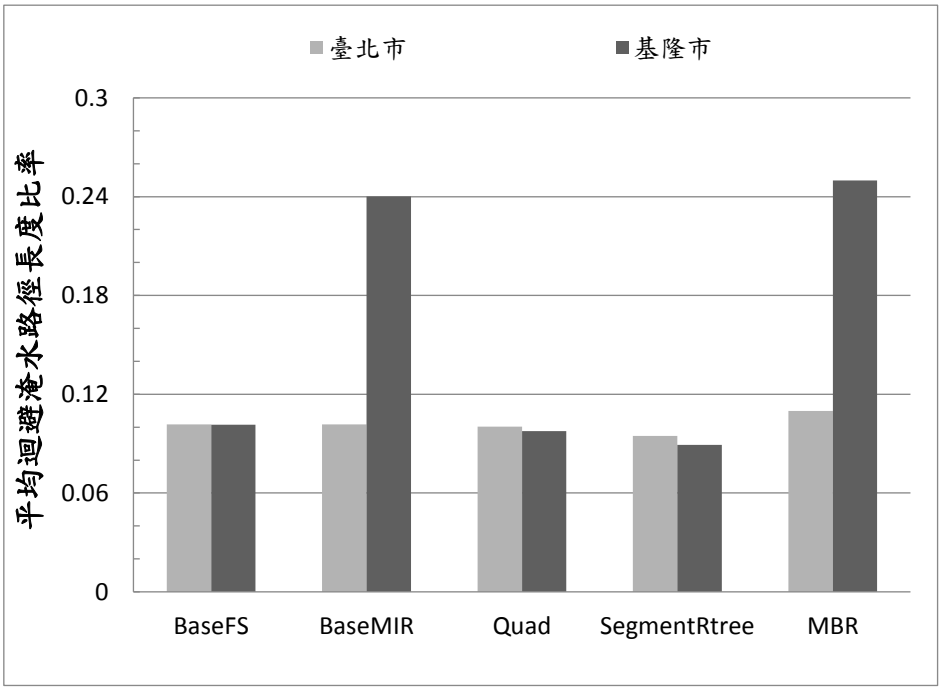


圖 5-8　平均迴避淹水長度比率

為了進一步解釋為何此五種方法會得到不同的長度比率，以下依照不同方法列出兩個資料集淹水區塊的覆蓋率，如下圖 5-9。注意到，表 5-4 和表 5-5 我們分別利用括號標註每個方法的排名，數字越小則名次越前。我們可觀察到，在基隆市淹水覆蓋率和長度比率的名次是一致的，而臺北市的 **BaseFS** 方法和 **BaseMIR** 方法的淹水覆蓋率相同，因此與淹水長度比率較無明顯的正比關係。仔細分析這些方法，**MBR** 方法的淹水覆蓋率最高，因為是將淹水區間的 MBR 直接輸出為淹水區塊，等於每一塊都超出淹水區間的面積很多，因此分割較不精準，所以較多道路被判定為不可行走，規劃的路徑長度也較長，反之 SegmentRtree 則最佳。至於我們所提出的 3 個方法，則以 **Quad** 方法的覆蓋率最小，路徑長度也最短。

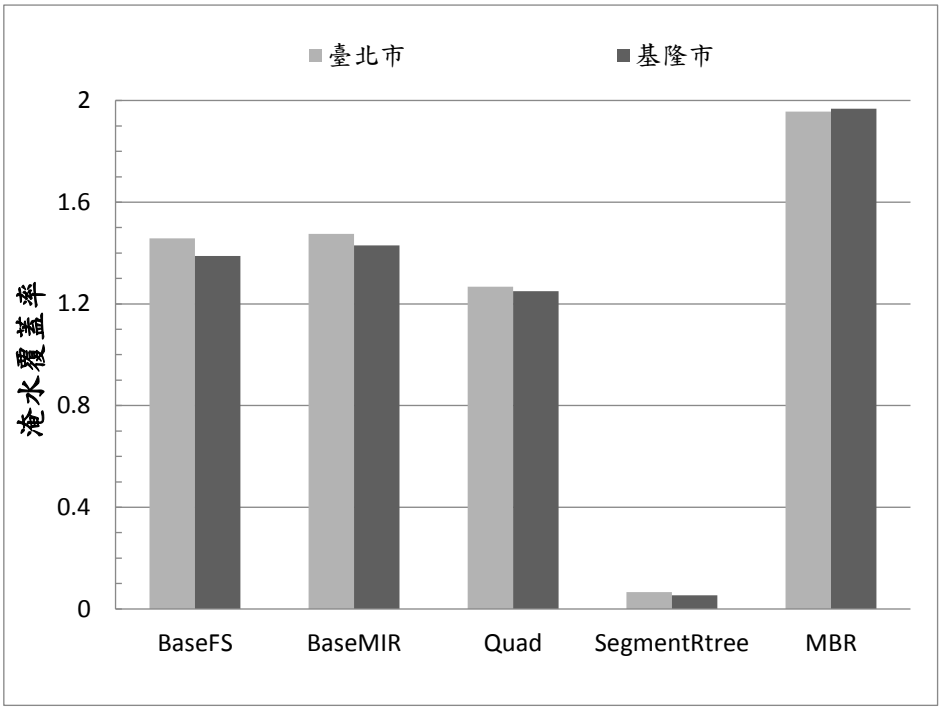


圖 5-9　淹水覆蓋率

表 5-4　基隆市平均迴避淹水長度比率和淹水覆蓋率

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
平均迴避淹水長度比率	0.101 (3)	0.24 (4)	0.097 (2)	0.089 (1)	0.249 (5)
淹水覆蓋率	1.388 (3)	1.429 (4)	1.25 (2)	0.054 (1)	1.967 (5)

表 5-5　臺北市平均迴避淹水長度比率和淹水覆蓋率

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
平均迴避淹水長度比率	1.457 (3)	1.475 (4)	1.267 (2)	0.065 (1)	1.956 (5)
淹水覆蓋率	0.101 (3)	0.101 (3)	0.1 (2)	0.0941 (1)	0.109 (4)

綜合而言，在基隆市使用 **Quad** 方法分割，平均迴避淹水長度比率為 **MBR** 方法的 0.4 倍，可見所提出的分割方法的確改善所規劃的道路，在市區淹水密度較大且道路較少的基隆市更有明顯的影響。

5.5　Offline 效率比較實驗

本節效率實驗中，先分別比較五種方法在 **Offline** 階段中的分割淹水區間模組及建立 R-tree 模組的效率(參照圖 2-1 和圖 2-7 的架構圖)，最後再比較各個方法在空間計算上所花費的時間。由於我們發現兩個資料集在分割淹水區間模組及建立 R-tree 模組的趨勢很接近，因此本節我們主要會以臺北市來做實驗結果的討

論，最後再與基隆市對照。五種方法在臺北市的分割淹水區間模組及建立 R-tree 模組的實驗結果，如下圖 5-10 所示。

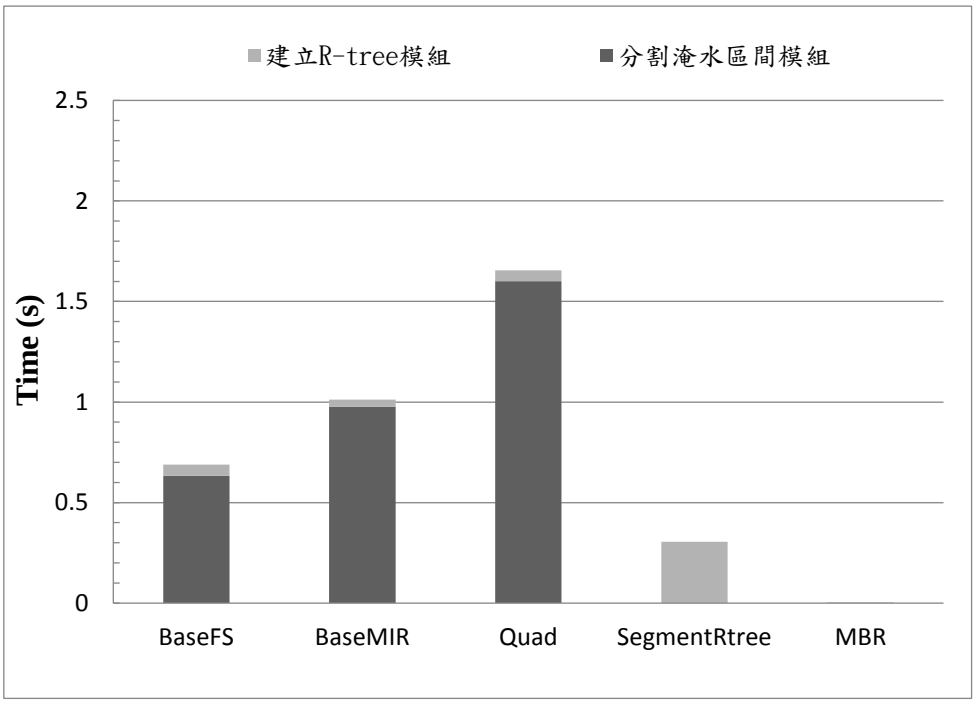


圖 5-10　臺北市 Offline 效率 (s)

注意到，**SegmentRtree** 及 **MBR** 兩個方法在分割淹水區間模組，實際上是以形成淹水區塊的方式，並非透過分割淹水區間的方式，因此所花費的時間較短。而在我們所提出的三個方法中，可以看出 **BaseFS** 方法因為其較直覺簡單，因此花費最少的時間在分割上。而分割時間會隨著淹水區間處理方式的複雜程度而增加，其中在臺北市使用 **Quad** 方法的分割淹水區間模組時間為 **BaseFS** 方法的 2.6 倍。

在建立 R-tree 模組中，透過比較表 5-6 第二列和表 A-1 的名次可以看出所花費的時間會與淹水區塊數量呈正比，且 **SegmentRtree** 方法是五個方法中輸出最多淹水區塊的方法，因此會花費最多時間在建立 R-tree 上，而 **SegmentRtree** 方法所花費的時間是分割方法中輸出最少淹水區塊的 **BaseMIR** 方法的 8.7 倍。

表 5-6　臺北市 Offline 效率 (s)

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
分割淹水區間模組	0.633	0.977	1.601	0.002	0.0005
建立 R-tree 模組	0.055 (4)	0.035 (2)	0.053 (3)	0.303 (5)	0.001 (1)
總時間	0.688	1.012	1.654	0.305	0.0015

基隆市在分割淹水區間模組和建立 R-tree 模組的效率如下圖 5-11，可以看出整體花費時間較臺北市來得多，但趨勢很接近。在分割淹水區間模組上，基隆市使用 **Quad** 方法所花費的時間為 **BaseFS** 方法的 2.3 倍，兩個方法之間的差異較臺北市的 2.6 倍來得小。在建立 R-tree 模組上，輸出最多淹水區塊的 **SegmentRtree** 方法所花費的時間是 **BaseMIR** 方法的 7.5 倍，兩個方法之間的差異也較臺北市的 8.7 倍來得小。

比較兩個資料集，可以發現各方法的效率表現是一致的，不過雖然兩個資料集的淹水區間總數量差不多，但基隆市的淹水區間總面積較大，所以需要花費較多的處理時間。但最費時的 **Quad** 方法也可以在 2.5 秒內完成，所以在效率上是可接受的。

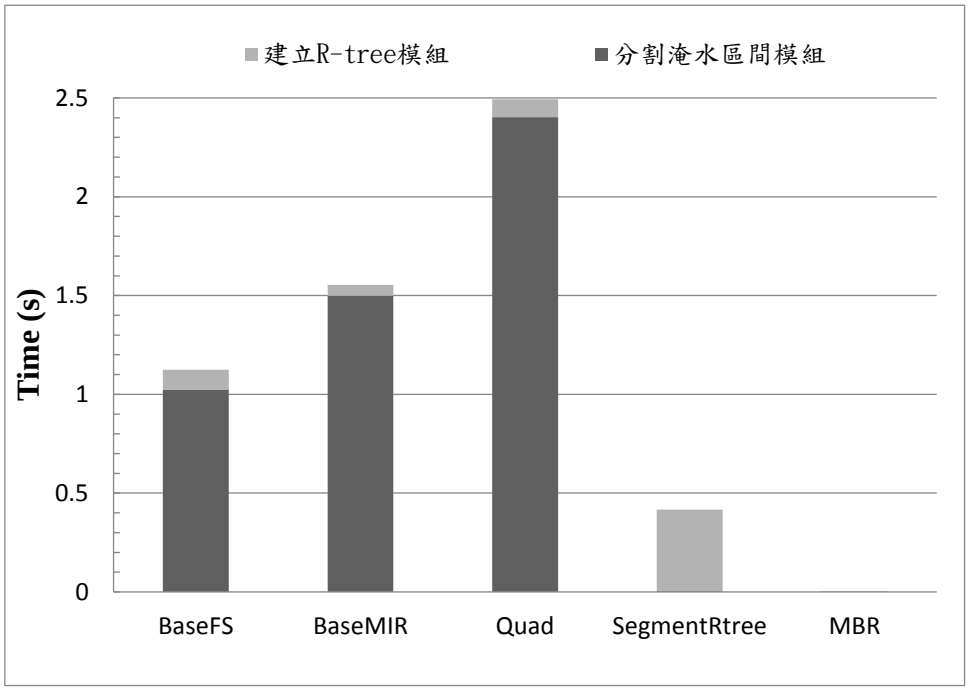


圖 5-11　基隆市 Offline 效率 (s)

表 5-7　基隆市 Offline 效率 (s)

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
分割淹水區間模組	1.024	1.498	2.403	0.003	0.0007
建立 R-tree 模組	0.1 (4)	0.055 (2)	0.089 (3)	0.412 (5)	0.001 (1)
總時間	1.124	1.553	2.492	0.415	0.0017

在前面第 3 章及第 4 章中，我們利用空間計算次數比較 3 種方法的效率，在接下來的實驗中，我們利用兩個資料集中的所有淹水區間比較各方法空間計算上的效率。由於 **SegmentRtree** 及 **MBR** 兩種方法並非使用分割淹水區間的方式，因此不包含任何的空間計算，以下針對 **BaseFS**、**BaseMIR** 及 **Quad** 三種方法，

來比較空間計算的次數及花費的時間，如下圖 5-12 及圖 5-13 所示。可以看出 **BaseMIR** 方法雖然在先前的範例上的空間計算的次數較 **BaseFS** 方法來得優異，但是整體而言，空間計算的次數及花費的時間皆比 **BaseFS** 方法來得高。而 **Quad** 方法所花的空間計算時間為三種方法中最低的。

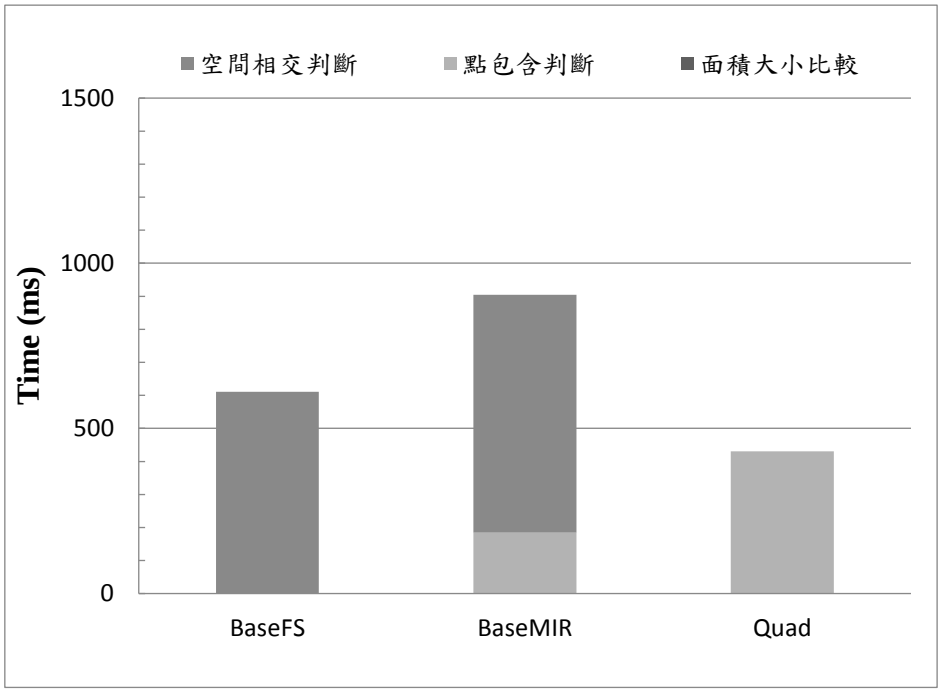


圖 5-12　臺北市空間計算效率 (ms)

表 5-8　臺北市空間計算效率{耗時(ms) / 次數}

演算法名稱	BaseFS	BaseMIR	Quad
面積大小比較	0.1 / 411	0.2 / 1979	1.5 / 16014
點包含判斷	-	184 / 411	429 / 2300
空間相交判斷	611 / 10550	720 / 13742	-
加總	611.1 / 10961	904.2 / 16132	430.5 / 18314

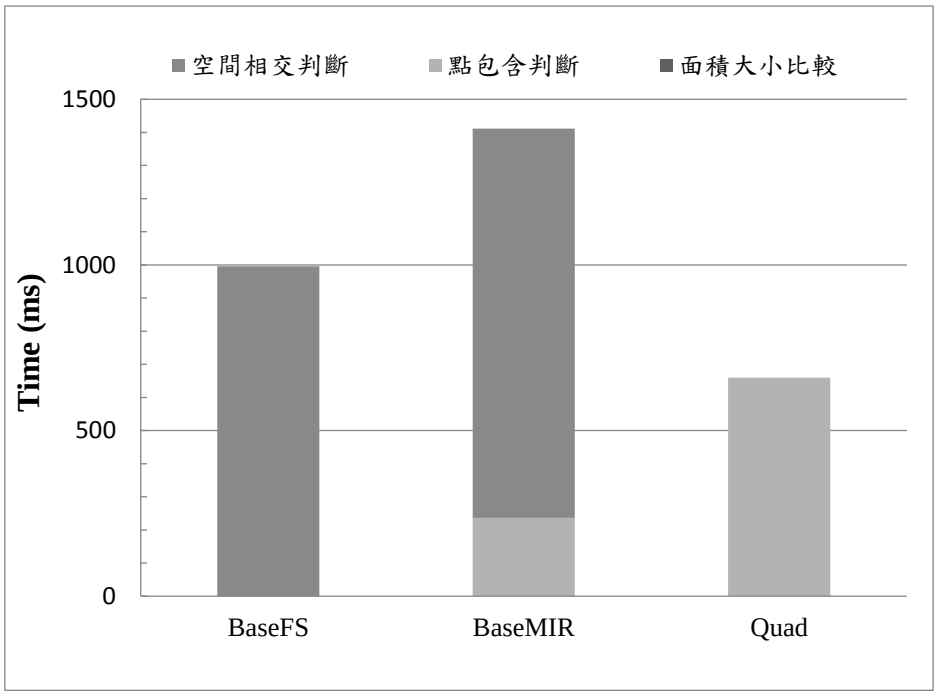


圖 5-13　基隆市空間計算效率 (ms)

表 5-9　基隆市空間計算效率{耗時(ms) / 次數}

演算法名稱	BaseFS	BaseMIR	Quad
面積大小比較	0.1 / 436	0.2 / 2136	1.9 / 25318
點包含判斷	-	236 / 436	658 / 3795
空間相交判斷	996 / 18891	1175 / 24149	-
加總	996.1 / 19327	1411.2 / 26721	659.9 / 29113

分析整個 **Offline** 階段所需的時間，以臺北市為例。**BaseFS** 方法在分割淹水區間模組花費的時間為 0.633 秒，空間計算時間共 0.611 秒，約 97%的時間都花費在空間計算上。**BaseMIR** 方法在分割淹水區間模組花費的時間為 0.977 秒，空間計算時間共 0.904 秒，約 93%的時間都花費在空間計算上。**Quad** 方法在分割淹水區間模組花費的時間為 1.601 秒，空間計算時間共 0.43 秒，只有 27%的時間花費在空間計算上。

實驗結果顯示，雖然 **Quad** 方法花費較多時間在分割淹水區間的處理上，但是空間計算所花費的時間為三種方法最低。顯示主要花費的時間在於遞迴處理四個象限方面。

5.6　Online 效率比較實驗

我們在 **Online** 階段要分別比較五種方法在兩個資料集中的移除淹水道路模組及規劃路徑模組的效率(參照圖 2-1 的架構圖)。由於我們發現兩個資料集在移

除淹水道路模組及規劃路徑模組的趨勢很接近，因此本節我們主要會以臺北市來做實驗結果的討論，最後再與基隆市對照。五種方法在臺北市的移除淹水道路組及規劃路徑模組的實驗結果，如下圖 5-14 所示。

臺北市在移除淹水道路模組中，首先比較兩個對照方法，發現分割出淹水區塊數量最多的 **SegmentRtree** 方法花費的時間為分割出淹水區塊數量最少的 **MBR** 方法的 29.8 倍。接著我們提出的 3 種分割方法中，發現 **Quad** 方法與 **BaseFS** 方法的效率接近，而 **Quad** 方法所花費的時間為 **BaseMIR** 方法的 1.4 倍。在規劃路徑模組中，可以看到 **SegmentRtree** 方法在規劃路徑上會花費最多時間，但也只是最快的 **MBR** 方法的 1.1 倍。

首先分析移除淹水道路模組所需的時間，如圖 2-8 的 L08-L12 所示，要讓每條道路去走訪 R-tree 來查詢判斷是否與淹水區塊相交，並找出所有未淹水的道路，因此花費的時間會隨著輸出淹水區塊的數量和 R-tree 的樹高而改變。我們將各方法在 **Offline** 階段分割出的淹水區塊數量及樹高列出，如表 A-1 及表 A-2。同時透過淹水區塊數量和樹高，可以看出 **MBR** 方法因為輸出的淹水區塊最少，且樹高低於其他方法，因此在移除淹水道路模組上會花費最少的時間。反之，**SegmentRtree** 方法則因為淹水區塊最多，因此在移除淹水道路模組上會花費最多的時間。

接著分析規劃路徑模組，其時間與未淹水道路數量有關(參考表 A-3)，但因為該數量的差距不大，所以 5 種方法所花費的時間相近。

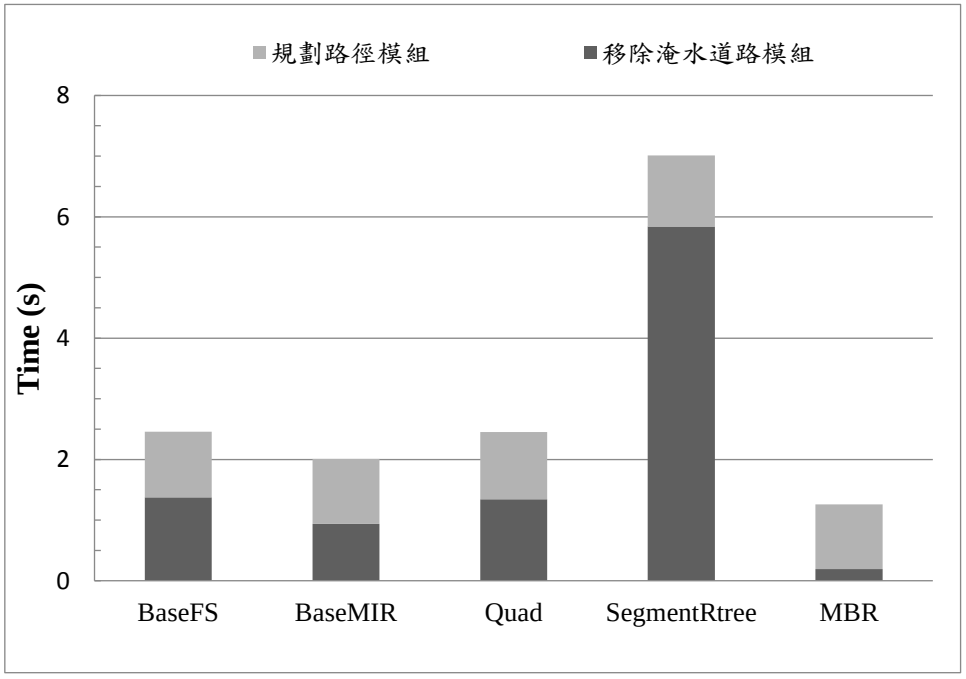


圖 5-14　臺北市 Online 效率 (s)

表 5-10　臺北市 Online 效率 (s)

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
移除淹水道路模組	1.373 (1)	0.934 (2)	1.341 (3)	5.838 (5)	0.192 (1)
規劃路徑模組	1.08 (1)	1.076 (2)	1.112 (3)	1.172 (5)	1.067 (1)
總時間	2.453	2.01	2.453	7.01	1.259

基隆市在移除淹水道路模組和規劃路徑模組的效率如下圖 5-15，可以看出整體花費時間較臺北市來得少，但趨勢很接近。在移除淹水區間模組上，首先比較兩個對照方法，發現分割出淹水區塊數量最多的 **SegmentRtree** 方法花費的時間為分割出淹水區塊數量最少的 **MBR** 方法的 22.3 倍，差異比臺北市的 29.8 倍來得小。接著比較我們提出的 3 種分割方法，發現 **Quad** 方法與 **BaseFS** 方法的效率接近，而 **Quad** 方法所花費的時間為 **BaseMIR** 方法的 1.5 倍，差異比臺北市的 1.4 倍來得大。在規劃路徑模組中，未淹水道路的數量差距也不大，所以 5 種方法所花費的時間相近，可以看到 **SegmentRtree** 方法在規劃路徑上會花費最多時間，但也只是最快的 **MBR** 方法的 1.2 倍，差異較臺北市的 1.1 倍來得大。

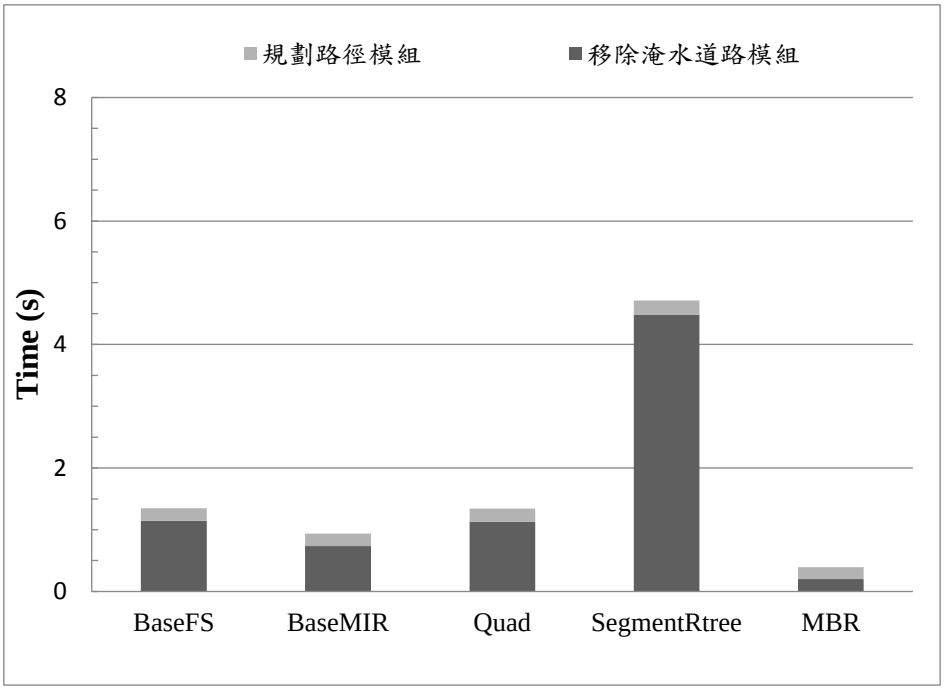


圖 5-15　基隆市 Online 效率 (s)

表 5-11　基隆市 Online 效率 (s)

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
移除淹水道路模組	1.145	0.735	1.132	4.484	0.201 (1)
規劃路徑模組	0.2 ()	0.2	0.21	0.229	0.189 (1)
總時間	1.345	0.935	1.342	4.713	0.39

整體而言，在 **Online** 階段移除淹水道路模組所花費的時間，會隨著淹水區塊數量和未淹水道路數量而增加。比較兩個資料集在 **Online** 階段的效率，以 **MBR** 方法花費最少時間，其花費的時間都在 1.3 秒以內。而 **SegmentRtree** 方法會有道路被淹水區塊所包覆但卻判斷成可以走的情況，因為在兩個資料集中的未淹水道路數量都最多，如圖 5-1 所提到的問題，其花費的時間都需要 4.7 秒以上，因此在 **Online** 階段所花費的時間會是五種方法中最高的，在實務上不可行。三種分割方法所花費的時間都在 2.5 秒以內，且以 **BaseMIR** 方法的效率最好，而 **BaseFS** 方法和 **Quad** 方法的效率差異不大。

5.7 改變門檻值 MaxRatio 之實驗

在 5.3 節，我們利用淹水覆蓋率選擇 MaxRatio 的值為 0.95。在本節中，要針對改變 **Quad** 方法的門檻值 MaxRatio 值來觀察是否對規劃出的迴避淹水長度比率和效率有影響，以驗證該節的作法是否合理。參數值 CellEdge 預設為 50 公尺，order 值預設為 64，起迄點資料使用同 5.4 節的 30 組。

首先，透過改變 MaxRatio 值為 0.55、0.75 及 0.95，來觀察對平均迴避淹水長度比率之影響，如下圖 5-16。可以發現 **Quad** 方法在基隆市資料集中，當 MaxRatio 設為 0.55 及 0.75 之間其長度比例有明顯差距。這是由於基隆市區的淹水密度較大且道路較少，因此在規劃迴避淹水時，當淹水範圍的面積越大，所能選擇的替代道路就越遠。以 MaxRatio 值設為 0.55 來說，只要淹水區間的面積占有 MBR 的 55%就不會再繼續分割，輸出的淹水區塊面積就會很大且提高淹水覆蓋率，如表 A-4 所示，淹水覆蓋率會與 MaxRatio 呈反比降低，導致長度比率亦隨之反比降低。當 **Quad** 方法將 MaxRatio 設為 0.55 時明顯看出長度比率最大，在基隆市規劃出的平均迴避淹水長度比率是 MaxRatio 設為 0.75 時的 2.5 倍。而 MaxRatio 設為 0.95 時則較 0.75 的長度比率來得較低一些。

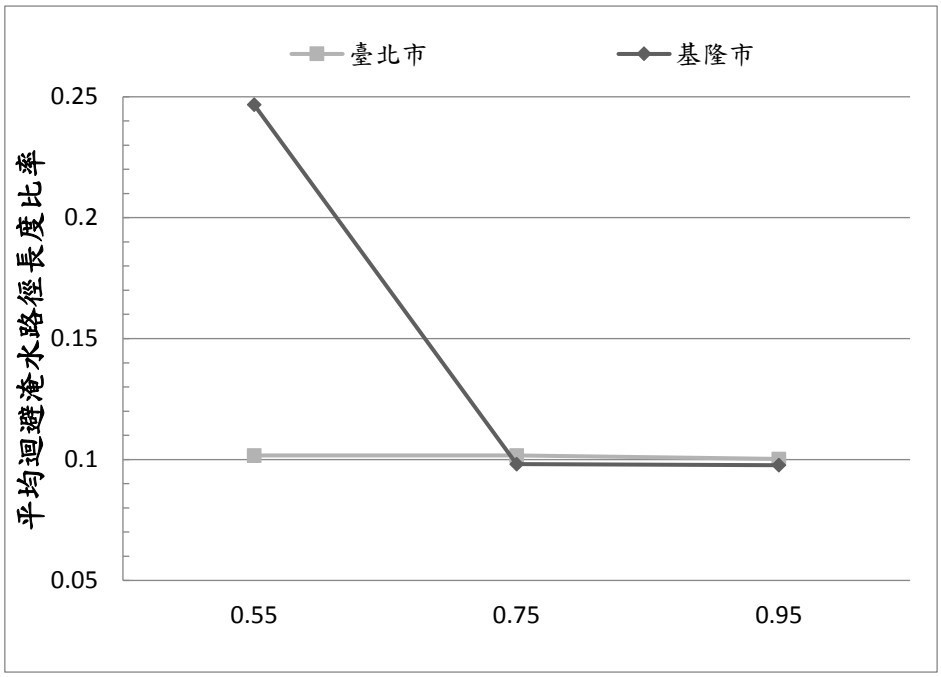


圖 5-16　改變 MaxRatio 值對平均迴避淹水長度比率之影響

表 5-12　改變方法 MaxRatio 值對平均迴避淹水長度比率之影響 (s)

MaxRatio 值	**0.55**	**0.75**	**0.95**
臺北市	0.101 (2)	0.101 (2)	0.1 (1)
基隆市	0.246 (3)	0.098 (2)	0.097 (1)

接下來，要觀察 **Quad** 方法在兩個資料集中改變門檻值 MaxRatio 對 **Offline** 階段的效率影響，如下圖 5-17 所示。我們把分割出的淹水區塊數量列出，發現當 MaxRatio 越高會將淹水區間分割出越多淹水區塊，且 **Offline** 階段的總時間會與分割出的淹水區塊呈正比，如下圖 5-18 所示。

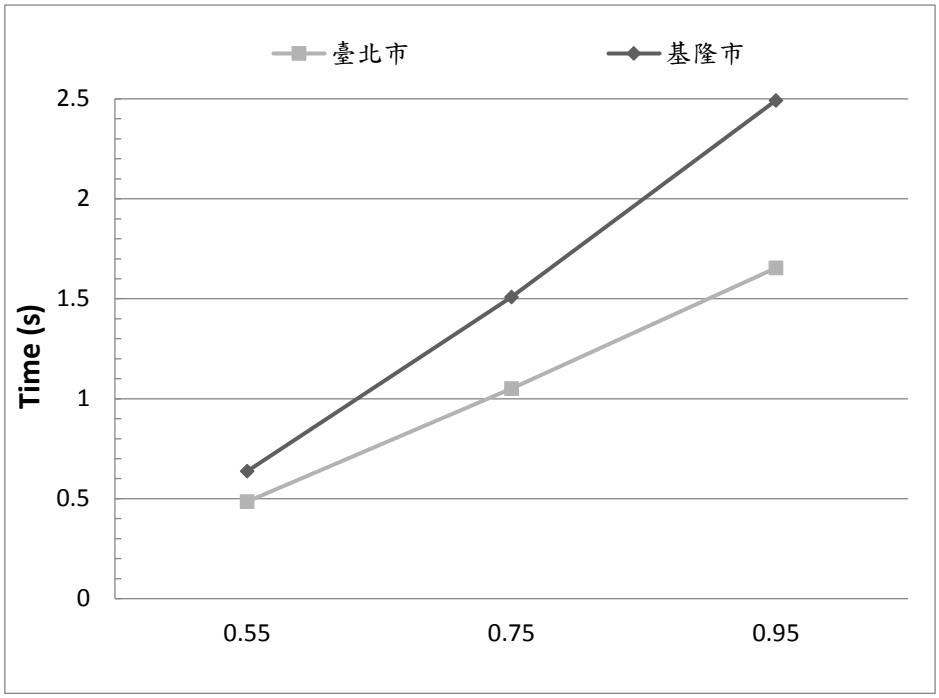


圖 5-17　改變 MaxRatio 值對 Offline 效率之影響

表 5-13　改變 MaxRatio 值對臺北市 Offline 效率之影響 (s)

MaxRatio 值	**0.55**	**0.75**	**0.95**
分割淹水區間模組	0.48	1.024	1.601
建立 R-tree 模組	0.005	0.026	0.053
總時間	0.485 (1)	1.05 (2)	1.654 (3)

表 5-14　改變 MaxRatio 值對基隆市 Offline 效率之影響 (s)

MaxRatio 值	**0.55**	**0.75**	**0.95**
分割淹水區間模組	0.631	1.468	2.403
建立 R-tree 模組	0.006	0.041	0.089
總時間	0.637 (1)	1.509 (2)	2.492 (3)

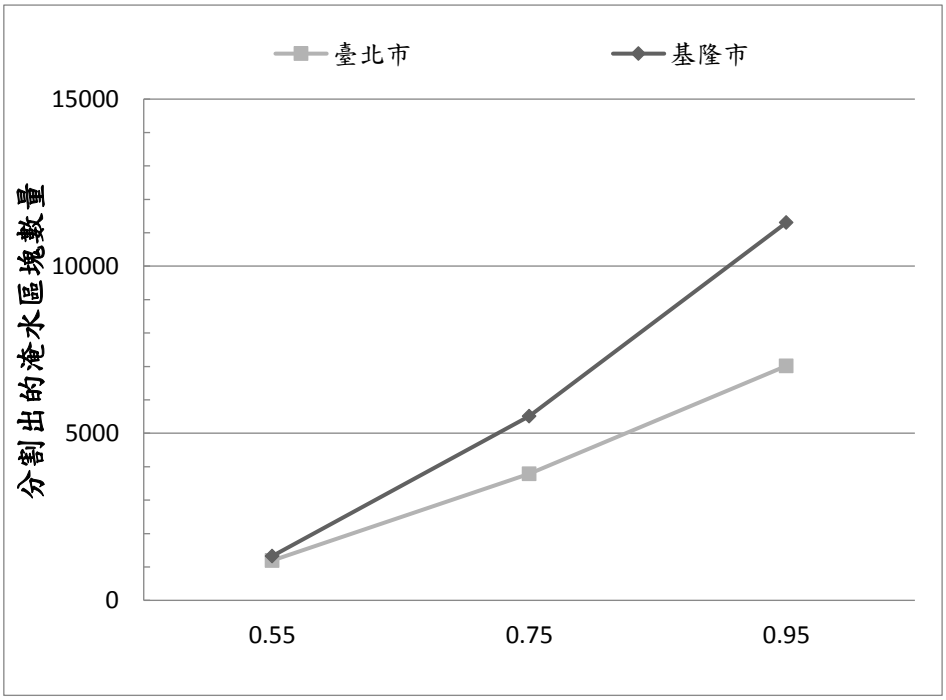


圖 5-18　改變 MaxRatio 值對分割出的淹水區塊數量之影響

表 5-15　改變 MaxRatio 值對分割出的淹水區塊數量之影響 (s)

MaxRatio 值	**0.55**	**0.75**	**0.95**
臺北市	1192	3786	7016
基隆市	1325	5515	11303

最後，我們要觀察 Quad 方法在兩個資料集中改變門檻值 MaxRatio 對 **Online** 階段的效率影響，如下圖 5-19。可以看出 **Online** 階段的總時間會與 MaxRatio 值呈正比增加，因此我們把樹高和未淹水道路數量一併列出，如表 A-5 及表 A-6。發現當 MaxRatio 設為 0.55 時，樹高為 2，是因為分割出的淹水區塊較少，因此在移除淹水道路模組走訪 R-tree 時會花較少時間，而規劃路徑模組所花費的時間也隨著未淹水道路的數量而增加。

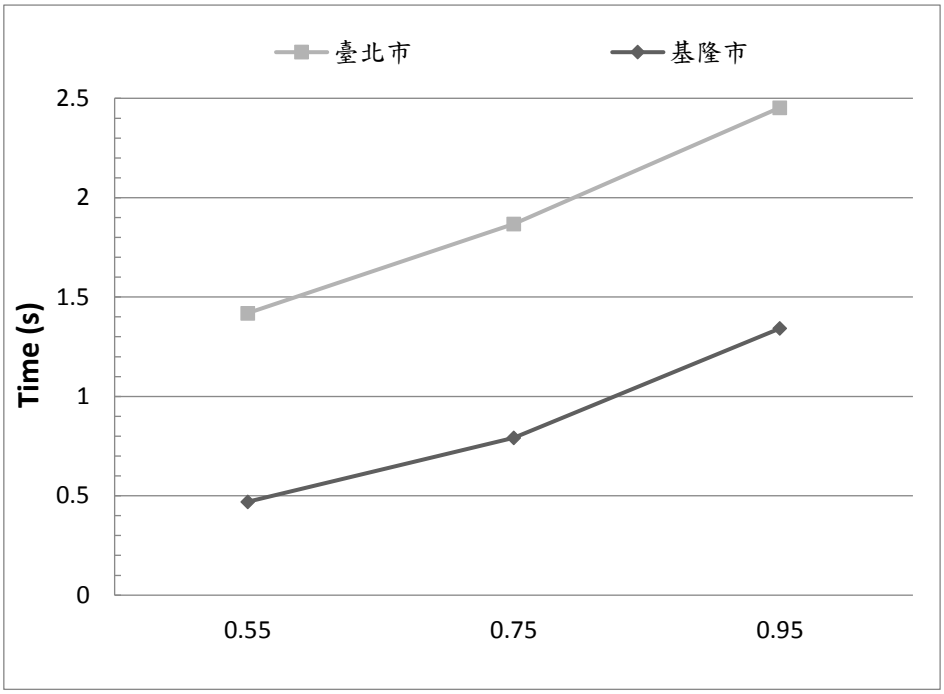


圖 5-19　改變 MaxRatio 值對臺北市 Online 效率之影響

表 5-16　改變 MaxRatio 值對臺北市 Online 效率之影響 (s)

MaxRatio 值	**0.55**	**0.75**	**0.95**
移除淹水道路模組	0.322	0.769	1.341
規劃路徑模組	1.097	1.098	1.112
總時間	1.419 (1)	1.867 (2)	2.453 (3)

表 5-17　改變 MaxRatio 值對基隆市 Online 效率之影響 (s)

MaxRatio 值	**0.55**	**0.75**	**0.95**
移除淹水道路模組	0.27	0.588	1.132
規劃路徑模組	0.2	0.204	0.21
總時間	0.47 (1)	0.792 (2)	1.342 (3)

實驗結果顯示，**Quad** 方法在不同資料集中，規劃出的平均迴避淹水路徑比率會與 MaxRatio 值呈反比，尤其在基隆市更明顯，但各模組效率皆會與 MaxRatio 值呈線性正比。當 MaxRatio 值設為 0.95 時，在臺北市和基隆市在 **Offline** 階段花費的總時間是 MaxRatio 值設為 0.55 的 3.4 倍及 3.9 倍，而臺北市和基隆市在 **Online** 階段花費的總時間是 MaxRatio 值設為 0.55 的 1.7 倍及 2.9 倍。結論是當 MaxRatio 值設為 0.95 時，雖然在 **Offline** 和 **Online** 皆要花較多時間，但是規劃出的迴避淹水路徑會最短。

5.8 改變 order 之實驗

在前面的實驗中，我們將 R-tree 的 order 值都設為 64。在本節實驗中，我們要在不同資料集中改變 order 值，觀察對五個方法在建立 R-tree 模組及移除淹水道路模組效率上的影響。由於我們發現兩個資料集建立 R-tree 模組及移除淹水道路模組的趨勢很接近，因此本節我們主要會以臺北市來做實驗結果的討論，並將基隆市列在本節末來做結論的比較。

首先，我們討論改變 order 對建立 R-tree 模組效率的影響，如下圖 5-20。可以看出 **SegmentRtree** 方法建立 R-tree 的時間會與 order 值呈非線性正比增加，當 order 設為 128 時的時間快速增加，當 **SegmentRtree** 方法將 order 設為 128 時建立 R-tree 的時間為 order 設為 32 時的 1.8 倍。而 **MBR** 方法因為淹水區塊數量最少，因此僅須極少時間建立 R-tree。比較我們提出的三種分割方法，可以看出 **BaseFS**、**BaseMIR** 及 **Quad** 方法的建立時間與 order 值呈線性正比增加，當 order 設為 128 時所花費的時間分別為 order 設為 32 時的 1.5 倍、1.6 倍及 1.5 倍。

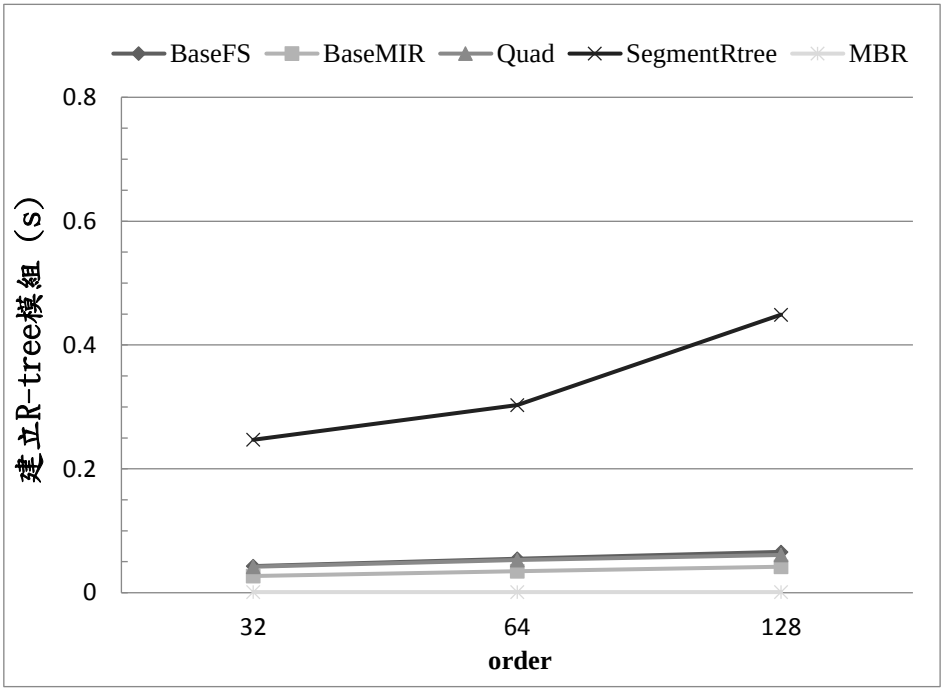


圖 5-20　臺北市改變 order 對建立 R-tree 模組效率之影響

表 5-18　臺北市改變 order 對建立 R-tree 模組效率之影響 (s)

order \ 演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
32	0.043	0.027	0.042	0.247	0.001
64	0.055	0.035	0.053	0.303	0.001
128	0.066	0.042	0.061	0.449	0.001

接下來，我們要討論臺北市改變 order 對移除淹水道路模組效率的影響，如下圖 5-21。由於改變 order 值會影響到 R-tree 的樹高，進而影響移除淹水道路模組所花費的時間，因此我們先將樹高列出，如表 A-7 所示。

首先觀察 **SegmentRtree** 方法當 order 值從 32 改為 64 時所花費的時間快速減少，這是因為其樹高從 4 降為 3。至於其他方法因為樹高維持在 3，所以時間雖然也有減少，但改變較少。當 order 從 64 進一步改成 128，則因為樹高再度降低，各個方法的時間也進一步微幅下降。以 **Quad** 方法為例，當 order 設為 32 時所花費的時間為 order 設為 64 時的 1.1 倍，當 order 設為 64 時所花費的時間也為 order 設為 128 時的 1.1 倍。比較我們提出的三種分割方法，可以看出 **BaseFS**、**BaseMIR** 及 **Quad** 方法的查詢時間與 order 值呈線性反比減少，當 order 設為 32 時所花費的時間分別為 order 設為 128 時的 1.3 倍、1.7 倍及 1.4 倍。

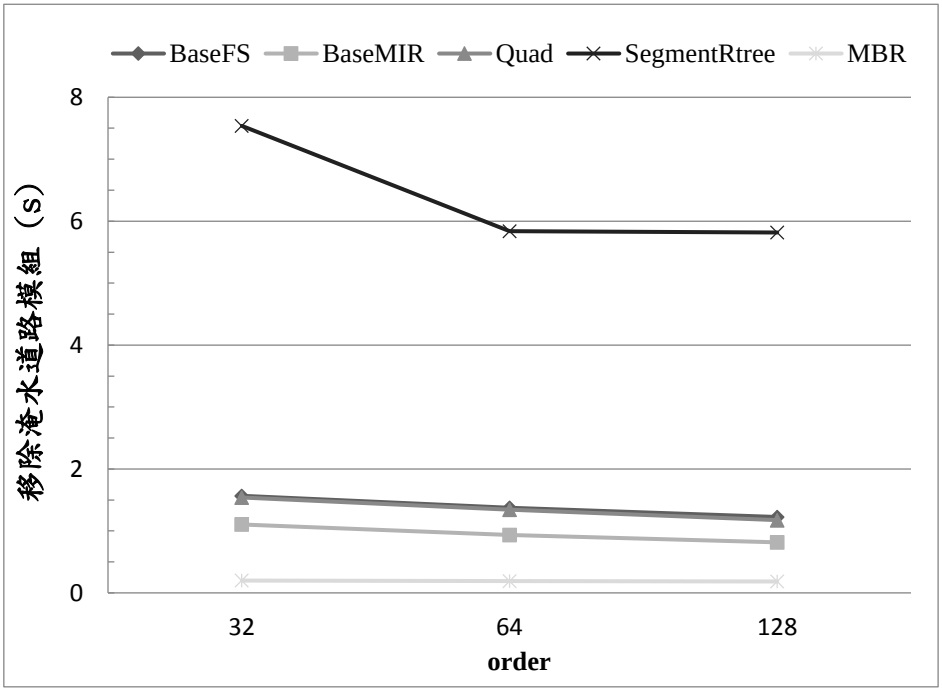


圖 5-21　臺北市改變 order 對移除淹水道路模組效率之影響

表 5-19　臺北市改變 order 對移除淹水道路模組效率之影響 (s)

order \ 演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
32	1.567	1.104	1.54	7.538	0.199
64	1.373	0.934	1.341	5.838	0.192
128	1.225	0.816	1.175	5.817	0.184

基隆市改變 order 對建立 R-tree 模組效率和移除淹水道路模組效率的影響，如下圖 5-22 和圖 5-23。可以看出建立 R-tree 模組效率的整體趨勢和臺北市相似，因此我們將基隆市的改變 order 後的樹高列出，如表 A-8 所示，一樣可以看到

SegmentRtree 方法因為在 order 為 32 時的樹高為 4，所以花費極多時間。比較我們提出的三種分割方法，可以看出 **BaseFS**、**BaseMIR** 及 **Quad** 方法的建立 R-tree 時間與 order 值呈線性正比增加，當 order 設為 128 時所花費的時間分別為 order 設為 32 時的 1.8 倍、1.4 倍及 1.7 倍。而查詢時間與 order 值呈線性反比減少，當 order 設為 32 時所花費的時間分別為 order 設為 128 時的 1.3 倍、1.1 倍及 1.3 倍。

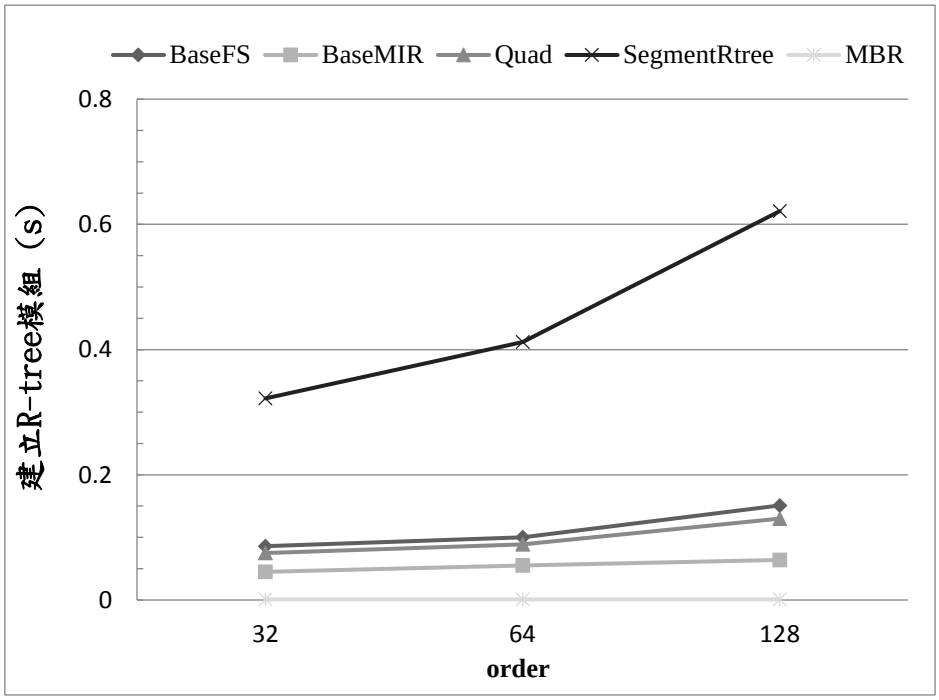


圖 5-22　基隆市改變 order 對建立 R-tree 模組效率之影響

表 5-20　基隆市改變 order 對建立 R-tree 模組效率之影響 (s)

order \ 演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
32	0.086	0.045	0.075	0.322	0.001
64	0.1	0.055	0.089	0.412	0.001
128	0.151	0.064	0.13	0.621	0.001

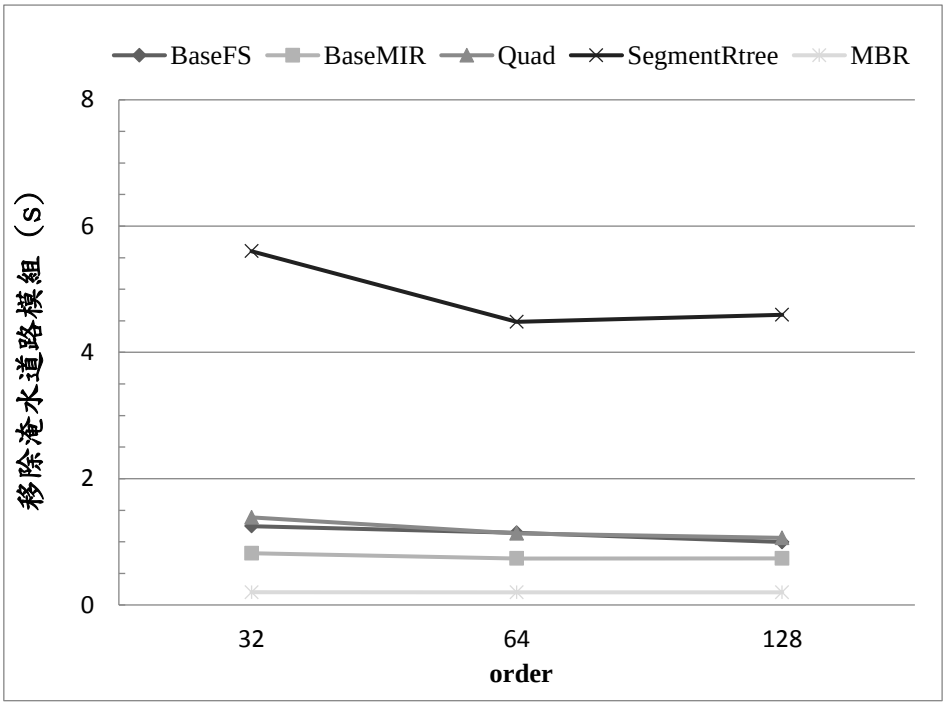


圖 5-23　基隆市改變 order 對移除淹水道路模組效率之影響

表 5-21　基隆市改變 order 對移除淹水道路模組效率之影響 (s)

order \ 演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
32	1.247	0.819	1.387	5.602	0.202
64	1.145	0.735	1.132	4.484	0.201
128	0.997	0.738	1.063	4.595	0.2

實驗結果顯示，**Offline** 階段時建立 R-tree 所花費的時間會與 order 值呈正比。對 **Online** 階段的移除淹水道路所花費的時間會與 order 值呈反比，當淹水區塊數量極大時，改變 order 時的影響會越明顯。若同時比較兩個資料集，可以看出 5 個方法中，除了 **SegmentRtree** 方法對改變 order 的影響最為明顯，其餘方法所分割出的淹水區塊較少，在改變 order 情況下的影響較不明顯，其中對 **MBR** 方法的影響最不明顯，而三種分割方法則會有些微影響。

第6章 結論與未來方向

在本論文中，我們提出分割淹水區間的 3 個方法以及 2 個對照方法。首先，提出了 **BaseFS** 方法，是一個比較直覺的方法，透過固定的淹水區塊邊長 CellEdge 來控制分割出淹水區間的大小及數量。接下來，為了降低空間計算的次數，我們提出了 **BaseMIR** 方法，透過取出一塊最小交點矩形來降低不必要的空間計算。最後，我們為了降低淹水覆蓋率，並且減少淹水區塊的數量，我們提出了 **Quad** 方法，希望透過門檻值 MaxRatio 來控制輸出的淹水區塊精準度，進而降低淹水覆蓋率，來達到我們希望的成果。另外兩種對照方法，**MBR** 方法是直接求取每個淹水區間的 MBR 不進行分割，直接傳送給建立 R-tree 模組。而 **SegmentRtree** 方法是將每個淹水區間的點序列，將鄰近的 2 個點連成一個線段來取 MBR 以建立 R-tree。

透過路徑長度實驗，在三種分割方法中我們得到的結果是，**Quad** 方法規劃迴避淹水的路徑最短，關鍵在於分割淹水區間時淹水覆蓋率最低，降低了誤判的機率。在 **Offline** 階段，雖然 **Quad** 方法花費較多時間在分割淹水區間的處理上，但是空間計算所花費的時間是三種分割方法中最低的。而在 **Online** 階段，**Quad** 方法在移除淹水及規劃路徑上的效率也較 **BaseFS** 方法佳。雖然 **BaseFS**、**BaseMIR** 和 **MBR** 方法的效率都有優於 **Quad** 方法之處，但是整體來說，這些方法分割後的淹水覆蓋率都較大，較容易造成誤判，造成平均迴避淹水的路徑較長，因此較不推薦使用。而 **SegmentRtree** 方法雖然所規劃的路徑長度最短，但是在規劃路徑時會包含很多不能走的道路，當規劃的區域擴大時，會對效率造成影響，在臺北市和基隆市的 **Online** 時間分別為 7.01 秒和 4.713 秒，在實務上不可行。因此整體來講都較推薦使用 **Quad** 方法來分割淹水區間。

本論文未來改進的方向，希望用更多的資料集來測試所提出的方法。另一方面，以 **Quad** 方法來說，我們發現在臺北市資料集上將 MaxRatio 值設為 0.55 可以有很好的效率及迴避淹水的路長結果，但是在基隆市資料集上的表現就不理想，必須將 MaxRatio 值改為 0.95 才能得到好的表現。因此 MaxRatio 和 R-tree 的 order 等參數值的設定，對其效率和輸出路徑都會有所影響。我們也希望未來以更廣泛的實驗或提出更好的評估方法來決定這些參數值的設定。

附錄 A 詳細實驗數據

我們將輔助說明實驗結果的數據列於本附錄中，並利用括號標註每個方法的排名，將數值越小的方法排名越前面，以利在實驗說明時可用來對照和分析之用途。

表 A-1　各方法輸出的淹水區塊數量

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
臺北市	7484 (4)	4904 (2)	7016 (3)	34646 (5)	411 (1)
基隆市	12716 (4)	7277 (2)	11303 (3)	45561 (5)	436 (1)

表 A-2　各方法建立的索引樹高

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
臺北市	3	3	3	3	2
基隆市					

表 A-3　未淹水道路數量

演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
臺北市	41913 (3)	41892 (2)	41981 (4)	42314 (5)	41704 (1)
基隆市	5674 (3)	5661 (2)	5795 (4)	6040 (5)	5527 (1)

表 A-4　改變 MaxRatio 值對淹水覆蓋率之影響

MaxRatio 值	0.55	0.75	0.95
臺北市	1.582 (3)	1.334 (2)	1.267 (1)
基隆市	1.575 (3)	1.315 (2)	1.25 (1)

表 A-5　改變 MaxRatio 值對樹高之影響

MaxRatio 值	0.55	0.75	0.95
臺北市	2	3	3
基隆市			

表 A-6　改變 MaxRatio 值對未淹水道路數量之影響

MaxRatio 值	0.55	0.75	0.95
臺北市	41795	41934	41981
基隆市	5636	5748	5795

表 A-7　臺北市改變 order 對樹高之影響

order \ 演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
32	3	3	3	4	2
64	3	3	3	3	2
128	2	2	2	3	2

表 A-8　基隆市改變 order 對樹高之影響

order \ 演算法名稱	BaseFS	BaseMIR	Quad	SegmentRtree	MBR
32	3	3	3	4	2
64	3	3	3	3	2
128	3	2	3	3	2

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, Pages 47-57, 1984.

[BKSS90] Norbert Beckmann, Hans-Peter kriegel, Ralf Schneider, Bernhard Seeger, “The R*-tree: an efficient and robust access method for points and rectangles”, Proceedings of the ACM SIGMOD international conference on Management of data, Pages 322-331, 1990.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger, “Multi-Step processing of spatial joins”, Proceedings of the ACM SIGMOD international conference on Management of data, Pages 197-208, 1994.

[BD98] Beman Dawes, David Abrahams, “boost c++ libraries”, http://www.boost.org/, 1998.

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

[EG07] Manfred Ernst, G¨unther Greiner, “Early Split Clipping for Bounding Volume Hierarchies”, Proceedings of the IEEE/EG Symposium on Interactive Ray Tracing, 2007.

[EK11] E. Kreyszig, “Advanced Engineering Mathematics”, 10th ed., John Wiley & Sons, Inc., New York, Page 423, 2011.

[FS75] H. Freeman, R. Shapira, “Determining the minimum-area encasing rectangle for an arbitrary closed curve”, Communications of the ACM, Vol. 18 Issue 7, Pages 409-413, 1975.

[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.

[MVJ14] Marek Vinkler, Vlastimil Havran, Jiří Bittner, “Bounding volume hierarchies versus kd-trees on contemporary many-core architectures”, Proceedings of the 30th Spring Conference on Computer Graphics, Pages 29-36, 2014.

[WG92] Jane Wilhelms, Allen Van Gelder, “Octrees for faster isosurface generation”, ACM Transactions on Graphics, Volume 11 Issue 3, Pages 201-227, 1992.

[WIKKP06] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, Steven G. Parker,

"Ray tracing animated scenes using coherent grid traversal", ACM Transaction on Graphics, Volume 25 Issue 3, Pages 485-493, 2006.

[ZHWG08] Kun Zhou, Qiming Hou, Rui Wang, Baining Guo, "Real-time KD-tree construction on graphics hardware", ACM Transactions on Graphics, Volume 27 Issue 5, Article No. 126, 2008.

[ZS98] Geraldo Zimbrao, Jano Moreira de Souza, "A raster approximation for processing of spatial joins", Proceedings of the 24rd International Conference on Very Large Data Bases, Pages 558-569, 1998.

[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.

[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 14] 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, 2014.

[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.

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