第六章 防災存活路網程式

6.1 路網結構

路網結構轉化為資料以運用於電腦程式之中的方法，有下列方式：

6.1.1 資料結構

1. 單連列（Singly-linked list, SLL）

單連列的兩個特性：1.每個單連列節點都包含一個元素、以及連結到下一個節點的節線（若下一個節點不存在，則該節線為空節線）；2.單連列有一個可以連結到第一個節點的表頭（若單連列為空集合，則該連結節線為空節線）。其資料結構可表示如下。

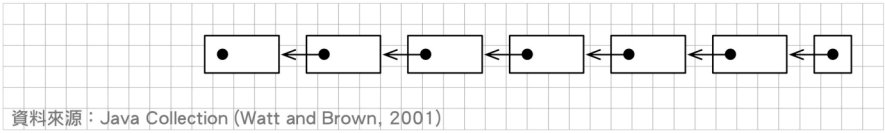

圖6.1 單連列

2. 雙連列（Doubly-linked list, DLL）

有別於單連列的特性：1.每一個雙連列包含一個元素、和連結到上一個節點的節線（若上一個節點不存在，則該節線為空節線）、以及連結到下一個節點的節線（若下一個節點不存在，則該節線為空節線）；2.雙連列有一個可以連結到第一個和最後一個節點的表頭（若雙連列為空集合，則該連節節線為空節線）。其資料結構可表示如下：

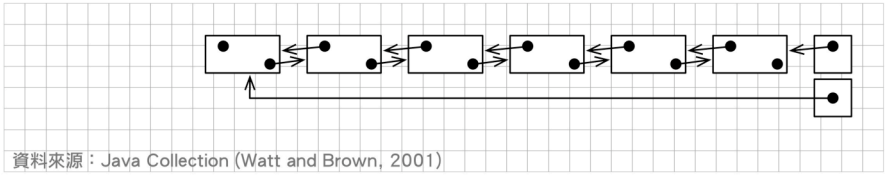

圖6.2 雙連列

本研究之最短路徑搜尋過程，因道路網結構稀疏之故，採用單連列之資料特性，以降低資料儲存所需空間，並可建立起供給、需求點之間的路徑，以形成最短路徑樹。

6.1.2 圖形的表示方法

圖形的元素，可分為點（node/vertex）、線（link/edge/arc）、與面（polygon）。在路網元素中，則以點、線為主要元素。因此，路網圖簡要來說，即為點與線所構成的集合（set）。點集合與線集合必須建構相互關係，方能展現路網的特性。串連點集合與線集合之間關係的方式，會影響路網演算的效率。

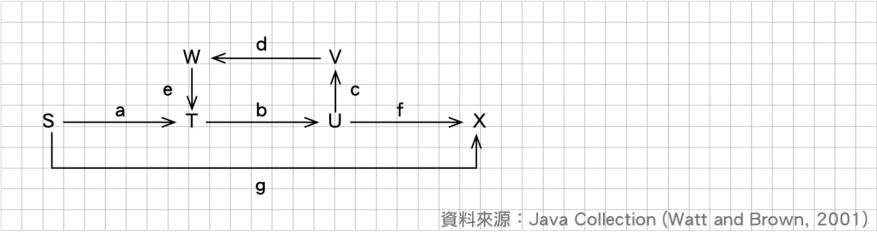

圖6.3 路網範例

圖論（Graph Theory）中，圖形資料表示常見的方式，包括有邊集合、鄰接集合、鄰接矩陣等三種常見的圖形表示法。本研究參考Watt與Brown所著Java Collections之路網範例（圖6.3），來說明此三種圖形表示法之差異：

1. 邊集和

所謂的邊集合表示法，就是將圖形中所有的邊，單獨視為一個集合，並與圖形中單獨的點集合互相對應；此方法即可表示出圖型。

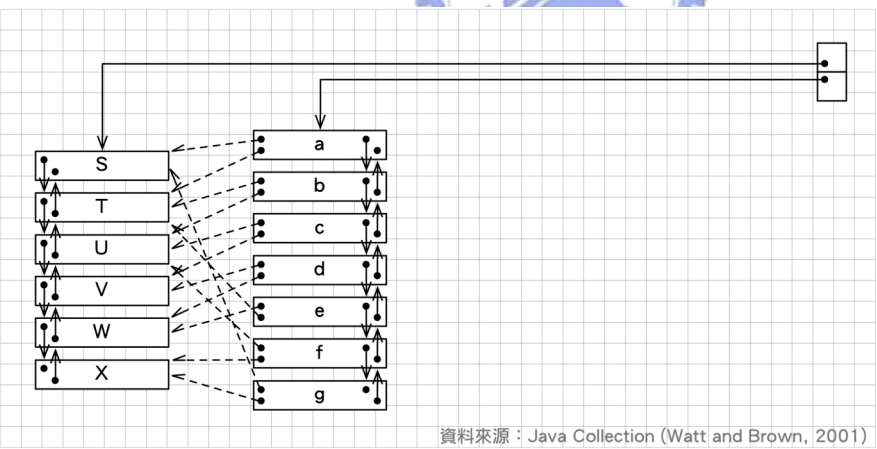

圖6.4 邊集和

2. 鄰接集合

所謂鄰接集合表示法，就是將圖形中的所有點，單獨視為一個集合，並將每個點的鄰接邊集合，與該點作關係的對應。

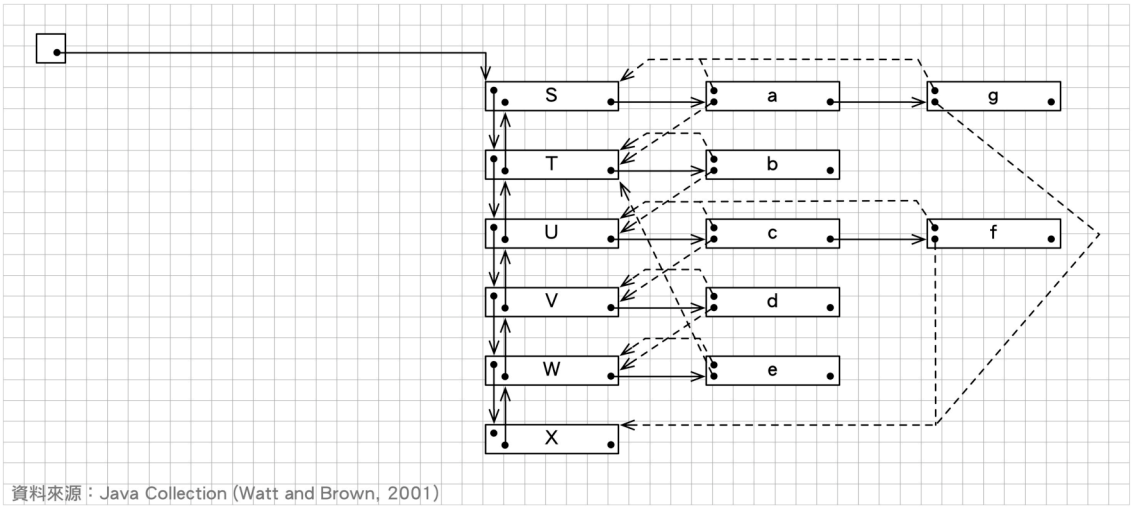

圖6.5 鄰接集合

3. 鄰接矩陣

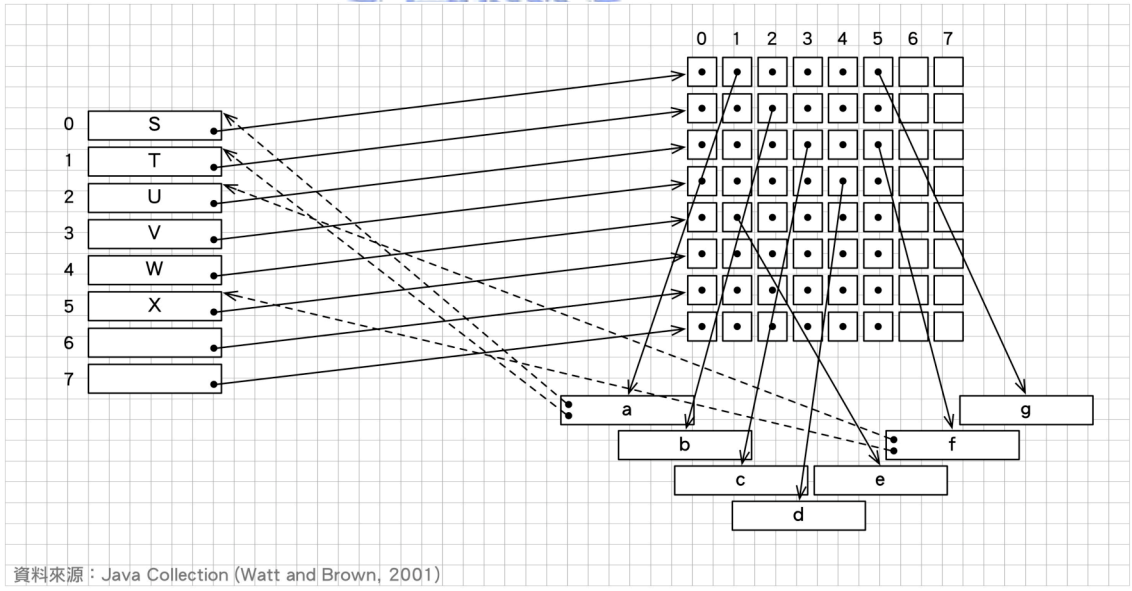

圖6.6 鄰接矩陣

此三種表示法的差異，會使原始資料的輸入成本不同。邊集合的資料輸入，單純的以順序方式，分成點集合、邊集合，建立路網結構的過程，考慮其他點、邊的因素最少；鄰接矩陣的資料輸輸入最為龐大，而且對於道路網而言，過大的矩陣，會有許多浪費的連結關係；鄰接集合的表示法，則介於邊集合、鄰接矩陣之間。不同的資料結構，會直接反應在運算演算法的情形之中；如下表所示，各運算的複雜度，皆會因圖形表示法不同而改變。

表6.1 圖形表示法比較

圖表示法	運算	演算法	複雜度
邊集合	包含邊	線性搜尋DLL	O(e)
	加入點	插入點集合DLL	O(1)
	加入邊	插入邊集合DLL	O(1)
	移除點	刪除點集合DLL 刪除多個邊集合DLL	O(e)
	移除邊	刪除邊集合DLL	O(1)
鄰接集合	包含邊	線性搜尋鄰接集合SLL	O(d)
	加入點	插入點集合DLL	O(1)
	加入邊	插入鄰接集合SLL	O(1)
	移除點	刪除點集合DLL 刪除多個鄰接集合SLL	O(e)
	移除邊	刪除鄰接集合SLL	O(d)
鄰接矩陣	包含邊	矩陣指標	O(1)
	加入點	尋找矩陣的行、列	O(m)
	加入邊	矩陣指標	O(1)
	移除點	清除矩陣行、列	O(m)
	移除邊	矩陣指標	O(1)

資料來源：Java Collections（Watt and Brown，2001）

6.1.3 統一塑模語言的圖形元素

本研究採用邊集和的方式，透過Java程式語言物件的特性，來構件點集合與線集合的關係。本研究並利用Java程式語言來構建所提出的防災路網模型，以下係由統一塑模語言（Unified Modeling Language, UML）的方式，來說明路網中，點與邊所構成的圖形結構。

1. 圖

圖類別（圖6.7）的元素組成，包含點和邊；圖擁有所有點、邊關係的完整資訊。例如：某一點的鄰邊集合、某一邊的兩個端點、以及圖中是否擁有某個點或某條邊等；此外，圖需隨時能加入或移除某些點和邊、以及辨認出局部圖中所包含的供需點集合等；詳細方法如表6.2所述。

表6.2 圖的方法

回傳值類型	方法名稱	功能簡述
Void	addEdge(Edge edge)	在圖形中加入一條邊。
Void	addNode(Node node)	在圖形中加入一個點。
Vector	adjacentNodeSet(Node node)	回傳圖形中某一點的鄰接點集合。
Vector	getDemandNodeSet()	回傳圖形中所有的需求點集合。
Edge	getEdge(Node n1, Node n2)	回傳某兩個點之間的邊。
Vector	getSupplyNodeSet()	回傳圖形中所有的供給點集合。
boolean	hasEdge(Edge edge)	回傳true如果圖形中有某一條邊。
boolean	hasNode(Node node)	回傳true如果圖形中有某一個點。
Vector	incidentEdgeSet(Node node)	回傳圖形中某一點的鄰接邊集合。
void	removeEdge(Edge edge)	移除圖形中的某一條邊。
void	removeNode(Node node)	移除圖形中的某一個點。

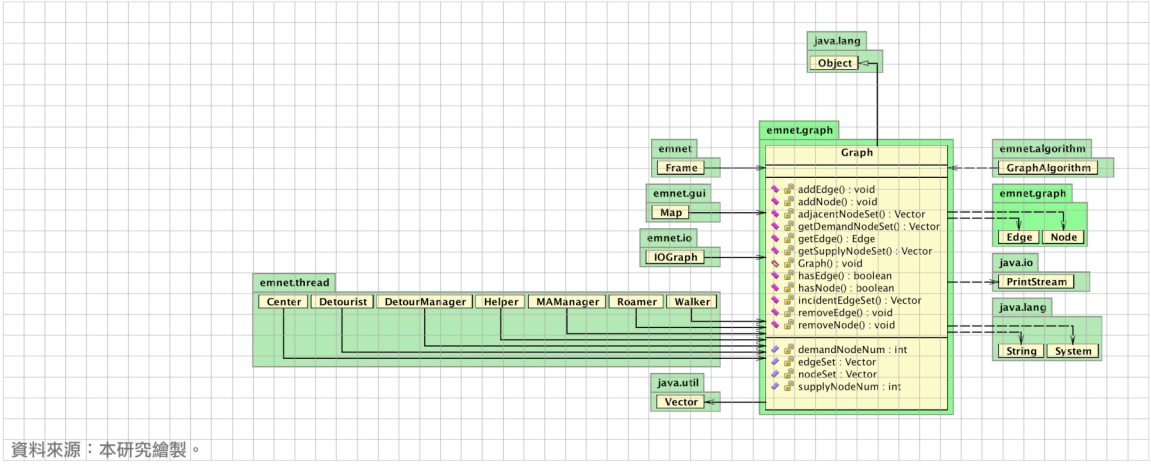

圖6.7 圖形類別

2. 點

點類別（圖6.8）在本研究是圖的唯二元素之一，在尋優演算過程中，需隨時了解各點是否已被拜訪的資訊；因此，需要有能表示拜訪的標記、以及詢問拜訪標記的方法。另外，在進行最短路徑森林的演算過程中，由於多個執行緒同時進行最短路徑搜尋，為要保持演算正確完整性，在某點演算完成之前，同一時間一個點只能被一位漫遊者所拜訪；因此，在針對某點進行演算的過程時，需要有隔絕其他執行緒進入的機制，其他執行緒需等待直至佔據該點的執行緒結束；詳細方法如表6.3所示。

表6.3 點的方法

回傳值類型	方法名稱	功能簡述
boolean	isOccupied()	檢測該點是否正被取做運算。
boolean	isVisited()	檢測該點是否已被拜訪。
void	leave()	設定該點離開被運算狀態。
void	occupy()	設定該點進入被運算狀態。
void	setNeutral()	設定該點不為供需點。
void	unOccupied()	設定該點離開被運算狀態；與leave()功能相同。
void	visit()	設定該點已被拜訪。

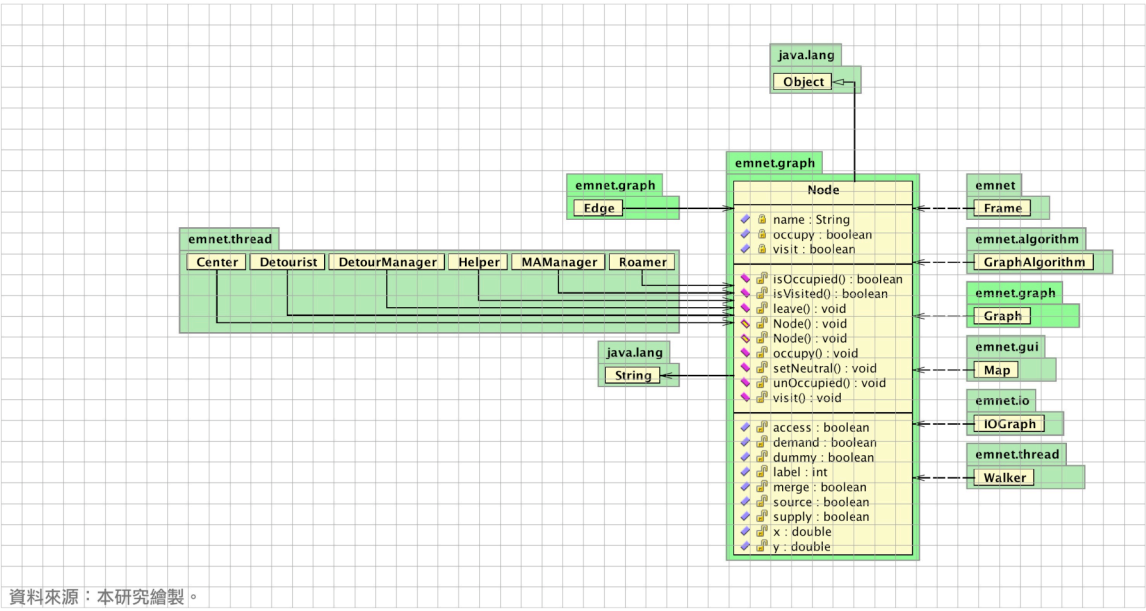

圖6.8 點類別

3. 邊

邊類別（圖6.9）是圖中最重要的元素，因為邊通常被用於展現權重屬性，而成為尋優演算過程中的主要參考要素。邊無法獨立存在，需要一個或兩個點方能組成，而本研究探討簡單路網，不考慮連結在同一點的邊，意即迴圈（loop），因此所有邊都具有兩個端點；也提供分辨某一邊歸屬於何種路網的方法；詳細方法如表6.4所示。

表6.4 邊的方法

回傳值類型	方法名稱	功能簡述
int	getN1Label()	取得點標籤。
int	getN2Label()	取得點標籤。
boolean	isMAEdge()	檢測是否為互援邊。
boolean	isMATestEdge()	檢測是否為互援測試邊。
boolean	isVisited()	檢測該邊是否已被拜訪。
void	setMAEdge()	設定該邊為互援邊。
void	setMATestEdge()	設定該邊為互援測試邊。
void	setNeutralEdge()	是定該邊為中性邊。
Node	theOtherNode(Node n)	取得構成該邊且不為點n的另一點。
void	visit()	設定該邊已被拜訪。

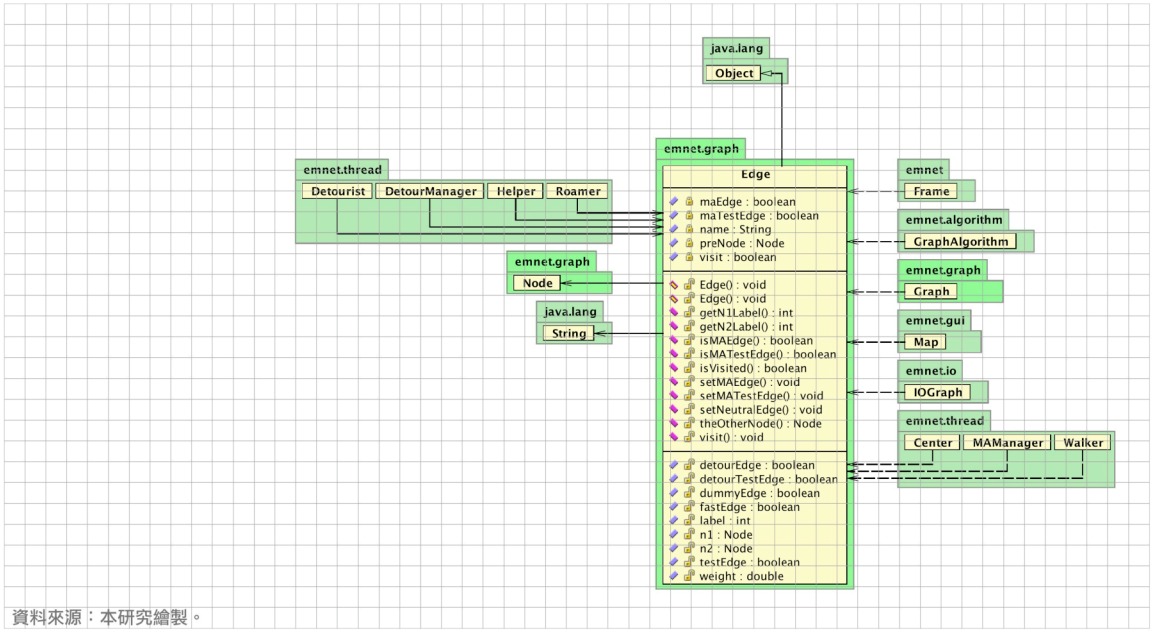

圖6.9 邊類別

6.2 責任分區

6.2.1 演算概念

最短路徑森林，即為許多最短路徑樹所組成的路網，由於最短路徑樹具有一對多的最短路徑特性，因此最短路徑森林具有多對多的最短路徑特性。由第五章路網構建之原理，利用最短路徑森林演算可有效發展多對多最短路徑之演算法。

6.2.2 演算流程

在初始路網之中，運用Java多工執行緒的技術搭配第五章之演算法，使每個供給點可同時依循最短路徑演算法，透過命名為漫遊者（roamer）之Java物件，以各供給點為樹根，各自巡遊並找出獨立的最短路徑樹；發展期間漫遊者將不斷與通訊中心（自定之Java物件，用以蒐集各漫遊者之狀況與資訊）聯繫整體路網的最新狀態，未觸及之領域不斷以最短路徑演算法擴張；已觸及之領域則相互比較、並歸屬於離供給點成本較小的路網之中；此過程不斷循環，直至所有需求點都被囊括於某一樹狀路網集合之中；其演算流程架構如圖6.10所示。

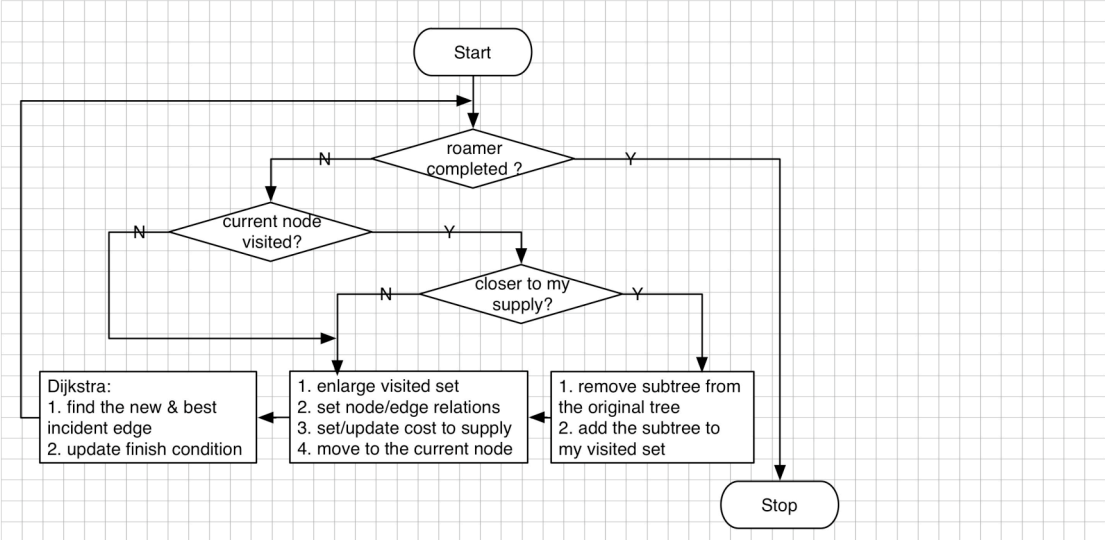

圖6.10 責任分區之演算流程

6.3 系統繞路

6.3.1 演算概念

區內系統繞路路網之目標，主要在消彌因路段毀損而造成部分地區孤立的情形；然而，在受損路段所在位置、及各區域的需求均為未知的情形之下，就須考慮每個路段毀損時，會對供需之間的影響。因此，系統繞路的演算概念，透過模擬區內最短路徑樹之路段毀損後，剩餘不與供給點相連的區域，透過繞路者的尋優巡遊，找出對內潛在需求點之聚集成本、對外橋接成本組合為最小的節點與路徑，已作為區內系統繞路之路徑。

6.3.2 演算流程

在最短路徑樹中，透過命名為繞路者（detourist）之Java物件，以最小系統成本，探求出另一條替代路線，以達到分區內的救援單位。首先挑選出最短路徑樹中某一路段，模擬其為無法使用，繞路者採取聚集成本、橋接成本最小的尋優方式，循環比較，直至連結至原供給單位的成本最小為止，則獲得該路段毀損時區內繞路之最佳路徑，其演算流程架構如圖6.11所示。

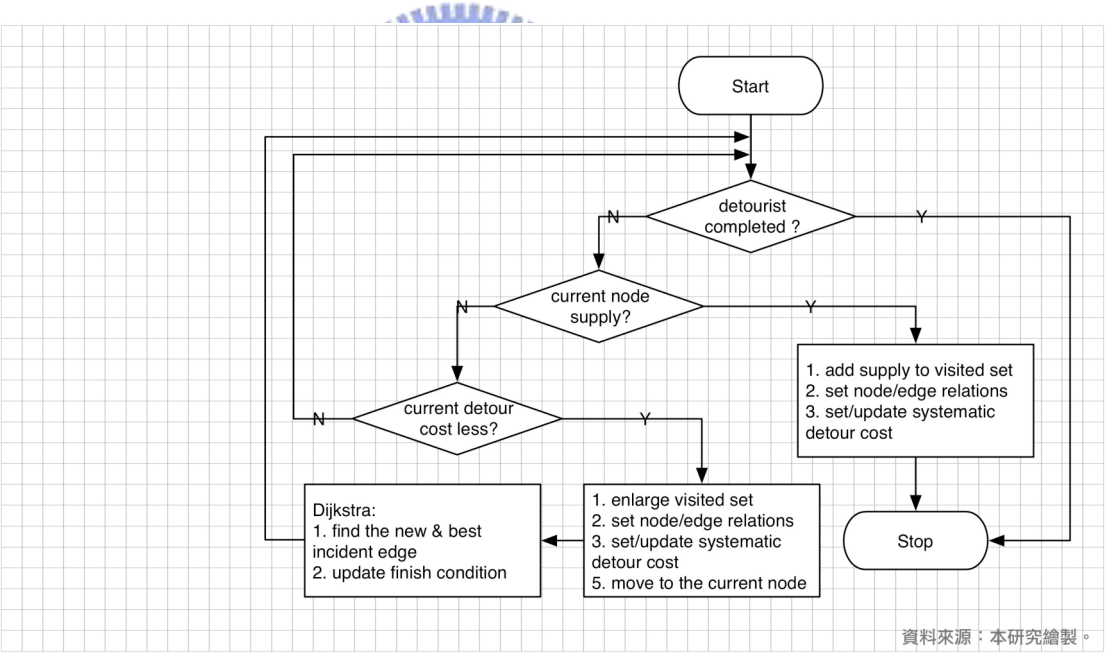

圖6.11 系統繞路之演算流程