

國立交通大學

交通運輸研究所

博士論文

No. 049

防震災存活路網設計模型

Survivable Network Design Model for Earthquake Disaster

研究生：侯鵬曦  
指導教授：徐淵靜

中華民國九十五年十二月

防震災存活路網設計模型  
Survivable Network Design Model for Earthquake Disaster

研 究 生：侯鵬曦  
指導教授：徐淵靜

Student: Peng-Hsi Hou  
Advisor: Yuan-Ching Hsu

國立交通大學  
交通運輸研究所  
博士論文

A Dissertation  
Submitted to Institute of Traffic and Transportation  
College of Management  
National Chiao Tung University  
in Partial Fulfillment of the Requirements  
for the Degree of  
Doctor of Philosophy  
in  
Management

December 2006  
Taipei, Taiwan, Republic of China

中華民國九十五年十二月



# 防震災存活路網設計模型

學生：侯鵬曦

指導教授：徐淵靜

國立交通大學交通運輸研究所

## 摘 要

防災路網是防災城市的重要空間系統，因此防災路網之設計原理，乃是一重要議題。本研究針對震災特性，提出評量防災道路之空間防災性、路段關鍵性等方法，並應用存活路網理論來發展整體防災路網模型，設計目標為可靠、快捷、與全面。最短路徑森林路網、系統繞路路網、及互援路網，將此三種模型整合於存活路網模型。該模型具有諸多優勢；最短路徑森林確保責任分區內救援單位至災害需求地點的路徑最短；系統繞路則容忍區內路段遭受災害毀損而仍維持救援單位至區內所有需求場所的成本最小；互援路網構成區間橋接路徑以連結外部資源，提供區間救援單位相互備援的保障。最後本研究透過Java技術實作此存活路網模型，並提出棋盤式測試路網，以進行雙供給均勻需求分佈之情境模擬，結果發現，供給點彼此逐漸遠離時，平均旅行成本趨勢下降，繞路成本陡降漸緩，互援成本逐漸增加，而路網成本初期變幅明顯，而後漸減收斂。

**關鍵字：**存活路網、災害、繞路、連結度

# Survivable Network Design Model for Earthquake Disaster

Student: Peng-Hsi Hou

Advisor: Dr. Yuan-Ching Hsu

Institute of Traffic and Transportation  
National Chiao Tung University

## Abstract

Disaster-prevention network is a crucial system in a disaster prevention city. Therefore, how to establish design principles for the network becomes an important issue. In this paper, we proposed the method to assess the spatial performance and the importance against disasters in individual road sections. In the viewpoint of comprehensive network, we proposed the network design model with respect to the characteristics of earthquake disaster based on survival network theories. The design goals are to achieve features of rapidity, connectivity, and efficiency. Three network models: Shortest Path Forest Network (SPFN), Systematic Detour Network (SDN), and Mutual Assistance Network (MAN), are developed and integrated in the survivable network model. The proposed survivable network for earthquake disaster has significant advantages. The SPFN promises the shortest path within the responsible territories. The structure of SDN assures the network connectivity wherever any road link fails. The bridges of MAN let the supply units in different territories connect to each other hence insufficient self-capacity of one supply may be allowed. Finally the survivable network model was implemented with Java technology. And a case in a test network of grid pattern was given to show the features in the scenario of evenly scattered demand areas with two supply resources. It was found that where the two resources were increasingly separating from each other, the average travel cost steadily decreased; detour cost dropped dramatically and then declined slowly; mutual assistance cost gradually increased; network cost seriously varied in the initial stage, but converged to a gently decreasing trend afterwards.

**Keywords:** Survivable network, disaster, detour, connectivity

## 誌 謝

漫長的日子裡，本論文得以順利付梓，要感謝的人很多。首要感謝指導教授徐淵靜老師，在研究過程中給予我的諸多支持與鼓勵。並感謝陳武正老師、羅啟源老師、林峰田老師、錢學陶老師、馮正民老師、汪進財老師等口試委員提供我諸多寶貴意見，使本論文更加完整。感謝求學期間所內的藍武王老師、黃承傳老師、黃台生老師、許鉅秉老師、曾國雄老師、溫傑華老師給予的珍貴教誨，修業期間台大數研所張鎮華教授的教導，投稿期間指正建言的M.G.H. Bell教授、賞識本研究貢獻的林國峰教授。以及博班學長姐們、同學輝煌、銘德、明智、淑美、曾董，以及彥蘅、亦姝、昱凱、益三、易詩、Jacky、永祥，國科會研究夥伴們，陳苑惠老師、孟慧、高桂、岳德、容禎、兆鉅、青峰、俟之，還有叮嚀生活細節的所辦洪小姐、柳小姐、何小姐，協助諸多出國事項的研發處陳善理小姐，每個人對我的幫助，都是砌疊此論文的重要基石。

求學期間，受資策會副執行長黃國俊博士的提攜，尤為感謝。此期間也讓我認識了不可多得的貴人，資工所朱海燕副所長對我積極鼓勵，紀舜、駿逸、立豪、濠瑞等好友，使我對資訊工程的認識更加深入而廣泛，產支處何文雄處長給我的箴言與肯定，Frances、賢書、宜葶、書彥、惠敏、培燕、Sophia、Luke、羅姐、簡大哥，生活的歡笑煩惱，一路陪我走過論文的最終時光；此外，部長宗承、玟如，是我碩士畢業後少數仍持續聯繫的學弟妹，以及部長的大學好友俊逸工程師，謝謝他有興趣一起研究程式。

最後感謝陪伴我數十年的、親愛的家人與關心我的摯友。

十二月 · Percy · 三重



# 目 錄

摘 要 .....	I
ABSTRACT .....	II
誌 謝 .....	III
目 錄 .....	V
表目錄 .....	VIII
圖目錄 .....	IX
第一章 緒論 .....	1
1.1 研究背景與動機 .....	1
1.2 研究目的 .....	1
1.3 研究方法 .....	2
1.3.1 研究理論 .....	2
1.3.2 研究工具 .....	3
1.4 研究內容 .....	3
1.4.1 研究架構 .....	3
1.4.2 章節安排 .....	4
第二章 文獻回顧 .....	5
2.1 防災實務相關研究 .....	5
2.1.1 都市防災系統 .....	5
2.1.2 防災據點 .....	5
2.1.3 防災路網 .....	6
2.1.4 綜合評析 .....	7
2.2 防災評估相關研究 .....	7
2.2.1 道路評估相關指標 .....	7
2.2.2 路網評估相關指標 .....	8
2.2.3 綜合評析 .....	10
2.3 路網模型相關研究 .....	10
2.3.1 最短路徑 .....	10
2.3.2 繞徑問題 .....	11
2.3.3 存活路網 .....	13
2.3.4 綜合評析 .....	16
第三章 防災路網分析 .....	17
3.1 問題陳述 .....	17
3.1.1 現況問題 .....	17
3.1.2 課題探討 .....	18
3.2 地震與防災路網 .....	19
3.3 防災路網系統之組成 .....	20
3.4 防災路網系統之功能 .....	21

3.4.1 需求面.....	22
3.4.2 供給面.....	22
3.4.3 可控制的路線系統.....	22
3.5 防災路網系統之架構.....	23
3.5.1 功能架構.....	23
3.5.2 系統架構.....	24
<b>第四章 防災道路評估.....</b>	<b>27</b>
4.1 防災道路關鍵性評估.....	27
4.1.1 評估方法.....	27
4.1.2 路網規模.....	27
4.1.3 評估指標.....	28
4.1.4 指標演算.....	28
4.2 防災道路空間性評估.....	29
4.2.1 評量架構.....	29
4.2.2 道路空間評估指標.....	30
<b>第五章 防災存活路網建構.....</b>	<b>33</b>
5.1 責任分區路網模型.....	33
5.1.1 責任分區之定義.....	33
5.1.2 責任分區之陳述.....	34
5.1.3 責任分區之原理.....	35
5.1.4 責任分區之演算法.....	37
5.2 系統繞路路網模型.....	39
5.2.1 系統繞路之定義.....	39
5.2.2 系統繞路之陳述.....	39
5.2.3 系統繞路之原理.....	40
5.2.4 系統繞路之演算法.....	42
5.3 互援路網模型.....	42
5.3.1 互援路網之定義.....	42
5.3.2 互援路網之陳述.....	43
5.3.3 互援路網之原理.....	44
5.3.4 互援路網之演算法.....	45
5.4 防災存活路網評估模式.....	46
5.4.1 防災存活路網模型.....	46
5.4.2 防災存活路網評估指標.....	46
5.4.3 防災存活路網之綜合評估.....	47
<b>第六章 防災存活路網程式.....</b>	<b>49</b>
6.1 路網結構.....	49
6.1.1 資料結構.....	49
6.1.2 圖形的表示方法.....	50
6.1.3 統一塑模語言的圖形元素.....	53
6.2 責任分區.....	56

6.2.1 演算概念.....	56
6.2.2 演算流程.....	56
6.3 系統繞路.....	57
6.3.1 演算概念.....	57
6.3.2 演算流程.....	57
6.4 互援路網.....	58
6.4.1 演算概念.....	58
6.4.2 演算流程.....	58
6.5 路網範例與操作 .....	59
6.5.1 路網構建.....	59
6.5.2 應用操作.....	63
<b>第七章 棋盤式路網特性分析 .....</b>	<b>67</b>
7.1 測試路網.....	67
7.2 供需點情境設計 .....	69
7.3 測試路網之設計結果 .....	71
7.4 綜合建議.....	76
<b>第八章 結論與建議 .....</b>	<b>77</b>
8.1 結論 .....	77
8.2 建議 .....	79
<b>參考文獻.....</b>	<b>81</b>
<b>附錄一 模擬數據結果 .....</b>	<b>87</b>
<b>附錄二 路網範例 .....</b>	<b>91</b>
<b>附錄三 程式碼 .....</b>	<b>101</b>
<b>個人簡歷.....</b>	<b>235</b>

## 表目錄

表6.1 圖形表示法比較 .....	52
表6.2 圖的方法.....	53
表6.3 點的方法.....	54
表6.4 邊的方法.....	55
表7.1 棋盤式測試路網之點資料 .....	67
表7.2 棋盤式測試路網之邊資料 .....	68
表7.3 雙供給點情境設計 .....	70



## 圖目錄

圖1.1 研究架構.....	4
圖2.1 防災據點功能 .....	6
圖3.1 防災路網系統之組成.....	21
圖3.2 防災路網之供需架構.....	23
圖3.3 防災路網之功能架構.....	24
圖3.4 防災路網之系統架構.....	25
圖4.1 防災路網之設計規模.....	27
圖4.2 關鍵性指標演算流程.....	28
圖4.3 道路空間解構 .....	30
圖4.4 防災道路空間之評量架構 .....	32
圖5.1 防災路網模型架構 .....	33
圖5.2 最短路徑樹符號定義.....	34
圖5.3 簡理一 .....	35
圖5.4 簡理二 .....	36
圖5.5 責任分區演算法概念.....	38
圖5.6 系統繞路模型 .....	41
圖5.7 防災路網結構 .....	43
圖5.8 互援路網原理 .....	44
圖6.1 單連列 .....	49
圖6.2 雙連列 .....	49
圖6.3 路網範例.....	50
圖6.4 邊集和 .....	50
圖6.5 鄰接集合.....	51
圖6.6 鄰接矩陣.....	51
圖6.7 圖形類別.....	53
圖6.8 點類別 .....	54
圖6.9 邊類別 .....	55
圖6.10 責任分區之演算流程.....	56
圖6.11 系統繞路之演算流程.....	57
圖6.12 互援路網之演算流程.....	58
圖6.13 點資料之輸入界面 .....	59
圖6.14 點資料轉檔方式 .....	60
圖6.15 邊資料之輸入界面 .....	61
圖6.16 邊資料轉檔方式 .....	62
圖6.17 點邊資料輸入 .....	63
圖6.18 供需點設定方式 .....	64
圖6.19 路網演算結果.....	65
圖6.20 專家意見之輸入介面.....	66
圖7.1 邊集合表示法 .....	67
圖7.2 棋盤式測試路網 .....	69
圖7.3 ATC之趨勢.....	71

圖7.4	<i>MTC</i> 之趨勢 .....	72
圖7.5	<i>LD</i> 之趨勢 .....	73
圖7.6	<i>AMAC</i> 之趨勢 .....	74
圖7.7	<i>NC</i> 之趨勢 .....	75

# 第一章 緒論

## 1.1 研究背景與動機

台灣關於地震災害的研究，自從921集集大地震之後便蓬勃發展。此乃因該地震造成了台灣近年來死傷人數規模最大的災情，因此受到了高度的關注。事件發生後，郊區、山區坍塌嚴重，導致道路系統癱瘓，並形成路面多處被孤立的地區，並須藉由穿梭來回災區的直昇機運送傷患民眾、各種民生物資，以彌補毀損道路系統中斷的問題。類似的情景發生如中橫公路多處坍塌、南投與台中之間的連絡橋被震落，而致完全無法使用。此外許多重要聯絡道路的路面破裂造成落差，車輛因而通行受阻。相較於郊區，都市的災情亦為顯著。由於都市地區人口與經濟活動聚集密度高且建築物密集，地震造成之建物倒塌與路面破壞，及地震所延伸的二次災害，對民眾的生活與都市的機能影響甚鉅。

都市災害防制的觀念，雖然近年來逐漸受到重視；然而，在都市規劃與建設發展的過程中，如何落實防災觀念，將相關規劃與設計轉而為具體的設施與設備，仍尚有許多應努力之處。而在整體都市防災系統中，避難空間系統與道路空間系統之規劃與設計尤其重要，並為減輕災情最關鍵的項目。由於地震災害後，大規模的救援行動，都需藉由道路網提供救援單位來完成，而防災路網與防災據點之間的密切配合，才將可使緊急應變作業順利進行。

根據內政部建築研究所的研究報告指出（李、錢，民88），防災型都市，短期以訂定緊急應變計畫、中期以充實防災設施與健全防災體系、長期以建構安全的防災都市為目標。其中，緊急應變計畫之訂定目的，乃在建構都市完整之避難圈。而緊急應變計畫的主要的工作內容，即包括了避難空間、防災據點、防救交通動線系統等的指定，以形成都市防災避難路網。

防救災交通動線系統之指定即為本研究的焦點，本研究簡稱該系統為「防災路網」。交通動線系統影響救災效率甚鉅，指定交通動線系統，應有其合理的邏輯思惟，方能發揮路網的最大效益。相較於短期緊急應變之做法，本研究將防災路網之指定規劃與設計，定位於朝向中期落實防災設施之理念進行，最後並達成長期防災都市常態基礎建設之願景目標。

## 1.2 研究目的

防災路網之目標，為災時能保持路網之可靠、快捷及全面之條件。防災路網之規劃，應同時從道路環境、救災單位、及救災活動三方角度進行；防災路網之落實，則應著眼於長期的基礎建設。由於震災發生時點無從預期，時間敏感度難以估計，故應以長遠眼光來看，建立詳盡的模式以審視道路空間，進而了解救災單位、救災活動路線等與道路之間的緊密關係，方能整合而規劃出完善之防災路網。

本研究之願景，乃期許台灣各城市皆有完善之防災路網基礎。因此，即便在地震災害發生時，所有相關緊急疏散、救援、物資輸送等相關行動，皆能藉受助於此防災路網有效運作而不受災害外力影響，並藉

以使原日常交通正常運行而毫無窒礙。本研究之目標，乃提出此路網結構之樣態；本研究之標的，乃藉由以下項次達成研究目標。

(1) 提出防災道路之路網結構：路網在整個地震災害時期，為提供媒介系統的關鍵角色，在整個地震發生的期後，很明顯，當避難、救援、輸送物資、搶修工程、運送廢棄物等工作進行的同時，其重要性絕對是毫無疑問的。然而，就聯絡各種活動的路網通道而言，是否有一個合宜的路網結構，即便是路網受到破壞的情況下，仍能維持原來機能；或，能減少孤立地區的數量；或，能夠提高各工作的行進效率；亦或是，在各工作彼此間的相互干擾中，能夠合理的控管。本研究首先探討防災路網的模型，探討合宜之目標架構；並透過規劃的程序，完成模式運算所需之基本資料；最後並以路網模型為核心，實作使用者介面，以提高實用性。

(2) 建立防災道路路段之評估方法：本研究以防災觀點提出一套道路的空間品質評估方法，將可運用於實際調查的過程當中。並訂定三維空間指標，綜合作為衡量防災道路路段的適切性。

(3) 建構防災路網模型與評估方法：本研究欲從震災的角度與觀點，構建一個路網模型，具備多項良好特性，以提供地震災害相關運輸活動能夠順利進行的路網環境。並建立評估不同防災道路模型之方法，以作為不同路網間相互比較之基礎。

(4) 提供路網設計之操作介面：實作路網設計之操作的介面，可提高模式、規劃程序的實用性，並可減低路網規劃時間及成本。

## 1.3 研究方法

### 1.3.1 研究理論

本研究在道路評估方面，採用了分析階層程序法；路網建構方面，則採用圖形理論相關方法。

(1) 分析階層程序法：分析階層程序法 (Analytic Hierarchy Process, AHP) 為 Saaty 於 1970 年所發展出來的量化方法，廣泛應用在許多領域，用來排定方案屬性品質的優劣順序。最好的決策案，為由學者專家使用權重獲致最高值的選擇案。分析階層程序法之特色為將評估問題分成許多層級與單元，並透過受訪者對各單元內的項目進行經驗衡量之判斷，再將各部分判斷結果重組成方案的優劣關係。

(2) 圖形理論：圖形理論 (Graph Theory) 係探討點、線、面等元素關係的一種理論。現實生活中的許多關係，都可以輕易的轉換為點、線、面關係後，再透過圖形理論的方法進行分析。而路網問題，本身即為幾何圖形的一種；因此，路網問題更適用圖形理論加以詮釋，以塑造模型來重現問題。本研究防災路網規劃所採用的圖形理論，包括：最短路徑樹模型、存活路網模型、以及繞路模型。

### 1.3.2 研究工具

本研究為將路網規劃模型之應用與地理資訊資料相容性提高，乃採以地理資訊資料由ArcView軟體輸出，透過Excel軟體來轉換格式，並由研究撰寫之程式讀入，來進行演算工作；最後由研究撰寫之程式來顯示成果。研究過程所運用之研究工具，包含以下幾項：

(1) ArcView：ArcView為ESRI公司所出產之軟體，為地理資訊系統軟體之市場主流之一。

(2) Microsoft Excel：Excel為Microsoft公司開發之軟體，提供數學、統計、資訊、工程計算等應用功能。

(3) Expert Choice：Expert Choice為專家決策輔助系統軟體；核心原理係採用Satty所提出之分析階層程序法。

(4) J2SE：J2SE全名為Java 2 Platform Standard Edition，為昇陽（Sun）公司開發之程式語言，提供了桌上型、伺服器等工作電腦，完整的應用程式；它也是企業版本J2EE、以及網路服務（Java Web Service）的基礎核心。Java之程式語言，具有簡單、高效能、安全、分散式、動態式、可攜、豐富、物件導向、多執行緒、結構化、解譯式等特性；透過Java虛擬機器（JVM），使得Java程式可以達到「write once，run anywhere」的理想。

(5) Swing：Swing為Java程式中，使用者介面（User Interface，UI）的元件庫；相對Java早期處理圖形的技術（Abstract Windowing Toolkit，AWT），具有使用較少的系統資源、提供更多的元件、可根據程式設計來設計特殊的外觀等優點，為程式開發者提供了許多方便有效的類別。

(6) JBuilder：JBuilder為Java之整合性開發環境，本研究採Java程式語言，來進行路網規劃模式之相關演算，利用JBuilder所提供之Swing元件庫圖形化工具，可快速實作使用者介面之應用程式，以提高本研究之實用性。

## 1.4 研究內容

### 1.4.1 研究架構

本研究主要分為五大部分（圖1.1），包括：研究問題與課題、文獻回顧、防災路網架構、防災路網模式、路網特性分析、以及結論與建議。其中防災路網架構，分為防災道路評估、與防災路網建構兩大方向。

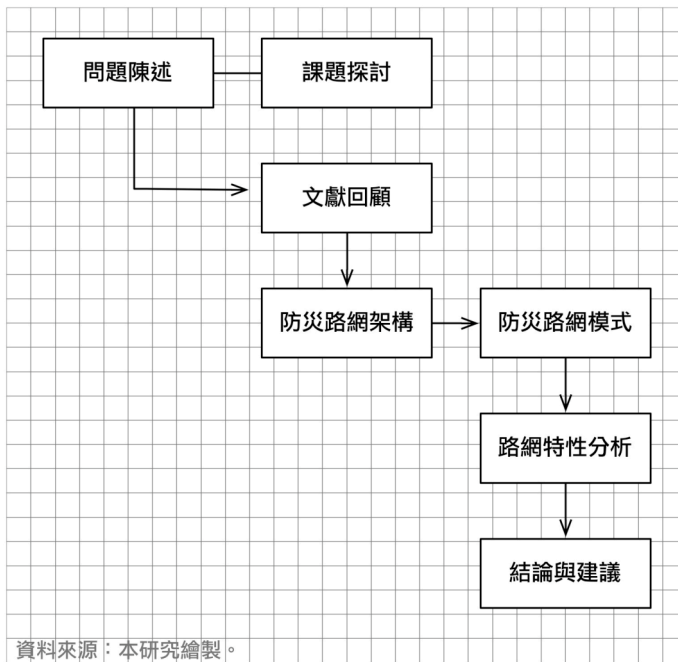


圖1.1 研究架構

#### 1.4.2 章節安排

本研究包含八個章節。在第一章中，陳述本研究之相關問題，勾勒出本研究之願景，並簡要的說明所採用之相關理論方法、工具，以及摘要出主要的研究內容。第二章為文獻回顧。由過去的實務相關研究、防災評估研究、路網模型研究等三方面進行資料蒐集，最後彙整比較，並提出看法及評論。第三章為防災路網分析。該章節探討地震災害與防災路網之間的關係，並由路網基本組成元素、系統功能之討論，提出防災路網系統之概念架構。第四章為防災道路評估。該章節提出並發展「道路關鍵評估」、「道路空間評估」兩個模組。首先，從防災功能性的角度，探討防災道路的角色定位；其次，分別訂定評估道路關鍵性與空間性的相關指標；最後，提供指標的計算方法。第五章為防災路網建構。該章節提出並發展「責任分區路網模型」、「系統繞路路網模型」、「互援路網模型」三個模組。首先，針對各模組做出明確之定義，並進一步陳述其內容，探討其內涵原理，再提供各模組之演算方法。最後，並提出「防災路網評估」模組，訂定防災路網之評估指標，建立防災路網之綜合評估模式，以做為不同防災路網結構之比較基準。第六章為防災存活路網程式。提出程式之架構、原理流程與操作方法。第七章為棋盤式路網特性分析。提出正方網格之路網型態，探討其路網結構及其效果。第八章為結論與建議。該章節總結本研究之貢獻性，以及提出未來可繼續研究與發展之方向。

## 第二章 文獻回顧

### 2.1 防災實務相關研究

#### 2.1.1 都市防災系統

對於都市防災之相關研究，過去曾提出許多觀點面向。在都市計劃防災系統規劃方面，對於地震災害及其可能引發之火災，相關研究提出防災生活圈之觀念。其架構除避難、醫療、物資、消防、警察等五大防災避難據點外，主要包含救災、避難、延燒阻隔三大防救災功能；救災路線功能為救災物資、器材及人員之運送道路，須擔負消防活動、各防災據點物資運送之路徑，因此宜分開規劃；然若受到實質環境條件因素，則須考量避難、救災路線合併、共同管制方式進行之；因此面對防災道路空間品質現況，如進一步檢討，則對整併或分開規劃可提供參考之方向。

過去關於都市防災系統的相關實務研究，包括：都市防災計畫（台灣省住都處市鄉局，1998）、都市計劃防災系統的規劃（張，1999）、防災空間系統規劃（李，1999）、規劃實務案例（李、謝，1999）、災害境況模擬（陳、林、賴，1999）、防救災交通動線系統及防救據點（何、李，1998）、避難救災路徑有效性（陳、詹，1999）、防災都市整備計畫（黃，1999）、防災規劃作業（何、黃，1997）、災害危險度評估（林、陳，1998）、防災規劃作業程序及設計準則（陳、黃、黃，1999）、國土城鄉防災綱要計畫（林，2003）等。

#### 2.1.2 防災據點

國內對於防災據點之規劃，採以現有設施之防災空間分析方法（李、錢，1999）。透過研究調查區內之實質環境與使用現況之調查分析，探討學校、公園等防救據點的防災對應力，以檢測都市空間因應防災機能，於空間量、區位選定之適切性。防災系統分為行政單元、避難、醫療、物資、消防、警察六大構面，來對應出實質據點空間；實質空間則以現況概要（據點定位、據點地址、據點面積、相關設施、出入口數、周邊建築現況）、危害度分析（自然環境災害、建築物災害、維生管線災害、交通系統、火災）、總合評估（可及性、有效性）等三類，共十四項準則評定之。

國外對於防災據點，有詳盡的規劃與著墨。如圖2.1為日本內閣府之防災白書所提出的基礎廣域防災據點整備的概念，其防災公園平日所扮演的休憩功能，可於災害時轉換成完全的防災據點，這是由於公園的規劃本身，強烈隱含防災據點所需具備之設施與功能。

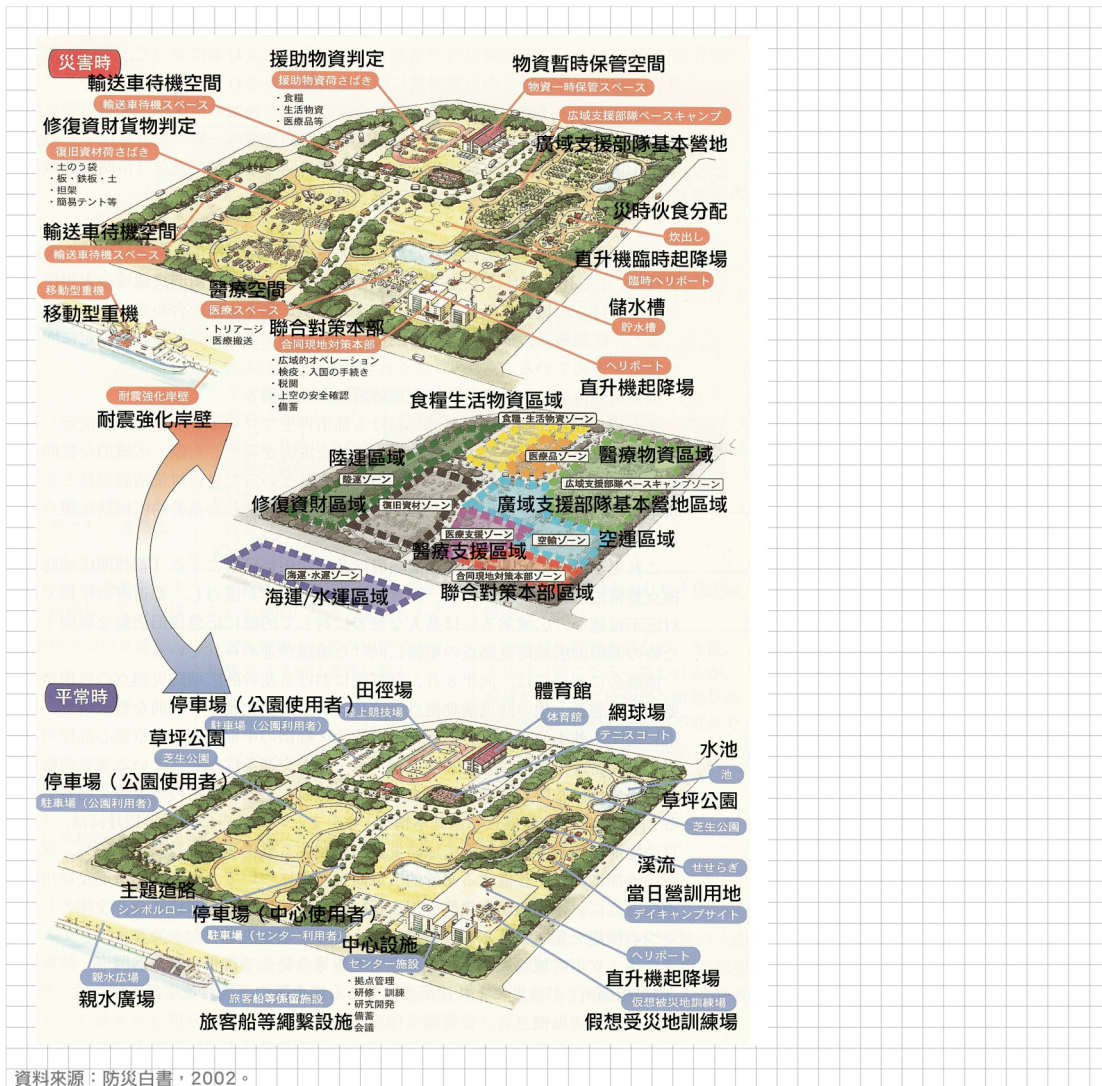


圖2.1 防災據點功能

### 2.1.3 防災路網

與防災據點同樣的思維，道路系統應具有轉換成防災路網的能力；同時，防災道路應具備防災設施、設備以提供防災功能。在日本神戶市復興計畫中，明示要將都市交通路網，對應都市空間，以構成廣域防災交通主軸基礎（1994）；此即指出，都市交通路網與防災路網之間，有著相當密切關係的事實。廣域防災交通網之規劃，主要是以對應都市空間結構為基礎，來規劃廣域交通軸、防災中樞據點、以及路海空防災據點。防災計畫之交通系統與兩項原則有關：1.都市與區域間長程交通系統規劃設置原則、2.都市性交通系統及設施規劃設置原則（何、黃，1997）。前者主要的目標，在規劃長程交通運輸系統防災之轉運連結路網，以疏散車流，降低區域間之人流、車流、物流之使用時間與空間之擁擠。後者主要的目標，係依據都市型態與適當運具，來健全都市內交通運輸系統之防災功能，以降低都市內之人流、車流、物流之使用時間與空間之擁擠。由此可知，防災路網之規劃，需與都市空間結構緊密配合，方能發揮其效用。



防災道路具有延燒阻斷之功能。為防止都市大火，日本江戶時代就已開始規劃防火帶以防止火勢延燒。採取之法，包括整建防火地、防火土，明治時代之不燃建築物防災計畫，以及現代耐火樹木植栽防火等都市防災方法（李、錢、李，1997）。

防災道路首次出現於日本為西元1879年北海道函館的「二十間坂」（にじゅっけんざか）。「間」是日本的長度單位，二十間約相當於36公尺，具有火災遮斷功能，發揮防火帶功能，其目的在抑制火災的蔓延，該防災道路的建造也是日本近代都市計畫的第一個案例。當時之策略，即採以道路寬度作為防災的手段。

目前台灣防災道路系統分類依據，主要係根據日本防災經驗，將防災道路系統劃依道路寬度分為緊急道路、救援輸送道路、避難輔助道路；緊急道路為20米以上之都市計畫道路，救援、運輸道路為15米以上之都市計畫道路，避難輔助道路為8米以上都市計畫道路（李、錢、李，1997）。

然仍有其他重要環境因素，如道路兩側空間的使用情形，也會對道路系統造成間接的影響。針對阪神地震所影響住宅地區街道之破壞進行分類的工作，以路網結構、道路寬度和臨街建築結構等資料分析，結果顯示臨街3米遠的木造房屋，才較不易對街道封閉造成影響（Tsukaguchi et al., 1999）；因此防災道路空間品質之評估，應考量道路兩側的建築物情形，以將道路通行之風險納入。

#### 2.1.4 綜合評析

根據國內外的實務研究可以發現，目前較著重在防災路網規劃的相關研究，多採以現存都市路網結構中，配合學校、公園等據點，並以路寬條件作為防災道路指定的準則，來進行路網之指定規劃。

以台灣防災道路規劃之現況觀察，防災道路係架構於都市現存路網之上，並以道路寬度為準則，用以歸屬道路之防災功能定位。此法之優點在於準則明確，防災路網結構易於確立，且道路寬度具空間特性，某種程度可反映安全性、通暢性之雙重意義；但若兩段相同路寬條件，如一者路側為公園空地，另一者雙側狹擠老舊公寓，因兩者為相同路寬，結果可能獲致相同評價。此外，該指標之獨立性，亦可能使道路與救災單位之區位配置等研究脫勾，較不易展現防災路網結構與救災單位之間的關係。

### 2.2 防災評估相關研究

#### 2.2.1 道路評估相關指標

##### 1. 避難救災功能評估指標

陳、詹（1999）提出幾項重要的單一指標，以檢視路段的避難救災功能。其中包括：

- （1）街道調和比：街道調和比，為沿街建築物高度（ $H$ ）與道路寬度（ $D$ ）之比率。該比值建議以1-1/3之間為適當、1-2/3為適宜。該比值顯示出道路空間呈現閉鎖的機率，若比值過大，於大火災時有飛灰之危險性，而於地震時，建築物倒塌將使道路阻

塞，而妨礙緊急救援機具之通行。

(2) 街道建物比：街道建物比與街道調和比不同之處，乃在此指標以街道兩側之建築物數量來代表活動量。然而，此指標同時也隱含建築物數量影響道路有效寬度之因素；做此解釋時，則與街道調和比有相似之意涵。

(3) 路段人口負荷比：路段人口負荷比，在衡量一條路段的疏散效果。若於單位時間內匯集容納的人車過多，則將不利於避難救援。

(4) 高危險性路段：高危險性路段，係指位於地震帶、斷層、鬆軟地盤或環境敏感地區；或有高架橋、陸橋橫越，以及有地下管線經過處。地震容易造成路面斷裂、地層下陷而遭阻斷無法通行。

(5) 停車所佔道路長度與面積比例：路邊停車是影響避難救援最直接且最重要的因素。因為停車問題會直接影響道路空間，而降低了道路避難救援之要求。

(6) 道路兩側落下物：道路兩側物體於地震時，皆有可能掉落而影響避難救援道路之功能，因此指定為防災路網之道路，應避免兩側有掉落物之情形發生。這樣的掉落物包括有招牌、建物外牆、窗子、冷氣、電線桿等。

## 2. 佔有指標

在許多地區，狹窄的街道往往密度很高，要建造新的街道或是拓寬現有的街道並不容易，因此 Tsukaguchi、Jung (2002) 提出了一種新的方法來檢驗道路空間的配置是否合理 (Tsukaguchi and Jung, 2002)。他們延伸交通流佔有指標 (occupancy indicator of traffic flow) 的概念並加以擴展，提出新的佔有指標，這個新的指標可以檢驗住宅街道之道路空間配置是否恰當，此外，此研究不僅能對單一路段分別檢驗，也提出了檢驗整個住宅地區街道服務水準的方法。

## 3. 人行道指標

在行人設施方面，Mori、Tsukaguchi (1987) 提出兩種不同的方法：第一種是基於行人行為來評估，另一種則是基於行人的意願來評估 (Mori and Tsukaguchi, 1987)。透過行人密度和道路寬度等指標，來衡量出使用人行道的服務品質，然而一般來說，人行道都不足以滿足行人流 (pedestrian flow)，因此透過另一種以行人對人行道的認知來加以衡量。前者方法可用於所有的人行道，特別是行人流較大的人行道；後者則適用於行人流較小的人行道。

### 2.2.2 路網評估相關指標

就路網績效指標的角度而言，以往相關研究有以下看法：

## 1. 連續性指標

毛利（1981）所定義之連續性指標，係為用於評估地區內部網路，可否達成指定之路徑通往指定地點的條件。指標之計算，乃透過節點矩陣(Adjacency Matrix)求得。若圖形為簡單無向圖(Simple Undirected Graph)，則該矩陣元素為 $\{0,1\}$ ，來表示兩點間道路之無、有；透過下面之計算，即可獲致指標 $C = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{C_{ij}}{n(n+1)/2}$ 。

其中 $i$ 、 $j$ 為不同結點， $C_{ij}$ 為 $\{0,1\}$ ，分別表示 $(i,j)$ 間，其最短路徑的邊集合元素不皆為1、以及皆為1，兩種情形；由此可知，此指標之定義，在連接圖的初始值皆為1，而在路網有破壞情形下重新衡量，才會有所改變。另外，簡單無向圖的連續性指標的範圍，必落在 $0 \leq C \leq 1$ ；非簡單圖的連續性指標，才可能有 $C > 1$ 的情形發生。

葉、吳（1998）認為，該指標以行人步行的觀點來看時，若任意兩點間的路徑中，只要有一路段無法通行均視為 $C_{ij} = 0$ ，為其指標之缺失。因此將該指標改進以 $C'_{ij}$ 來取代其 $C_{ij}$ ：

$$C'_{ij} = \frac{\sum e'}{\sum e}$$

其中 $e = [\ln(p) \times \ln(m) \times \ln(v)] \times l$ ，用來反應由行人( $p$ )、機車( $m$ )、汽車( $v$ )三者組成之因素，以及路段長度 $l$ ，共同反應其對地區性道路行人步行連續性的影響。 $\sum e'$ 表示任意兩點之最短距離中，具備有「行人道路」條件之各路段 $e$ 值的總和； $\sum e$ 則表示任意兩點之最短距離中，全部「道路」之各路段 $e$ 值的總和。

連續性指標具有災前、災後做比較的特性，因此，該指標相當適於做災後的路網評估，可綜合判斷路網水準的改變程度。然而，若要在災前作為防災路網規劃的參考，則可建議模擬路段毀損的方式，方可判斷出各路段對整體路網的重要性。

## 2. 最短路徑變動比

陳、詹（1999）認為，災變損失的程度，與救援之反應時間、以及疏散的快慢有極為密切的關係，而救援與疏散的速度，則取決於動線系統的整體效能。透過路段通行成本計算之最短路徑組合，與部分路段因地震災害中斷所形成新的最短路徑組合之成本差距，即可作為評定路徑系統的整體效能的指標，即：

$$R = \frac{TC'_{ij} - TC_{ij}}{TC_{ij}} \times 100\%$$

其中 $TC_{ij}$ 表示從節點 $i$ 至節點 $j$ 的最短路徑旅行成本， $TC'_{ij}$ 表示某種路段毀損的情況下，節點 $i$ 至節點 $j$ 新的最短路徑旅行成本，用 $R$ 作為衡量指標，適用於評估兩個確立的節點之間，災後的路徑改變程度。

連續性指標與最短路徑變動比之間的差異，在於後者加入了路段權數特性的概念；同樣地，若要在災前作為防災路網規劃的參考，亦可透過模擬路段毀損的方式，方可判斷出各路段，對兩個確立點之間的重要性。

### 3. 路段關鍵性

徐、侯（2005）提出，災害期間救援機會高頻次地使用某些路段，這些相形重要的路段，稱之為關鍵路段。路段關鍵性，係在路網中，依照使用者所認定之服務種類，透過深度優先搜尋（Depth-First Search）的方法，計算任意兩點之間使用路段的各種可能性，求取相對所有可能情形下之比值，以用來判別通過各路段的機率。而路段通過機率達特定門檻值以上時，該路段即為關鍵路段。

關鍵路段的判別，可以綜觀整體路網的可靠性，若要在災前作為防災路網規劃的參考，亦可透過模擬路段毀損的方式，找出關鍵路段，並以平行邊的規劃方式，來降低其關鍵性。路網中，如所有路段的關鍵性降低，相對路網而言，可靠性就反而隨之提高。

#### 2.2.3 綜合評析

在防災道路的評估方面，評估法多採以指標擬定，並進行區域調查以反應路段實際狀況。而在避難救災功能評估指標方面，調查執行與量化過程有其模糊地帶。如建築物數量，以棟數或層數來衡量，精細度有所不同；此外，路段人口負荷雖極為重要，然而調查不易；高危險性路段，不易以量化方式描述；停車佔據道路長度與面積，並非固定不變；而道路落下物之物品與單位也多有不同。

而在於防災道路設計考量方面，如應符合的水準條件，過去相關研究則較少著墨。相較而言，防災據點之功能與定位，則明確詳盡許多。

至於在防災路網的評估，多採以災前、災後路網破壞的模擬狀況，來進行比較評估；此乃由於災害發生時間、地點不確定性所致。而災害模擬方法，多著墨於路段毀損之觀點，對於救災單位亦可能受災的情形則較少探討。

### 2.3 路網模型相關研究

圖論（Graph Theory）是探討路網模型的主要理論方法之一，圖形結構係由節點、節線兩種元素來組成，藉由節線權數的指定，可展現如成本、機率等各種概念；例如，路網就是屬於最常見的一種圖形。節線的權數在路網圖之中，可表示為路段長度、或旅行時間。路網圖在圖論中有許多實際應用的層面，本研究探討之防災路網模型，多採用圖論領域發展的相關理論。

#### 2.3.1 最短路徑

##### 1. Bellman-Ford演算法

Bellman-Ford演算法（Bellman，1958），乃計算權賦有向圖中，單一來源之最短路徑；並容許權數為負。所謂的最短路徑，即是在該賦權有向圖中， $u$ 到 $v$ 的所有路徑中，找到一條具有最小

權重總合的有向路徑  $P$ 。然而Bellman-Ford演算法的效率比Dijkstra演算法低，因此多用於圖形中包含有負權數的情況。

## 2. Dijkstra演算法

Dijkstra演算法 (Dijkstra, 1959) 乃由荷蘭的電腦科學家Dijkstra (1930-2002) 所提出。該演算法主要在解決非負權數邊 (nonnegative edge) 之有向圖 (directed graph) 中，單一來源的最短路徑問題。Dijkstra演算法先從起點開始，確定連接週邊節點的最短路徑後，在慢慢將範圍擴大，直到涵蓋所有節點的最短路徑。

## 3. Floyd-Warshall algorithm

Floyd-Warshall演算法 (Floyd, 1962; Warshall, 1962)，乃計算權賦有向圖中，所有配對之最短路徑問題。該演算法採用鄰點矩陣 (Adjacency Matrix) 為輸入項，以表示有向圖  $(V, E)$  中，邊的權數關係。該演算法允許權數為負的邊，然不允許路徑權數總和為負的圈 (cycle)，並可計算出每一對節點間，最小權數總和的路徑。

## 4. A\* 搜尋演算法

A\* 搜尋演算法 (Hart, Nilsson, and Raphael, 1968; Hart, Nilsson, and Raphael, 1972) 為最佳優先搜尋 (best-first search) 的方法之一。圖形中給定一個起點與一個終點，該演算法以啟發式的評價方法，對每個節點的最佳路徑評價予以排序，進而找出最小權數總和的路徑。

## 5. Johnson的演算法

Johnson的演算法 (Johnson, 1977)，乃在解決所有配對的最短路徑問題。該演算法運用了Bellman-Ford演算法來檢查負權數的圈，最後再結合Dijkstra演算法來求取解答。

### 2.3.2 繞徑問題

#### 1. 橋段問題

尋找橋段 (bridge) 問題 (Tarjan, 1974) 乃由Tarjan所提出。橋段之定義，即在圖形中，連接任意兩點間的所有路徑，都會使用到某路段；該路段即稱為橋段。

該理論為：路段  $(v, w)$  為圖形  $G$  之橋段，若且唯若  $v \rightarrow w$  在  $\vec{T}$  之內，而  $H(w) \leq w$ 、且  $L(w) > w - ND(w)$ 。

$\vec{T}$ ：有向根樹

$v \rightarrow w$ ：從節點  $v$  至節點  $w$  的路段  $(v, w)$

$v \rightarrow w$ ：從節點  $v$  至節點  $w$  的路徑

$v - -w$  : 圖形  $G$  中不包含於樹  $\vec{T}$  的路段  $(v, w)$

$v = \text{NUMBER}(v)$  : 節點數量

$ND(v)$  : 樹  $\vec{T}$  中節點  $v$  子代的節點數量 (包含  $v$  在內)

$$S(v) = \left\{ w \mid v \rightarrow w \right\} \cup \left\{ w \mid \exists u \left( v \rightarrow w \text{ and } u - -w \right) \right\}$$

$$H(v) = \max(S(v))$$

$$L(v) = \min(S(v))$$

橋段問題將是路網設計中相當重要的觀念，因為即使兩點間有許多連結彼此的路徑方式，然而一旦橋段被移除了，所有的路徑都將無法再連接此二節點。

## 2. 路段重要性問題

路段重要性 (most vital links) 的問題於1982年由Corley和Sha (1982) 提出，他們的問題為單一最重要路段問題。1983年，Malik論文提出了最短路徑之  $k$  最重要路段問題。而1989年時，Ball等人也發表了一般化的重要路段問題 (Ball, Golden, and Vohra, 1989)，在類似的條件下與Malik觀點呼應。而Malik則再次重申其論文之論點，並與Mittal、Gupta共同發表文章，以說明最短路徑之  $k$  最重要路段的演算法之複雜性 (Malik, and Mittal, 1989)；該文明確地說明決定  $k$  最重要路段之演算法機制，另外，也提供與Dijkstra演算法所費時間相同之演算法，來解決單一最重要路段之問題。

這些關於路段重要性的問題，具體說明路網或最短路徑中，若移除重要的路段，將造成特定兩點間最短距離之增量成本。

## 3. 繞路問題

Nardelli等人於1998年延伸了路段重要性之問題，直指出最短路徑上中臨界繞路路段 (detour-critical edge) 之特性。繞路問題點出了圖論中一個很特殊的觀點；所謂的繞路，即是發生在某一對點間之路線中，有任一路段無法通行時，如何從原路網中，找尋另一路段來連接此兩點的策略。而他們提出的演算法，乃在尋找出路段無法通行時，會造成此兩點間繞路最遠的情形。此外2001年時，Nardelli等人再次針對同樣的問題，改善了更快速的演算方法；並將原來 detour-critical edge 的用字，採取路段重要性理論的初期用詞，即最短路徑的最重要路段 (the most vital edge)。

下面則是所謂的繞路問題中，繞路、以及繞路臨界路段的正式定義。若  $P_G(r, s)$  表示無向圖  $G$  中，節點  $r$ 、和節點  $s$  之間的最短路徑，所謂在  $u$  節點的繞路 (detour)，其定義為： $u \in P_G(r, s) = \langle r, \dots, u, v, \dots, s \rangle$ ，最短路徑  $P_{G-e}(u, s)$  從  $u$  到  $s$  不使用到路段  $e = (u, v) \in P_G(r, s)$ 。而，繞路臨界路段之定義為： $e^*(r, s) = \max_{i=1, \dots, k} \{d_{G-e_i}(s_i, s) - d_G(s_i, s)\}$ 。

而繞路問題的目標，就是在找出路網之中， $e^*(r,s)$ 這些臨界路段。因為對原路網而言，這些路段的毀損，將會造成兩地之間，繞路過程，耗用最可觀的時間。

### 2.3.3 存活路網

圖論之中有一支研究方向，專注於探討容錯（**fault-tolerable**）路網模型，稱之為存活路網（**survivable network**）。最早有關存活路網演算法的知識，是源於1969年，由Steiglitz、Weiner、和Kleitman所提出。他們曾經應用該演算法來求解10個節點、和58個節點，連接度分別採用 $\{3,4,5,6\}$ 、和 $\{6\}$ ；但是該結果的最佳解並未流傳下來，因此無法評斷該演算法的效果（Grotschel et al, 1995）。

存活路網模型探討的範疇，為節點、節線權數固定的模型；若要探討權數含有機率概念的問題時，則要引用路網可靠度（**network reliability**）的計算，常見的路網可靠性問題，包含有： $k$ 場站可靠性（**k-terminal reliability**）、可及性（**Reachability**）、 $s,t$ 連結度（**s,t-connectedness**）等。以上該領域目前主要關注的幾個研究問題，在演算的複雜度上， $k$ 場站可靠性已被證實是NP困難（**NP-hard**）的問題、而可及性及連結度則是P完成（**P-complete**）的問題。

而就路段機率都固定的情形下，可靠度則為路徑排列組合的問題；然而，以本研究的觀點，所構建之防災路網，雖有路段、節點毀損之可能性，不過相對而言，那個「可能性」是多少？和哪些因素有關？要如何去估量？這些都必須要進一步以實驗方式驗證才具意義；因此在本研究中，進行路段權數固定的討論；相對的，直接認定防災路網應是被穩固的，或者是即使在某種規模、條件的破壞之下，仍保持其某種品質的連接性，方為更重要之觀點。

本研究在考量路網設計規劃時，採用權數固定模型，也就是存活路網之連結度（**connectivity**），來作為防災路網規劃的研究思考方向；以規劃的角度而言，存活路網模型提供了更好的思維，以解決路網遭受災害所面臨不確定性的問題。

存活路網為本研究防災路網模型的主要核心方法，存活路網所涵蓋的意義，相當多層次，因之正式的模式介紹，往後章節會有更詳細的定義。在此，則先用以下較口語的概念來說明。

存活路網中所謂的「存活」（**survivability**），即是指：發生重大災害事件，仍有能力復原路網服務之意（Grotschel et al., 1995）；而所謂的重大災害事件，在路網中即指：節線破壞、或節點破壞。而，具有存活性質的路網，即為存活路網。

存活路網在圖論中的另一種說法，即是 $k$ 連接圖（**k-connected graph, k-CON graph**）。圖 $G$ 的連接度（**connectivity**）寫做 $\kappa(G)$ ，若圖 $G$ 移除節點集合 $S$ ，可使得圖 $G-S$ 不連接、或只剩一個節點，則稱圖 $G$ 的連接度為 $\kappa(G) = |S|$ 。

$k$ 連接圖，隱含了兩種意義：第一種即為上述之定義，可以容許 $S-1$ 個節點的刪除，仍保持圖形連接；在此情形之下，該圖同樣會滿足第二種意義：容許 $S-1$ 個節線的刪除，仍保持圖形連接的情形，稱之為 $k$ 邊連接圖（**k-edge connected graph, k-ECON graph**）；但是，反之不亦然；也就是說， $k$ 邊連接圖未必是 $k$ 連接圖；因此也可知 $k$ 連接圖的形成條件高於 $k$ 邊連接圖。

## 1. 存活路網通式模型

存活路網通式模型，係由Grotschel（1995）等學者所提出。其方法乃是以「存活條件」使路網具有存活之特性；存活條件有兩種：邊存活條件、點存活條件。以下分別依照該學者所提出之定義分別說明之。

若圖形 $G$ 中的節點集合為 $V$ ，而 $V$ 中任一子集合 $W$ 的任一元素節點 $s$ ，與 $W \setminus V$ 集合中的任一元素節點 $t$ 之間，存在至少 $r_{st}$ （非負整數）條使用不同邊的路徑，則稱 $r_{st}$ 為圖形 $G$ 的「邊存活條件」。

採取與上述相同條件的類似觀點；若 $V$ 中的另一子集合 $Z$ 的尺寸為 $k_{st}$ （非負整數），而移除 $Z$ 會使得 $W - Z$ 的任一元素節點 $s$ ，與 $(W - Z) \setminus (V - Z)$ 集合中的任一元素節點 $t$ 之間，存在至少 $d_{st}$ （非負整數）條使用不同邊的路徑，則稱 $k_{st}$ 、 $d_{st}$ 為圖形 $G$ 的「點存活條件」。下面定義，即可說明路網之存活條件。其中， $con(W)$ 即為邊存活條件， $d(Z, W)$ 即為點存活條件。

$$con(W) := \max\{r_{st} \mid s \in W, t \in V \setminus W\} \quad (2-1)$$

$$d(Z, W) := \max\{d_{st} \mid s \in W \setminus Z, t \in V \setminus (Z \cup W)\} \text{ for } Z, W \subseteq V \quad (2-2)$$

若另引進一個整數 $x_{ij}$ ，來表示圖形 $G$ 中、邊集合 $E$ 的任一元素邊 $e_{ij}$ （節點 $i$ 、和節點 $j$ 之間的邊），則可將存活路網設計問題，轉化為下列線性規劃型式的限制條件：

$$\sum_{i \in W} \sum_{j \in V \setminus W} x_{ij} \geq con(W), \text{ for all } W \subseteq V, \phi \neq W \neq V \quad (2-3)$$

$$\sum_{i \in W} \sum_{j \in V \setminus (Z \cup W)} x_{ij} \geq d(Z, W), \text{ for all eligible } (Z, W) \text{ of subsets of } V \quad (2-4)$$

$$0 \leq x_{ij} \leq 1, \text{ for all } ij \in E \quad (2-5)$$

$$x_{ij} \text{ integral, for all } ij \in E \quad (2-6)$$

其中，第一類限制式（2-3），即在滿足「邊存活條件」；第二類限制式（2-4），即在滿足「點存活條件」；第三、第四類型為零一限制式（2-5）、（2-6）。此即存活路網通式模型。這個存活路網的通式模型，囊括了：第一類限制式的 $k$ 邊連接圖（ $k$ -edge connected graph， $k$ -ECON圖）、第二類限制式的 $k$ 點連接圖（ $k$ -connected graph， $k$ -NCON圖）兩種存活路網；而需要同時考量兩種限制式的圖形，實務上則較少見。預期道路（節線）遭受地震破壞之觀點，防災路網之替代路徑之建構，則可採以 $k$ 邊連接圖之路網模型。



## 2. 二邊連接模型

具有替代路徑之路網模型，其中當以二邊連接模型為所有  $k$  邊連接模型中，成本最小之模型。二邊連接模型是存活路網模型中的一個特例；一個路網圖如果提供了一種保障，即在除去任意一邊的情況下，此路網圖仍然相互連接，則稱這個路網為二邊連接（two-edge connected，簡稱 2ECON）。這個名字的由來，是因為至少要移除兩個以上的邊，才會使得該路網被一分為二，因而獲得此稱號。二邊連接模型如下：

$$\sum_{i \in W} \sum_{j \in V \setminus W} x_{ij} \geq 2, \text{ for all } W \subseteq V, \phi \neq W \neq V \quad (2-7)$$

$$0 \leq x_{ij} \leq 1, \text{ for all } ij \in E \quad (2-8)$$

$$x_{ij} \text{ integral, for all } ij \in E \quad (2-9)$$

此即刪除存活路網模型的第二類限制式，並令  $r_{st} = 2$ 。而由此模式亦可知，邊的存活度愈高，則存活路網模型第二類限制式的  $r_{st}$  就要愈大；此即代表替代路徑的數目就愈多，然而路網結構之成本也就愈形可觀。

## 3. 存活路網相關演算法

存活路網模型，是基於基本的圖論演算法發展而來的接續性的問題，相關基礎的演算法，多採用啟發式解法。包含以下幾種：

（1）貪婪耳構建法：耳（ear）是指圖形中一種最大的路徑（maximal path），其中內部點的度（degree of vertex）皆為2。耳解構（ear decomposition）則是指：組成集合  $P_0, \dots, P_k$  中， $P_0$  是一個圈， $P_i$  是  $P_0 \cup \dots \cup P_{i-1}$  中的一個耳，其中  $i \geq 1$ 。二連接圖和耳解構有相當密切的關係；事實上，一個圖形要是為二連接圖，若且為若該圖有耳解構（Whitney, 1932）。因此，要建立最少邊的二連接圖（edge-minimal 2CON Graph），可由耳解構的程序來建立（Lovasz & Plummer, 1986）。貪婪耳構建法（greedy ears construction heuristic）簡要來說，即是利用二連接圖的耳解構特性，首先建立一個包含欲要點（desired nodes）圈，再重複尋找可以涵蓋欲要點的耳，直到所有欲要點皆被包含為止，即完成一個可行的二連接圖。

（2）隨機鬆散構建法：隨機鬆散構建法（random sparse construction heuristic），首先係找出特殊點的任一子集合，並找出該集合的深度優先樹，透過隨機尋找一個不存在於該樹之中的邊，必能建立一個圈；其次，再隨機加入含有特殊點的耳，直到所有的特殊點皆被納入2CON圖中。然而，由於隨機鬆散建構法並不考量成本資訊，因此計算所得之結果，未必是成本最小的解。

（3）交換法：交換法（interchange heuristic）包含：一最佳（one-optimal）交換法、二最佳（two-optimal）交換法、以及三最佳（three-optimal）交換法。任何一個2CON圖，至少會包含一個圈；而若由該圈中相同節點，替換某些邊集合，來產生新的圈，也將會是一種可行解。這個概念，即是交換法的基礎。一最佳交換法嘗試以不在圈中

的一個邊，來替換原圈中的一個邊，達到更低成本的目的；類似的觀念，二最佳交換法則嘗試同時交換二個邊，來達到相同的目的；同理，三交換法則是同時交換三個邊。

(4) 樹法：樹法 (tree heuristic) 將邊的零壹變數限制鬆綁，並欲以提高邊的連接度來達成連接圖的目標，亦即邊滿足  $x(\delta(W)) \geq r(W)$  的條件 (Goemans & Bertsimas, 1993)； $x_e \geq 2$  可以解釋為邊  $e$  使用了  $x_e$  次，以達成提高連接度之目的。

#### 2.3.4 綜合評析

路網模型相關理論主題豐富，無論是從概念、或落實到各種演算方法，皆相當獨立而完整。然而，本研究欲關注的幾種路網模型之結合，例如在存活路網模型中來考量繞路問題；例如繞路問題中，同時關注多個需求點，而非僅單一供給對；則過去的文獻則較少著墨。此外，以責任分區做為節點單位，來探討點連接度的觀念，也屬獨特而突破之想法；因此，尚未尋獲相關文獻可供參考。

## 第三章 防災路網分析

### 3.1 問題陳述

防災路網有許多待釐清與解決的問題。由於現存的路網存在著多功能的使用，日常交通與緊急防救交通動線重疊，致使防災路網扮演的角色定位容易混淆；然而，由防救功能之特殊性，方可察覺對於路網的特殊需求，應有別於一般交通旅次，以下分別描述防災路網規劃時所面臨的問題。

#### 3.1.1 現況問題

由於災害的發生並非日常性，在沒有危機處理意識的情況下，防災的觀念並不是相當普遍，加上一般災害發生時，多數仍能將就臨場應對，因此不易明確區別防災道路與一般道路之間的差異。然而由道路設計的角度來看，便可以發現一般道路多以交通觀點為主要思考，而較缺乏防災觀念。因此可以說目前並沒有真正的防災道路，而僅能說是使用交通道路來作為防災活動之使用。防災道路之意義，應在於額外提供救災/避難活動所需之所需道路設備條件與適當路線，以提高救災/避難活動進行之順暢；並且對於道路使用者而言，災害發生與未發生之間的感受差異降到最低，達成使用者仍能進行日常性交通活動之目標。以下為本研究蒐集並觀察到之相關問題：

#### 1. 道路使用困難

(1) 直接災害：道路的直接災害，包括橋樑斷裂、道路隆起、道路下陷、路面破裂等，造成車輛無法順利通行。

(2) 間接災害：發生地震災害，陸橋、建築物、高架道路倒塌阻絕道路之二次災害，會造成道路無法使用之情況。此外，道路有安全風險；例如發生地震，住宅週邊道路狹窄，使民眾有危險感，甚至道路上方之廣告、盆栽、危險物品等，造成道路環境不安全。

(3) 無專用設計：救援功能無法發揮。例如發生火災，受災地之大樓本身不具備有完善消防系統，道路所提供之消防系統與受災地相距過長，造成水壓不足，無法發揮功能。尺度較大的救援車輛，例如雲梯車，災點周圍之道路無法提供合適的運作空間環境。

(4) 多功使用：路邊停車問題使道路容量相對減少；而發生災害時，無關的民眾與媒體，多無法與救災人員配合，導致災害地點事件周圍旅次與交通量提高，而使道路容易被佔用；而災害中期，可能會發生民眾臨時搭用帳棚而佔用道路的情況；此外，救護路線與其他旅次目的之交通路線混合，可能形成運行阻礙。

## 2. 災害對象主體的不確定性

防災路網針對的對象即為災害，然而因為無法預測災害發生的時間、空間，以致規劃路線的著力點容易失焦。此外，需求地點不明確，由於無法預測災害發生的時間，而原先規劃的供給單位，亦有可能成為受援對象；同樣地，需求對象，則未必在每一次的災害之中，都具有需求性質。

## 3. 沒有絕對穩固的路網

無法保證原先所規劃之路網可以被使用，地震災害有可能直接、間接破壞路網設施，而造成道路無法使用之情況；因此，規劃的路網，無法提供完全的保障。

## 4. 路網結構功能定位不明確

由於沒有獨立的路網，災情發生時，救護、避難、物資輸送等運輸旅次沒有較獨立的路線，與其他旅次目的之交通混合在一起，無法相互配合；因此，會降低各種旅次運輸的效率。

## 5. 其它相關問題

(1) 缺乏危機意識之建立：由於災害的發生並非日常性的，在沒有危機處理意識的情況下，防災的觀念並不是相當普遍；另外，一般災害發生時，多數道路仍能勉強臨場應對，因此，對道路的一般日常使用，使得防災道路動線的啟動，未必所有的使用者都清楚；因而，要達到專線專用的程度，需要緩衝時間，所以並無法立即發揮應有功能。

(2) 缺乏路網即時資訊之蒐集：地震監測網的密度、蒐集情報的種類、與適當處理為有用的資訊，都是控管路網所需情資的基礎，如何使決策者處在情資充裕的環境，左右著路網最佳使用的結果。

(3) 缺乏完善防災道路之設計準則：由道路設計的角度來看，一般道路以交通觀點思考，較缺乏防災觀念。而造成此問題的另外一個主要因素，則是因為防災道路設計標準未明，因此對於欲落實規劃建設防災路網的相關單位，也不知從何進行。

### 3.1.2 課題探討

如欲針對上述問題提出解決方案，首先須確立防災路網之樣態並指定其為防災路網，方能進一步在該路網上進行硬體設施設備擴充、措施制度控管，而確立此路網之所在即為本研究之重點項目。本研究主要研究目標，乃提出一個在地震災害時期，能提供地震災害相關運輸活動順利進行之路網結構。而這樣的路網，以四個層面考慮：一是如何將路網因災害不確定所產生不可靠的特性納入解決；二是如何以有限的救援資源使路網能合理地全面涵蓋可能的需求區域；三是如何提供救援單位至需求區域最快捷的路網結構；四是如何確保路網道路空間的安全性。在以上四個層面中，由於災害特性之故，首當重視路網的

可靠特性。因此，本研究探討的防災路網模型，首重路網的可靠性；其次，確立建立未知需求區域的原則，並在以快捷、層級等概念，使防災路網具備高效率、易運作的特性。

### 3.2 地震與防災路網

地震為一種不確定性的現象，目前尚未有精準的系統，可以正確地預測地震。而所謂的不確定性，係指：發生時間、發生地點、發生規模等的不確定。由於這些不確定因素，也造成了災害的不確定性；亦即：災害發生時間、災害發生地點、以及災害發生規模等。

地震時間不確定性所造成的災害，在白天和夜晚有很大的差異，主要是由於日間人口、夜間人口，其活動地理分布不同的關係。日間活動人口有聚集的特性，例如中心商業區，高密度、集中的地點，而夜間活動人口則有分散的特性，例如市郊住宅區，低密度、散佈的區域。地震發生的地點，範圍可以斷層帶、地震區來概分；但，關注的焦點，則在於災害發生的地點，也就是人口所在之處；此與災害發生的時間，關係密切，時間與地點的組合，會形成災害不同的特性；日間人口集中在商業地點，同時住宅地區人口密度較為下降；反之，夜間人口集中在住宅地區，同時商業地點人口密度較為下降。地震規模，慣用以1935年美國地震學家芮克特所發明的芮氏地震規模來衡量，並以對數尺度表現，每一點代表著十倍的地表運動。而地震發生則以機率概念表示；小規模的地震，發生機率高，反之，大規模的地震，發生機率低。根據芮氏地震規模的算法，六點零的地震可以造成嚴重損害，七點零則屬於重大地震，足以造成廣泛而嚴重的傷害。不過，芮氏地震規模最大的地震，不見得是造成人員及財物損傷最大的地震，此與地震發生的時間、地點有關。

建構防災路網的目的，也是在於降低不確定性。即透過一些路網規劃的手法，讓不確定的情形，落在規劃者的預期範圍之內。地震仍是不確定的，不過災害的不確定情況，則可因路網規劃之多角化的考量，而有效掌控。

防災路網的基本元素為防災道路，本研究將其定義為：地震災害時期，一個環境安全、設施充足，並能提供地震災害相關運輸活動進行之道路。而防災路網，則定義為：地震災害時期，一個可靠、全面、快捷，並能提供地震災害相關運輸活動進行之防災道路系統。其中，所謂地震災害時期，可依地震發生的時間特性定義，並區分為四個階段：準備時期、發生時期、應變時期、穩定時期。地震發生的時間點之前，皆為準備時期的階段；地震發生後，即進入發生時期，並開始進行系列的震災相關運輸活動，亦即應變時期；當活動之進行跳脫紊亂，並明顯受到掌控，則進入穩定時期；穩定時期中，當防災路網的狀況條件，趨近於上次準備時期的狀況條件時，即進入下階段的準備時期。因此，由本研究所界定的地震災害時期，係指一種長期、連續無間斷的時間概念；換言之，即使在任一看似與地震無關的時間點，都屬於地震災害時期的範疇（準備時期）。

防災路網之可靠性，與其他路網有很大的差異。防災路網是一種破壞預期的路網。地震災害的現象，會反映在路網直接、間接破壞的結果，然而路網實質的破壞，未必直指路網功能的失敗，此即可靠的概念。透過路網的規劃設計，使之在一定程度的破壞下，網路功能仍可運作順暢。防災路網之全面性，乃在於其屬道路建設的範疇，應以政府立場來予以規劃、落實。而由於地震災害的不確定性，無法預測其發生

地點的特性，防災路網所觸及的角落，應完整而全面，以達社會公平的期待。防災路網之快捷性，於地震災害發生時，為發揮路網功能、檢驗路網功能成敗的關鍵時刻。在這個階段中，時間最為寶貴，社會損失、成本隨著時間流逝而快速提高，因而路網快捷的概念，成為各地震災害相關運輸活動，立即體驗路網效果的第一感受。

地震災害會造成交通供給、需求量，在短時間內劇烈變化，這種異於一般日常交通的現象，應有良好的管理機制來予以控制，方能提高運輸效率，進而順暢地完成立即的避難、救援、後續的物資輸送、以及盡速讓日常交通回到正軌；然而若有良好的路網架構，則管理機制的運作便能更有效的實行。

地震災害相關的運輸活動，包括有：避難、救援、物資輸送。而此三種運輸活動，又皆可細分為兩種類型：系統性（可規劃）、非系統性（不可規劃）。所謂的避難，可分為臨時避難、階段避難；臨時避難為人群立即、非系統性的反應，故就範圍尺度而言，屬於步行距離的範疇。階段避難係指經過規劃安排、系統性的作為，故就範圍尺度而言，除了步行距離的範疇，亦包含了車輛距離的範疇。救援、物資輸送，以車輛距離的範疇為主，但亦可輔以昇昇機、飛機、船舶距離的範疇。一般而言，救援、物資輸送皆可屬於系統性的運輸活動，然而必須在資訊充裕、可互相協調、並有主導單位的情況下，方可稱上系統性的行為。防災路網，就地震災害相關的運輸活動而言，屬於車輛距離的範疇、並以主要支援系統性活動、次要輔助非系統性運輸活動之路網。

### 3.3 防災路網系統之組成

路網係由路口、路段兩種基本元素所組成；而路網之防災特性，則表現於供給點與需求點之關係。

依據防災供給單位、防災據點之特性，本研究將防災路網區分為兩種尺度來探討防災路網：局部地區、全域地區。局部地區乃以單一供給點為中心，藉以發展局部有效率的防災路網系統；全域地區乃探討各區相互救援的模式，增加防災能力，因而發展出全域的防災路網結構。

考量防災路網型態配置之理由在於：1.現實供給單位具有區域考量；2.透過設施備援方式來提高供給的可靠性。以圖3.1為例，全域地區共可分為甲、乙、丙三個局部地區；每一區內各自發展其防災道路系統，而其中之防災道路，都應具備一個環境安全、設施充足，並能提供地震災害相關運輸活動進行之道路條件。因此，在本研究的「第四章 防災道路評估」中，將提出防震觀點下，建立合宜的道路空間品質架構與模式，以作為評估現有道路狀況之方法，進而可以優先考量作為防災道路路段。

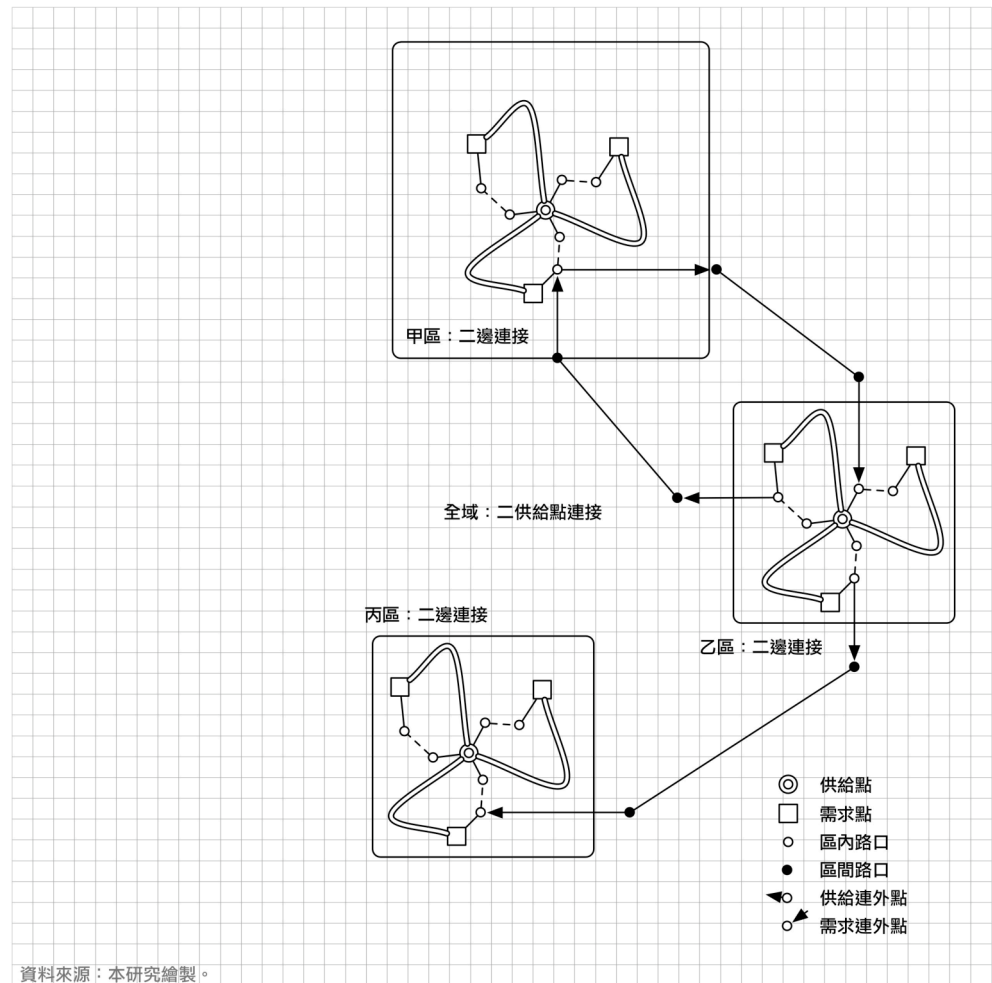


圖3.1 防災路網系統之組成

其次，在路網結構方面，面臨地震災害所帶來的種種不確定因素，如未知的破壞路段地點、未知的不足供給設施，本研究認為應找出特殊路網結構的設計方式，應掌握及降低不確定性之衝擊。此外，合宜範圍範圍的形成之法，也在「第五章 防災路網建構」中詳加探討。

### 3.4 防災路網系統之功能

防災道路的功能，本研究採用供給面、需求面的方式進行探討。防災道路為救援活動的媒介通道，應配合救援供給的消防、警察、醫療、物資等單位，來連接受災需求的地區，方能發揮最大效用。

由於災害不確定性的因素很多，而實際的災害情況，在災害尚未發生之前，也無從準確判斷。因此，災情之情境推演，就顯得非常重要。雖然情境無法窮舉，然而面對有限的供給設施與固定的路網結構，合理運用的規劃方式應有跡可循；若能再配合擬定多個可能發生的主要情境，則遭遇災害時，便可套入類似的情境之中。憑藉著合理規劃的防災路網以發揮其功能性，即可快速因應災情、並有效率地降低社會成本損失。

### 3.4.1 需求面

在需求方面，由於災害發生時刻，真確的避難行為不易預測；初期的避難行為混亂，難以掌控避難人群並進行規劃。應於平時培養民眾之防災觀念，並提供安全的步行避難路線空間，以供隨機的緊急避難人流使用；民眾依其自主性，以反應其避難行為。

然而，混亂的避難行為在階段避難過程中，便可受到掌控而成為系統性的行為。階段避難，係為由分散的臨時避難場所集中至大型收容場所之過程。此乃經過規劃之程序，並可由步行方式與車輛運送方式同時進行。

### 3.4.2 供給面

在供給方面，災害發生時，包含了消防、警察、醫療、物資等明確的救援單位，收到充分的訊息後，即可進行系統性的規劃措施。救援行動與隨機的避難行為不同，任何一趟救援旅次，無論規模大小，皆是在特定目標下的結果。單位內的獨立行動，以各單位之目標邁進；而單位間的合作行動，則以全面資源考量，以系統性之目標邁進。

### 3.4.3 可控制的路線系統

一般道路網應區分為：無控制的路線系統、與可控制的路線系統兩者。若現存的路網結構為無控制的路線系統，則各救援單位依據單一小規模災害事件，各自單獨進行路網運用。若現存的路網結構隱含可控制的路線系統，則所有救援單位可遵循全面性的整體規劃，以系統性地有效運用路網，來因應大規模的災害事件。

由於災害發生地點的不確定性，使得真實災害的情況中，需求地點成為未知；而救援的對象無法確立，則救援路線的規劃便無端點。而若將整個城市中，所有的地點皆視為可能的受災害地，固然理想，但也將導致路網規劃過於綿密而成本極高的情形。

防災路網應為可控制的路線系統。防災路網之規劃目的，乃在提供需求、供給雙方面之最佳連結路網。由圖3.2之供需架構分析，防災路網結構規劃需先確立路網的服務對象。供給單位因為功能明確，位置所在相對容易確立；然而需求地點，則可能散佈整個城市之中；根據以往的相關研究，防災據點多為明確、且可供中長期避難之場所，包括公園、綠地、學校等可作為收容之場所。因此，防災路網應提供這些場所與供給單位之間的連結道路。包括了：（1）消防／警察單位與避難／收容場所之間的路網系統；（2）醫療／物資單位與避難／收容場所之間的路網系統。



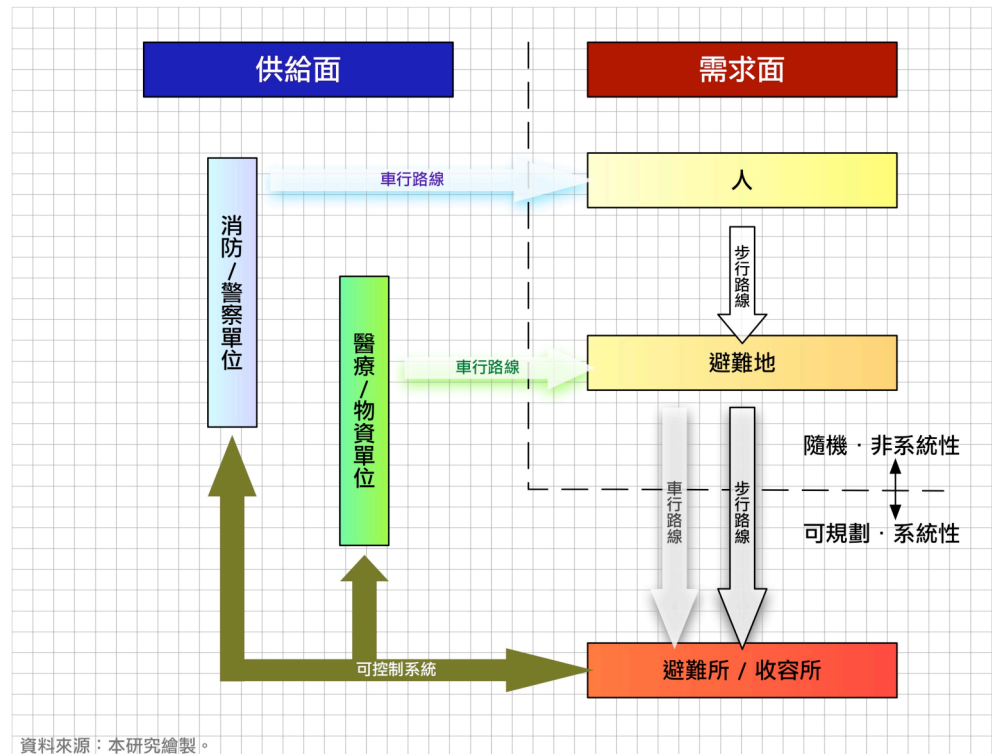


圖3.2 防災路網之供需架構

### 3.5 防災路網系統之架構

#### 3.5.1 功能架構

防災路網連結供給面之救援單位與需求面之避難據點，因此防災路網的結構，與這些功能單位、據點緊密相關；亦成為基礎路網篩選為防災路網的準則。在消防、醫療方面，以快捷性為主要考量準則。在物資、避難據點方面，則以全面性為主要考量準則。而原本的路網結構，則以道路關鍵性、空間性等準則，用來作為篩選出可能的防災路網結構。

圖3.3顯示，防災路網將路網系統，緊密與消防、醫療、物資等功能結合，並與避難據點連結。而這些據點，消防方面，包含了消防隊、警局；醫療方面，包含了醫學中心、地區醫院等；外來物資方面，包含了機場、港口、或聯外道路；避難據點方面，包含了綠地公園、學校等。

關鍵性、空間性將於第四章詳述其指標與準則；而全面性、快捷性則於「第五章 防災路網建構」，會有更深入的討論。防災路網之功能架構，依照相關準則，可篩選出重要、條件優良之道路，以作為車輛觀點之防災路網的候選路網結構。

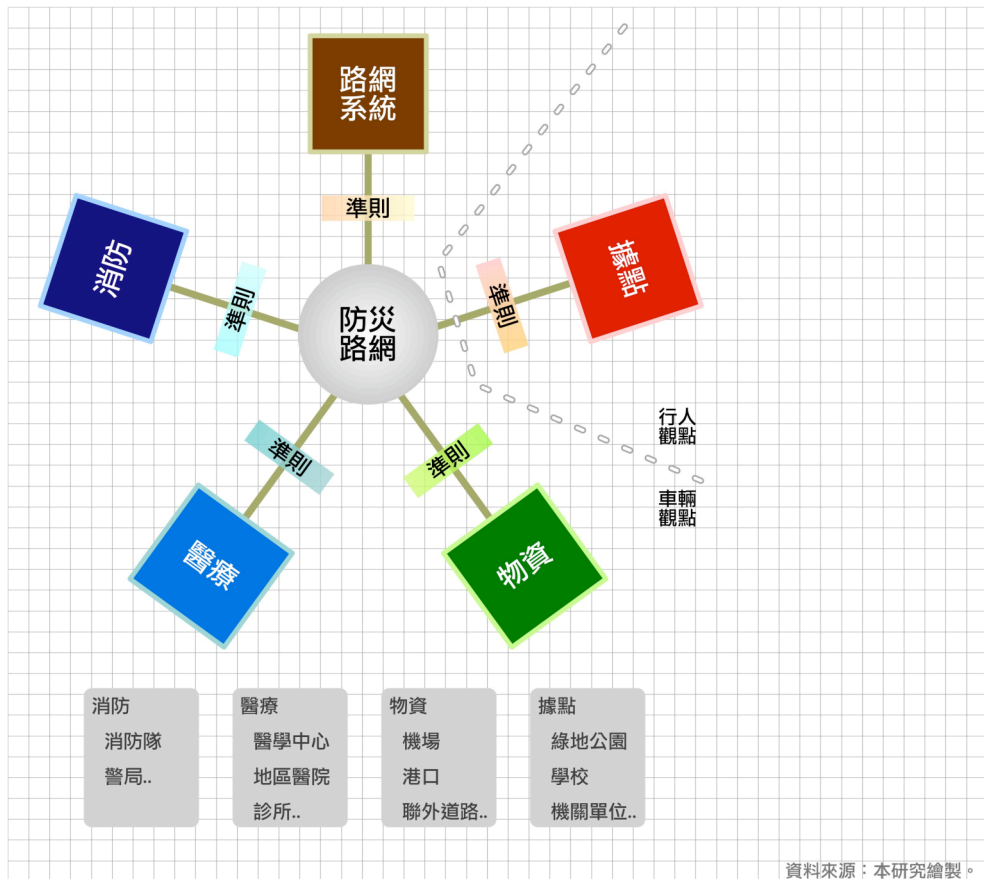


圖3.3 防災路網之功能架構

### 3.5.2 系統架構

本研究發展之防災路網系統架構，共分為「防災道路評估」與「防災路網構建」兩個階段，及「道路關鍵評估」、「道路空間評估」、「責任分區路網模型」、「系統繞路路網模型」、「互援路網模型」五個模組，以求得防災路網結構。最後並發展「防災路網評估」模組，以提供不同防災路網間的衡量比較基準（圖3.4）。

第一階段「防災道路評估」從現存路網中，評估道路環境條件較好及道路關鍵性較高之路段，以篩選作為第二階段輸入的基礎路網。第一階段以道路段為對象，先分析道路與相關據點之關係，再透過「道路關鍵評估」模組來衡量路段之關鍵性，並進行「道路空間評估」模組來衡量路段之空間性，以作為防災道路之綜合評估。

第二階段「防災路網構建」從篩選路網中，構建路網可靠性、快捷性、全面性之結構，以決定做為防災路網之最終結果。第二階段以道路網為對象，透過「責任分區路網模型」模組探討供給單位間的最佳界線；運用「系統繞路路網模型」模組來提高路網之可靠性，以解決路網遭受地震破壞的問題；使用「互援路網模型」模組來消彌巨觀路網觀點下，救援單位供給量能不足的情形。

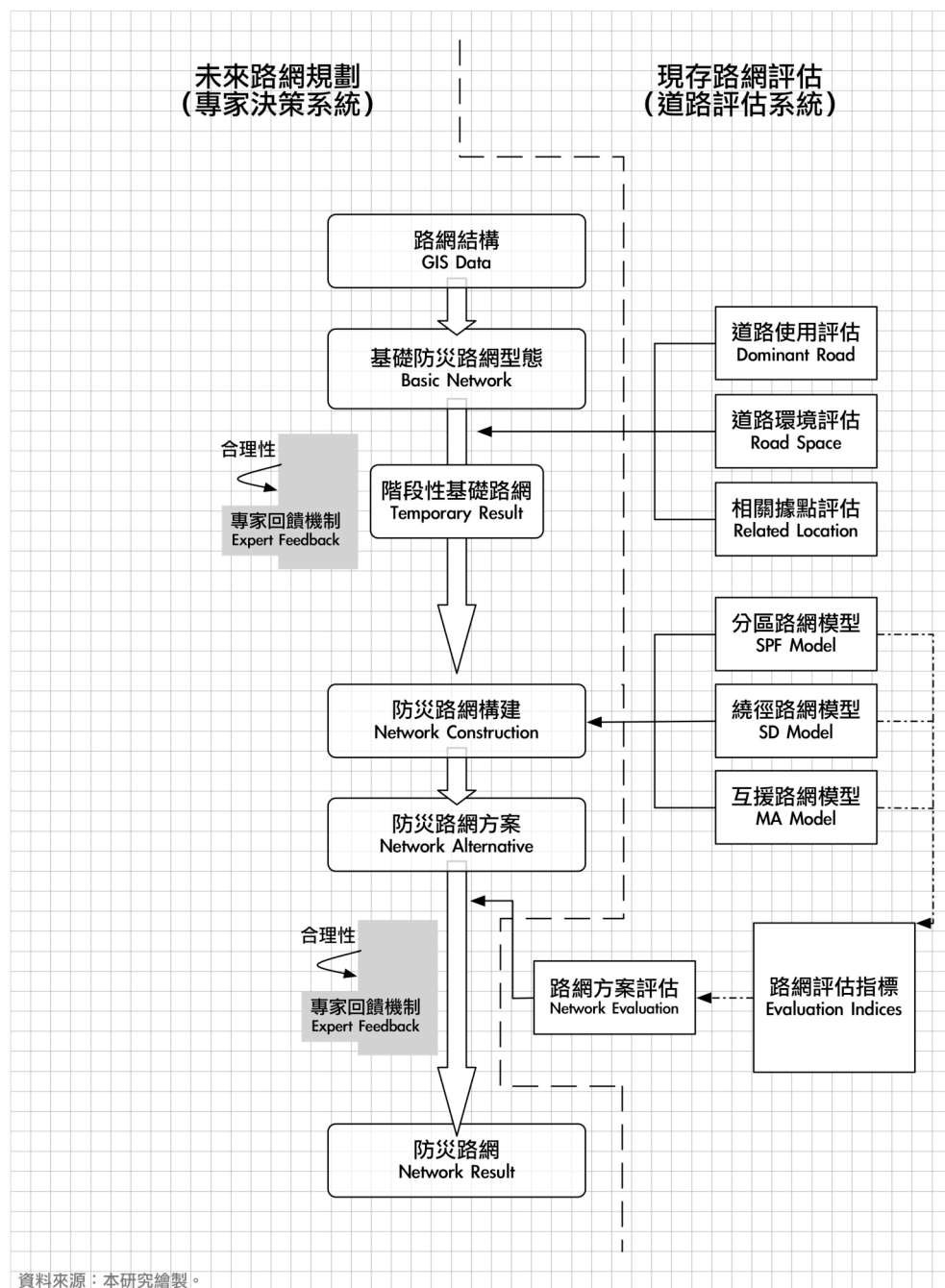


圖3.4 防災路網之系統架構



## 第四章 防災道路評估

### 4.1 防災道路關鍵性評估

關鍵性之防災道路，係為災時運輸旅次運用最頻繁之道路；關鍵性之原因，乃在於使用率頻繁之道路若受災而發生問題，則對救災活動影響甚鉅。然隨著災害時點遷變，防災道路功能也逐漸轉換，使用頻次也將隨之不同，防災道路對於各種救災活動之關鍵性，則非全然相同；為深入了解防災道路對救災運輸所造成之影響，因而須有一評估法來衡量此關鍵性。

#### 4.1.1 評估方法

本研究採用之關鍵性評估法，係先模擬各種防災緊急活動之車輛運行可能於現存路網中行經之路線，用以判別在防災路網中，被使用頻率較高的道路；這些道路路段於規劃防災路網時，將給予較高之優先性。

#### 4.1.2 路網規模

若在供需全面連結的情況下，以快捷性、成本性、替代性面向說明，將防災路網之規劃分為三種規模：初級規模、二級規模、與三級規模。

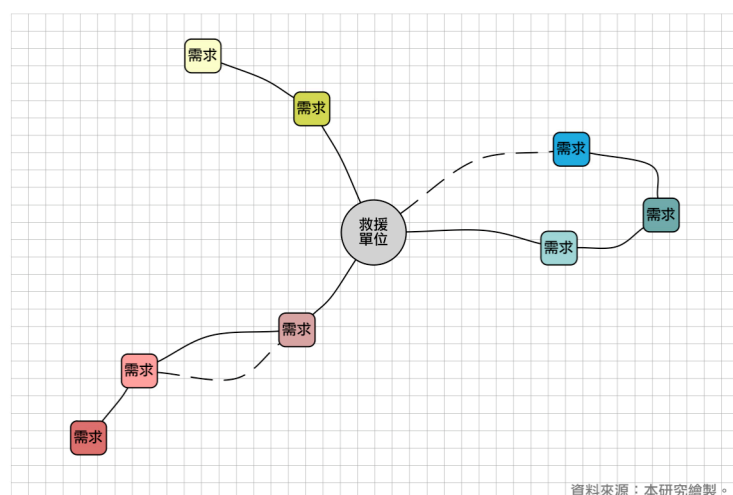


圖4.1 防災路網之設計規模

防災路網之初級規模，為全面、快捷、低成本、無替代性串接供需單位之路網，倘若以路線距離作為快捷的衡量指標，這樣的路網結構可以最短路徑樹來說明；而此路網結構的特性，各供需單位之間的可能行經路線，皆因無替代路線的緣故而僅為固定一種。防災路網之二級規模，為初級規模之部份擴展，特定供需單位之間包含替代路線以增加行經路線的多樣化，因此具有全面、快捷、中成本、中替代性連接供需單位之特性；若某緊急路段遭受破壞，則有機會採行其他路段所形成之緊急路線替代之。防災路網之三級規模，為初級規模之全面擴展，所有供需單位之間包含替代路線以增加行經路線的多樣化，因此為全面、快捷、高成本、高替代性連接供需單位之路網；若某緊急路段遭受破壞，則必有其他路段形成之緊急路線可替代之。

#### 4.1.3 評估指標

本研究之關鍵性評估指標分為四項：消防救援關鍵性指標、醫療救護關鍵性指標、物資輸送關鍵性指標、以及整體防災關鍵性指標。

通用指標定義可以下式表式：

$$P_a(e_i) = \frac{n_{a, e_i}}{n_{a, p}} \quad (4-1)$$

$a$ ：緊急活動類型，包含火災救援、醫療救護、以及物資輸送。

$n_{a, e_i}$ ：在緊急活動類型  $a$  之中，節線  $e_i$  被經過的次數。

$n_{a, p}$ ：在緊急活動類型  $a$  之中，供需之間所有的路線數目。

$P_a(e_i)$ ：在緊急活動類型  $a$  之中，節線  $e_i$  被經過的機率。

#### 4.1.4 指標演算

指標演算的過程，主要可分為以下主要步驟。首先為初始化動作，須先確立所有的供給單位、需求單位之地理位置，以及研究對象地區之路網結構；並從這樣的條件下，來探討供給地點至需求地點所可能行經的路線。假設從其中的供給單位集合、需求單位集合，各任取一單位作為災時可能形成的供需對，判斷此供需應歸屬於火災救援、醫療救護、或物資輸送其中之一，進行此供需對所形成到達兩地之路線方式，同樣之法，運用於同類型之供需對，進行演算之後便可獲致該緊急活動性質之關鍵性指標。

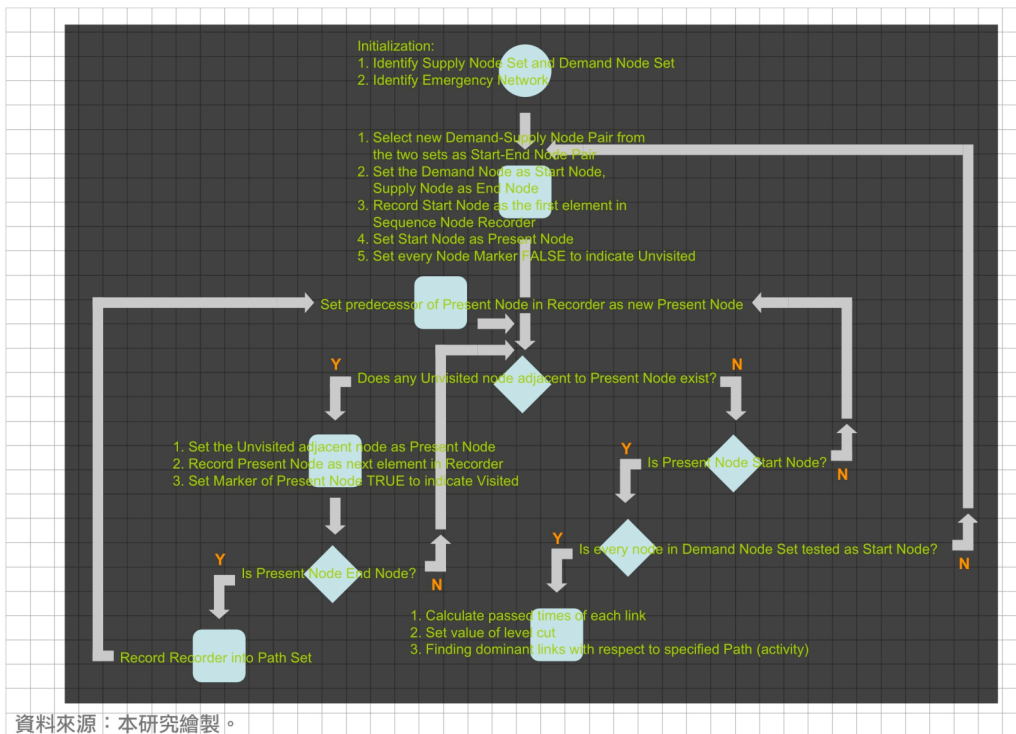


圖4.2 關鍵性指標演算流程

供需對之間的路線產生過程，係設定一當下點，作為路網中之巡遊者，其目標乃在尋找所有連結當下供需對之路線；假若路網中所有的節點皆被巡遊/拜訪過，則便完成該供需對之間所有路線之探尋；此後便設立新的供需對並重複相同過程，直至所有的供需對進行結束，如圖4.2所示。透過節點順序器之路線集合，針對同緊急活動性質之所有供需對，各種路線方式皆被掌握，而關鍵性指標值最終也可獲得。

## 4.2 防災道路空間性評估

目前在許多國家，例如日本與台灣等，均以道路寬度作為防災道路之分類與指定之依據。然而，道路空間的組成，並非僅為路寬之一維向度；道路之完整的空間結構體，應從三維向度評估，將能更貼近實際的道路空間狀況。因此如能加入其他衡量因子，以取代道路寬度單一指標，在未來指定防災道路的規劃過程，其空間結構則確保提供較佳的防災品質、並值得被信賴且作為防災道路。因此在空間的評估上，本節透過解構道路空間，以三維度及震災的角度來分析道路空間之品質，發展了多個指標以描述道路防災空間之性質，以表示道路路段防震效應之品質。透過道路之空間分析，這些因子以量化的方式經由專家意見的加權處理，可被計算成組成一綜合評量值，藉此進一步對路段的防災空間品質深入了解並簡化呈現，以供未來防災道路規劃之參考。

### 4.2.1 評量架構

眾多因素會道路空間的品質，地震發生後道路的狀態有可能會改變，而救援機具就須要隨著選擇合適的救援路線。過去的震災經驗顯示，大型救援機具，如消防車，很難通行一些狀況不良的路段，因而無法接近受災地點。根據內政部建築研究所的都市防災調查研究報告，九二一地震導致許多建築物倒塌，間接使得某部份的道路被阻隔，這些事件也揭露出道路周圍空間的重要性。對此，本研究將道路周圍的空間細分為多個組成空間，來探討各部份空間對道路直接、間接的影響。

首先，道路周圍的空間之劃定，可依實質三維空間位置特性，劃分為道路空間和近鄰空間，近鄰空間可再細分為上部空間、側部空間，如圖4.3所示。道路空間係為車輛運行所在之空間。道路寬度基本上決定了道路空間的規模大小，過去的許多文獻也指出道路寬度是影響緊急運輸的重要因素之一。道路空間應保持暢通，並避免潛在的物體影響道路空間的品質，例如上部空間之陸橋及電線塌落、側部空間建築物傾倒等，都有可能間接影響道路有效空間，因而規劃震災路網，如能先行確認各路段之空間品質，對於防災路網之規劃，有正面積極的意義。

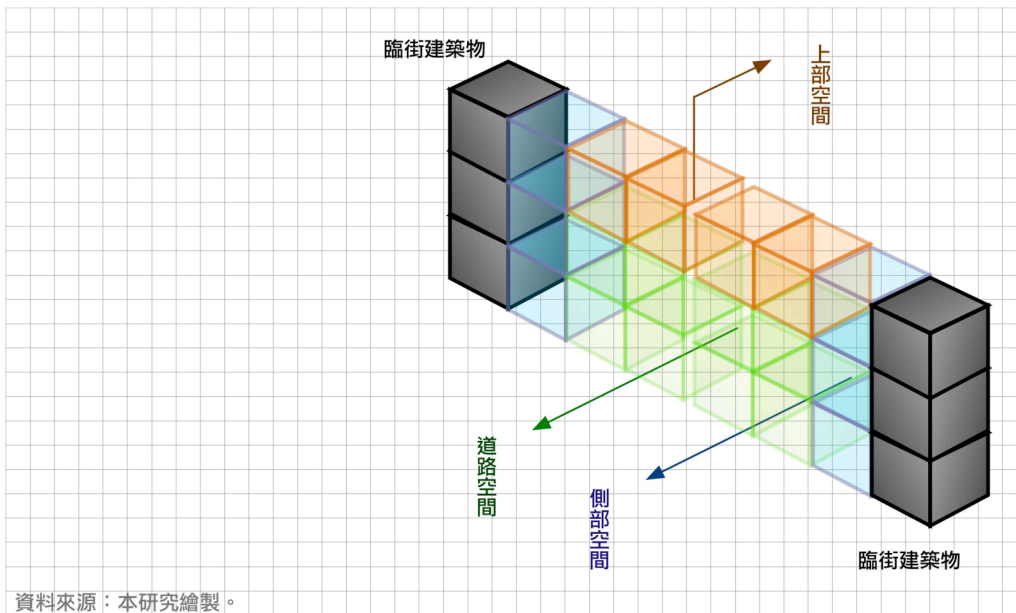


圖4.3 道路空間解構

從道路周圍空間區塊的角度來分析，可以區分為：

### 1. 道路空間

道路空間係指道路面上方，車輛運行之空間。因此道路空間關係著緊急運具能否運行順利之關鍵，狹窄的道路將不利於運具之運作。

### 2. 近鄰空間

近鄰空間係指道路空間周遭之空間。可再細分為上部空間、側部空間。

(1) 上部空間：上部空間位於道路車輛運行空間之正上方；此空間之物體，包括有陸橋、高架橋、電線等。

(2) 側部空間：側部空間位於道路車輛運行空間之兩側，此空間範圍包含人行道、建築物退縮空間等。存在於側部空間的物體，包含廣告招牌、建築物本身。

#### 4.2.2 道路空間評估指標

根據以往相關研究，影響道路防災空間品質之因素，可包括道路寬度、路邊停車格位面積、道路活動人口、道路交通量、道路地下管線、地層地盤、臨街建築物高度、臨街建築物數量、臨街建築物外牆與結構、臨街建築物屋齡、高架橋、陸橋、路側人行道行人流、路側土地使用型態、路側招牌廣告等；本研究篩選採用道路寬度、電線密度、建築物高度、人行道寬度等四種評估指標為代表來進行研究，經過調查並且歸一化為  $X_{ij}$  值。所有的準則都以百分率之概念做數值量化，此舉理由，乃在於避免短路段道路必相對優於長路段道路之不合理情形發生。



(1) 道路寬度：車道寬度即為道路空間寬度，就單向道路而言，道路寬度等於車道寬度，如道路為雙向道，則道路寬度則分割成各半。

$$RW_i = \left( \text{Vehicle Lane Width in Meter} \right)_i \quad (4-2)$$

(2) 電線密度：考慮電線當成潛在的空中障礙物影響道路淨高、或掉落物影響道路空間，電線的計算方式乃以橫跨上部空間次數計算之。

$$WD_i = \left( \text{Times of Wires across per Road Length} \right)_i \quad (4-3)$$

(3) 建築物高度：建築物隱含人口及災害後造成的旅次意義，道路承受更多的交通負擔，尤其在兩側有更高的建築物。建築物的高度相對於道路，可考慮為一項指標。假定路段  $i$  的長度為  $L_i$ ，建築物在道路的兩側可能有許多不同的高度，如以  $l_{ij}$  來表示各高度之建築物沿著路段  $i$  的長度，則路段之建築物高度即可轉換為百分率之概念。

$$BH_i = \sum_j \left( \text{Building Height in Floor} \right)_{ij} \times \frac{l_{ij}}{2L_i} \quad (4-4)$$

(4) 人行道寬度：台灣之人行道沿著路段通常具有不連續之特性，且人行道亦可能僅出現在路段的一側，或可能在同時出現雙側但兩側寬度不同；此處，寬度百分率單位用來處理此種情況，甚為簡易合適。假設路段  $i$  有分離的人行道片段，其中之一表示為  $j$ ，而該片段之人行道長度為  $l_{ij}$ ，所有的片段的人行道空間貢獻於一路段的人行道空間，如此可描述如式 (4-5) 如下：

$$SW_i = \sum_j \left( \text{Sidewalk Width in Meter} \right)_{ij} \times \frac{l_{ij}}{2L_i} \quad (4-5)$$

由式 (4-2) 至式 (4-5)，四類指標可由實際情況的調查資料計算獲得，然而，由於彼此之間單位不同而無法比較，因此須以標準化方式處理。對於道路空間正面的保護因子，如  $RW_i$  和  $SW_i$ ，將與道路網中的最高理想值做比較，同時負面指標，如  $WD_i$  和  $BH_i$ ，則透過相比於道路網中最高理想值之間的落差距離，再與最高理想值本身比較，計算方式可參考式 (4-6) 如下：

$$\begin{aligned} LOS_i &= \sum_j W_j X_{ij} = W_1 X_{i1} + W_2 X_{i2} + W_3 X_{i3} + W_4 X_{i4} \\ &= W_1 \frac{RW_i}{\max_k RW_k} + W_2 \frac{\max_k WD_k - WD_i}{\max_k WD_k} + W_3 \frac{\max_k BH_k - BH_i}{\max_k BH_k} + W_4 \frac{SW_i}{\max_k SW_k} \end{aligned} \quad (4-6)$$

由4.2.1與4.2.2節，建立起評估道路路段空間品質指標，其評估架構如圖4.4所示，包括四個層級：(1) 目標層顯示道路對於防制震災的品質；(2) 標的層代表相關的因子；(3) 準則層評估道路之路段；(4) 對象層即為針對實際調查該路段對象所獲得之資料，進行量化評量之工作。權重獲取方法可採用多準則評估等方法 (馮、林，2000)。

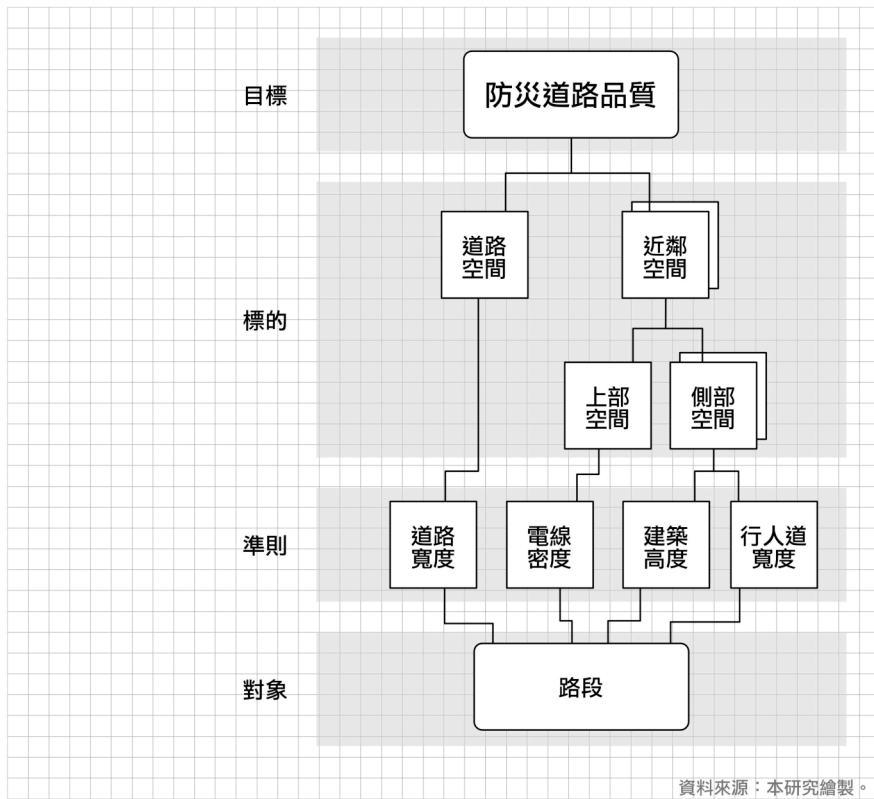


圖4.4 防災道路空間之評量架構

## 第五章 防災存活路網建構

本研究之防災存活路網之建構，採以可靠、全面、快捷三大目標（圖5.1）。地震災害之不確定性，致使路網遭受破壞時，仍可透過替代通路，確保與災區之連接，此乃路網之可靠；連結所有可能作為避難、收容候選場所之路網，以因應災害來臨時刻之準備，此乃路網之全面；從各類救援單位出發之救援機具，通過最具時間效率之路網，滿足各種緊急活動之需求，此乃路網之快捷。由第一章之防災路網架構，及第二章文獻回顧所獲之心得，並接續第三章之防災路網分析、第四章道路評估；本第五章探討此三大目標，透過採用存活路網、繞路路網、最短路徑樹等路網模型之延伸應用，發展出責任分區路網、系統繞路路網、與互援路網等模型，以作為建構合適之防災路網模型之基礎。最後，並建立此三大目標之相關指標，作為評量的基準；該評估模式可提供評估不同地區存活路網結構之方法。

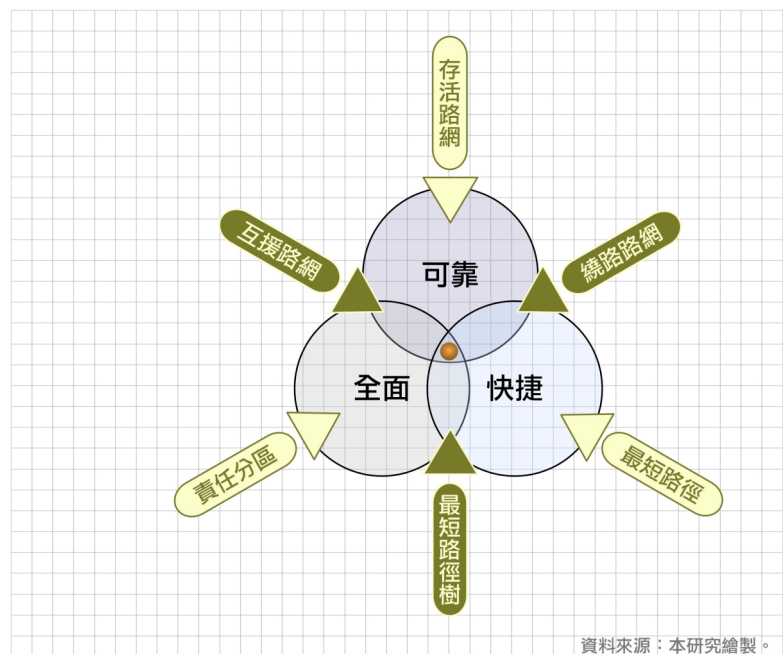


圖5.1 防災路網模型架構

### 5.1 責任分區路網模型

#### 5.1.1 責任分區之定義

「責任分區」：地震災害期間，時間效率極被關注。由路網所涵蓋的需求點，須有明確指定的救援單位來提供服務；換言之，即便路網錯綜複雜，任一救援單位都非常清楚並了解，它所負責的災點為何、應循走的路線為何。時間將被大量節省，假若每個救援單位在災害發生的時刻，都立即知道該往哪裡去。因此，每一個災點應歸屬於特定一個救援單位；而該救援單位負責的所有災點，便形成其責任分區。分區的目的在提高時間效率；因此歸屬於此救援單位的災點，必低於（或等於）由彼單位來救援將花費的時間成本；此即災點歸屬救援單位之責任分區的基本原則。由時間效率的觀點，透過救援單位區位與最短路徑樹的特性結合，便可獲致救援單位與需求單位之間最有時間效率之路網；首先作出以下相關定義，以進一步闡述。

### 5.1.2 責任分區之陳述

令  $T_{S_i}$  為具供給性質之節點  $S_i$  為根而發展的最短路徑樹， $d_{T_{S_i}}(P_{T_{S_i}}(D_N))$  表示  $T_{S_i}$  中從  $S_i$  至具需求特性之任意末端節點  $D_N$  最短路徑  $P_{T_{S_i}}(D_N)$  的距離成本， $d_{T_{S_i}}(D_k, D_N)$  表示  $T_{S_i}$  中最短路徑途中之任意需求點  $D_k$  至末端需求節點間  $D_N$  的距離。

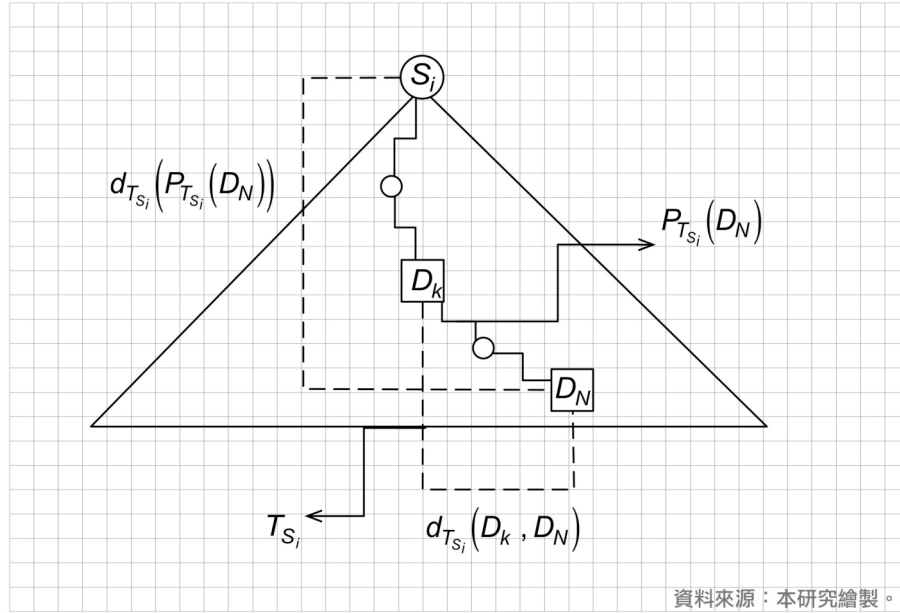


圖5.2 最短路徑樹符號定義

$S_i$ ：供給點  $i$ 。

$T_{S_i}$ ：最短路徑樹，以  $S_i$  為根。

$D_k$ ：最短路徑樹中之中途需求點  $k$ 。

$D_N$ ：最短路徑樹中之末端需求點  $N$ 。

$P_{T_{S_i}}(D_N)$ ：最短路徑樹  $T_{S_i}$  中，由供給點  $S_i$  到需求點  $D_N$  之路徑。

$d_{T_{S_i}}(P_{T_{S_i}}(D_N))$ ：最短路徑樹  $T_{S_i}$  中，由供給點  $S_i$  到需求點  $D_N$  路徑之成本權數。

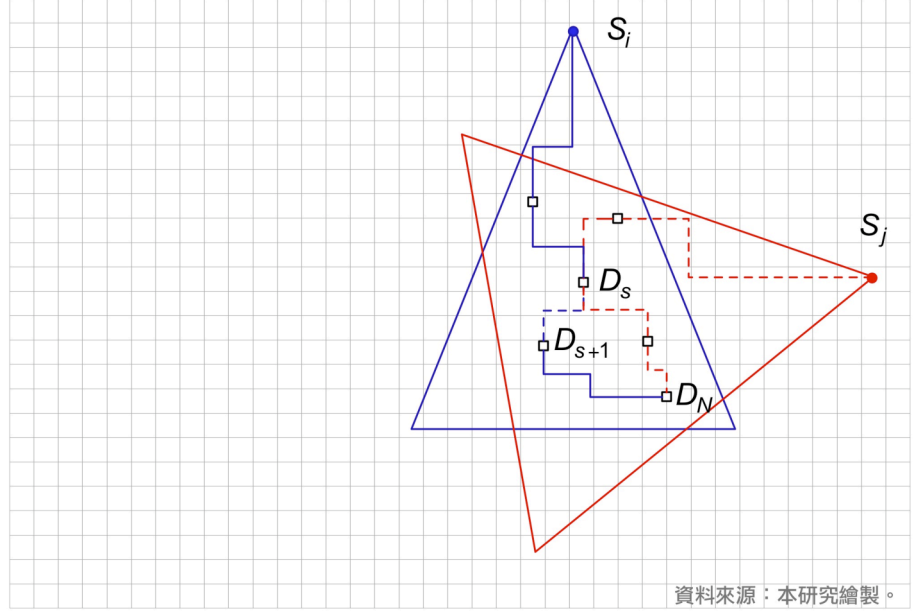
$d_{T_{S_i}}(D_k, D_N)$ ：最短路徑樹  $T_{S_i}$  中，由需求點  $D_k$  到需求點  $D_N$  路徑之成本權數。

圖5.2中，三角形範圍內，即為最短路徑樹之範圍；出發中心地  $S_i$  即為救援單位（供給點），其到達在此範圍內所包含的任何需求點，都具有成本權數最小之特性。路網之中若包含有多個救援單位，則每一救援單位應負責路網之中特定的需求點；此即表示各救援單位所形成的責任分區內，必比其他救援單位到達此區域內的需求點更具效率。透過以下的相關簡理，可以證實這樣的結論：責任分區演算法，可建構出多救援單位最具效率的責任分區路網。

### 5.1.3 責任分區之原理

#### 1. 簡理一

若某需求點  $D_s$  分由供給設施  $S_i$ 、 $S_j$  分別發展之最短路徑樹涵蓋服務，則  $D_s$  與任意末端點  $D_N$  間之任一點  $D_{s+1}$  必也同時由  $S_i$ 、 $S_j$  分別發展之最短路徑樹涵蓋服務。簡言之，若  $D_s \in \{P_{T_{S_i}}(D_N), P_{T_{S_j}}(D_N)\}$ 、與  $D_{s+1} \notin P_{T_{S_j}}(D_N), \forall s \in N$ ，則  $|\{D_s\}| = 0$ 。



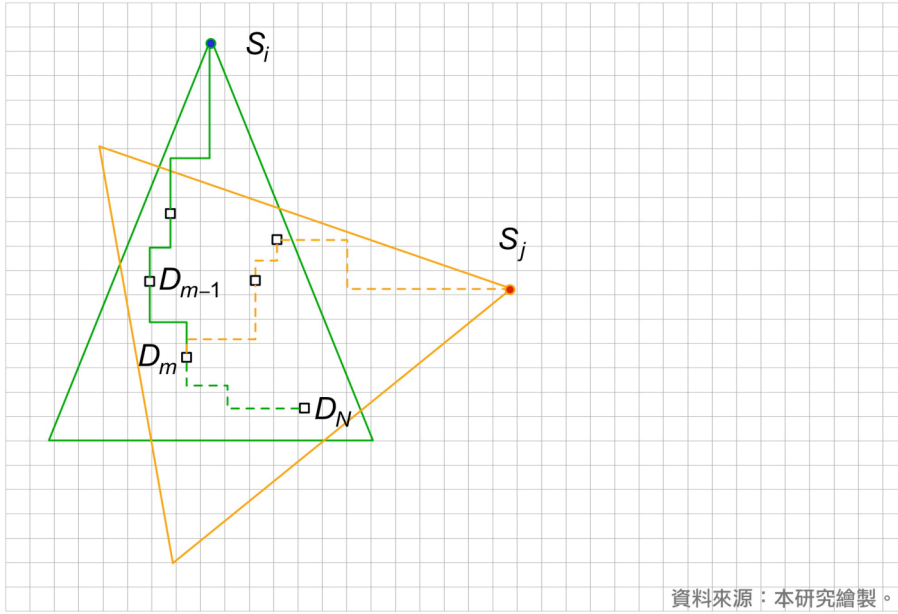
資料來源：本研究繪製。

圖5.3 簡理一

證明：令  $D_s$  為  $P_{T_{S_i}}(D_N)$ 、 $P_{T_{S_j}}(D_N)$  之共同需求點，則其路徑節點順序可表示為  $P_{T_{S_i}}(D_N) = \langle D_1, D_2, \dots, D_s, \dots, D_k, \dots, D_N \rangle$ 、與  $P_{T_{S_j}}(D_N) = \langle D_1, D_2, \dots, D_s, \dots, D_k, \dots, D_N \rangle$ ；由於  $P_{T_{S_i}}(D_N)$ 、 $P_{T_{S_j}}(D_N)$  是經由最短路徑演算法求得，若  $d_{T_{S_i}}(D_s, D_N) < d_{T_{S_j}}(D_s, D_N)$ ，則  $P_{T_{S_i}}(D_N)$  應修正為  $P_{T_{S_i}}(D_N) = \langle D_1, D_2, \dots, D_s, \dots, D_k, \dots, D_N \rangle$  才合理，反之亦然，因此可得  $D_k = D_k$ ；若要能同時滿足簡理一之前提條件（ $D_s$  之定義）： $D_s \in \{P_{T_{S_i}}(D_N), P_{T_{S_j}}(D_N)\}$ 、 $D_{s+1} \notin P_{T_{S_j}}(D_N), \forall s \in N$ ，由以上推論結果可證實該  $D_s$  並不存在，因此  $|\{D_s\}| = 0$ 。

#### 2. 簡理二

若某需求點  $D_m$  分由供給設施  $S_i$ 、 $S_j$  分別發展之最短路徑樹涵蓋服務，且  $D_m$  與供給設施  $S_i$  間之前一點  $D_{m-1}$  未由  $S_j$  發展之最短路徑樹涵蓋服務，則到達某一末端節點  $D_N$  的路徑上，這樣的  $D_m$  必定不多於一個。簡言之，若  $D_m \in \{P_{T_{S_i}}(D_N), P_{T_{S_j}}(D_N)\}$ 、與  $D_{m-1} \notin P_{T_{S_j}}(D_N), \forall m \in N$ ，則  $|\{D_m\}| \leq 1$ 。



資料來源：本研究繪製。

圖5.4 簡理二

證明：令  $D_m$  為  $P_{T_{S_i}}(D_N)$ 、 $P_{T_{S_j}}(D_N)$  之共同需求點，則其路徑節點順序可表示為  $P_{T_{S_i}}(D_N) = \langle D_1, D_2, \dots, D_m, \dots, D_k, \dots, D_N \rangle$ ，與  $P_{T_{S_j}}(D_N) = \langle D_1, D_2, \dots, D_m, \dots, D_k, \dots, D_N \rangle$ ；若  $P_{T_{S_i}}(D_N) \subset P_{T_{S_j}}(D_N)$ ，即  $D_j = D_j', D_{j+1} = D_{j+1}', \dots, D_{m-1} = D_{m-1}', D_m = D_m, \dots$ ，則  $\left| \{D_m\} \right| = 0$ ；若  $P_{T_{S_i}}(D_N) \cap P_{T_{S_j}}(D_N) = \{D_k\}, k \in K \subset N, |K| > 1$ ，由簡理一可知  $\left| \{D_s\} \right| = 0$ ，表示兩個不同的最短路徑樹中之任一最短路徑若合併於  $D_m$ ，則從  $D_m$  以後的路徑就會一直重疊直到  $D_N$  為止，而不再分離，因此  $\left| \{D_m\} \right| = 1$ ；所以  $\left| \{D_m\} \right| \leq 1$ 。

### 3. 簡理三

若需求節點  $D_k$  較靠近供給設施  $S_i$  則所有在  $D_k$  與任一末端節點  $D_N$  之間的需求節點，皆靠近供給設施  $S_i$ 。意即，若  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) = \min_{\forall S_j \in S} \{d_{T_{S_j}}(P_{T_{S_j}}(D_k))\}$ ，則  $d_{T_{S_i}}(P_{T_{S_i}}(D_{k+1})) \leq \min_{S_i \neq S_j, \forall S_j \in S} d_{T_{S_j}}(P_{T_{S_j}}(D_{k+1}))$ 。

(1) 前導推論： $P_{T_{S_i}}(D_N) = \langle S_i, D_1, D_2, \dots, D_k, \dots, D_N \rangle$  為最短路徑樹  $T_{S_i}$  中供給點  $S_i$  到達需求點  $D_N$  之路線，若  $d_{T_{S_i}}(P_{T_{S_i}}(D_N)) < d_{T_{S_j}}(P_{T_{S_j}}(D_N))$ ，則  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)), \forall k \in N$ 。

證明：

#### A. 情況1

若  $P_{T_{S_i}}(D_N) \subset P_{T_{S_j}}(D_N)$ ，則情形為  $P_{T_{S_j}}(D_N) = \langle S_j, D_1, D_2, \dots, D_N, S_i, D_1, D_2, \dots, D_k, \dots, D_N \rangle$ ，則  $d_{T_{S_j}}(P_{T_{S_j}}(D_k)) = d_{T_{S_j}}(P_{T_{S_j}}(S_i)) + d_{T_{S_i}}(P_{T_{S_i}}(D_k)), \forall k \in N$ ，因距離權數為正，故  $d_{T_{S_j}}(P_{T_{S_j}}(S_i)) > 0$ ，所以  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)), \forall k \in N$ 。

### B. 情況2

若  $P_{T_{S_i}}(D_N) \cap P_{T_{S_j}}(D_N) = \{D_N\}$ ，則  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)) = \infty, \forall k, k \neq N$ ，且已知  $d_{T_{S_i}}(P_{T_{S_i}}(D_N)) < d_{T_{S_j}}(P_{T_{S_j}}(D_N))$ ，所以  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)), \forall k \in N$ 。

### C. 情況3

若  $P_{T_{S_i}}(D_N) \cap P_{T_{S_j}}(D_N) = \{D_k\}, k \in K \subset N, |K| > 1$ ，則由簡理一、簡理二可知  $|D_m| = 1$ ；故可將情況3分成情況1和情況2之組合：令  $D_M$  即為合併點， $k \leq M$  時為情況2， $k > M$  時為情況1；因此  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)), k \leq M$ 、 $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)), k > M$ ，所以  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_j}}(P_{T_{S_j}}(D_k)), \forall k \in N$ 。

### (2) 簡理三證明

#### A. 基礎步驟

由簡理一可知上述結論於  $|S_i| = 2$  時成立。

#### B. 歸納步驟

假設  $|S_i| = n$  於  $n \geq 2$  時亦成立，即於  $n$  個樹之中存在  $P_{T_{S_i}}(D_N) \in T_{S_i}$  使得  $d_{T_{S_i}}(P_{T_{S_i}}(D_N)) \leq \min_{S_i \neq S_j, \forall S_j \in S} d_{T_{S_j}}(P_{T_{S_j}}(D_N))$ 。當  $|S_i| = n+1$ ，由簡理三之假設已知  $d_{T_{S_i}}(P_{T_{S_i}}(D_N)) < d_{T_{S_{n+1}}}(P_{T_{S_{n+1}}}(D_N))$ ；則由前導推論可獲知  $d_{T_{S_i}}(P_{T_{S_i}}(D_k)) < d_{T_{S_{n+1}}}(P_{T_{S_{n+1}}}(D_k))$ 。

### 5.1.4 責任分區之演算法

簡理三係由簡理一、簡理二推論而來，而且為責任分區之重要基礎。簡理三說明了，若想同時由多個供給節點自行發展最短路徑樹，則可透過快速將子樹歸給某一責任分區之方法，而無須重複節點距離的確認，此法將大幅提升演算效率，並快速區分出最佳的責任分區。而在各責任分區之結果中，達到每個需求節點所受到供給節點提供之服務、其權數成本必定小於（或等於）由其他供給節點所提供服務之目標。

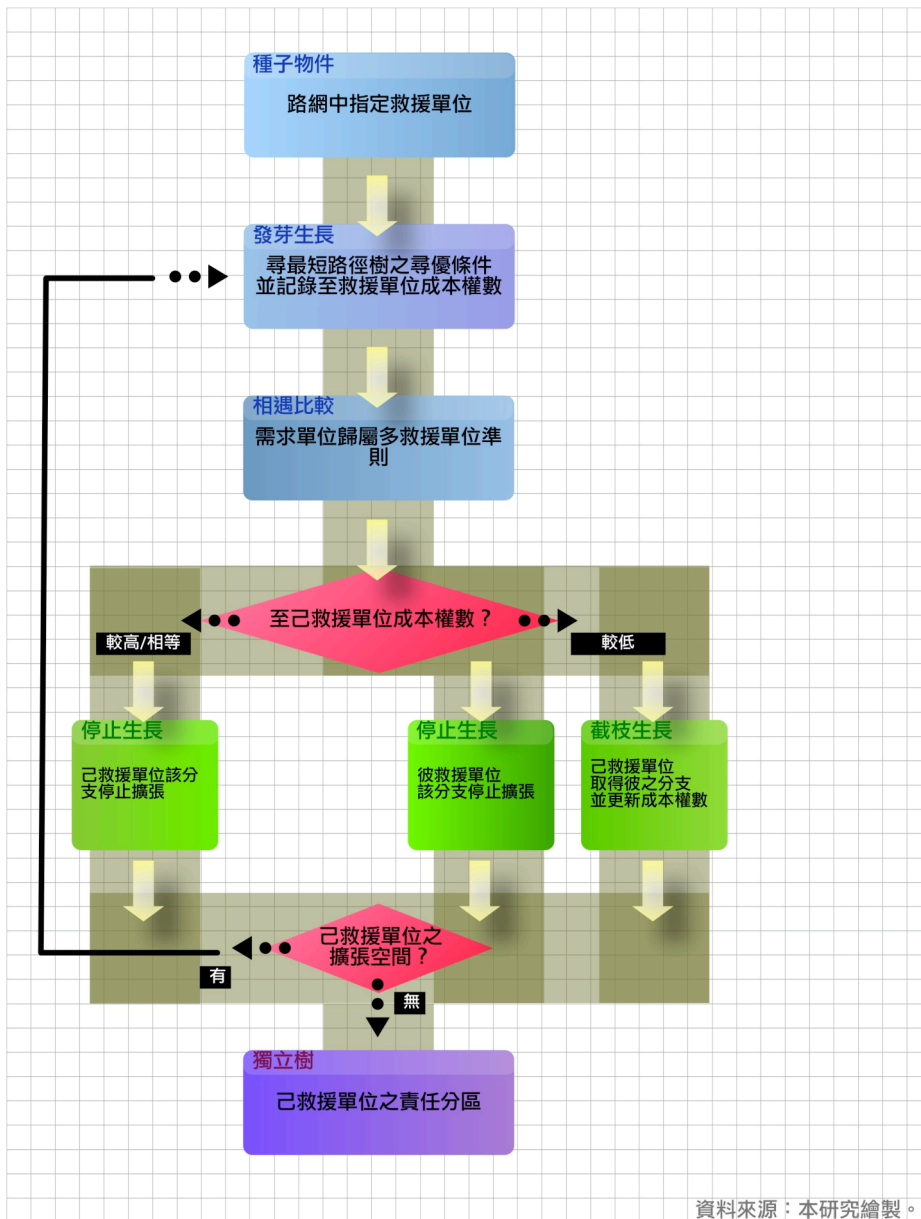


圖5.5 責任分區演算法概念

責任分區演算法概念如圖5.5。首先，將所有救援單位（供給點）當作多個種子物件，投入由第三章所產生的基礎路網之中；其次，採以最短路徑之尋優法使每一個種子發芽擴張，以觸及需求點，並記錄該點與種子之間的成本權數。當不同種子發芽產生之樹，於某一需求點相遇時，則進行優劣比較。基於簡理三，若相遇之需求點離某種子成本權數較高（或相同）時，則樹就不再繼續往此處生長；反之，若成本權數較低時，則除了將該需求點歸於己，原先該需求點生長過的子樹，也都截枝於新的樹，對新的種子更新成本權數後繼續生長。被截枝端則停止生長。此程序不斷進行，直到基礎路網結構已沒有種子發芽生長的空間，責任分區就確立完成。



## 5.2 系統繞路路網模型

本研究採用存活路網模型，做為防災路網規劃的主要方法之一；此乃由於地震造成路網破壞之不確定性。存活路網係在探討路網連結度之問題，本節即以道路之節線觀點，透過該模型之延伸應用來尋找替代路徑的方法，因此構建之路網，為因應道路可能被破壞的情況。

存活路網模型之防災路網，以口語表達之意，就是保留了路網具有替代道路的特性；換而言之，面對預期性的路網破壞，已有使用替代道路做繞路的準備；因此，防災路網規劃，如能將繞路成本的特性納入考量，並深化於存活路網模型之中，將使得路網更具效率。繞路是基於原先路線無法使用，而採行不同替代路徑的觀念。所謂替代路徑，係指整條原路線的過程中，任一途中地點，發生轉換至其它路徑的情形。假若，無論災害發生地點在原路線上的任一位置，皆可透過特定替代路徑重新連接兩點，此即前述之存活路網模型；然而，繞路除了是有替代路徑之功能外，仍須要考量繞路的成本。倘若繞路之替代路徑為成本極高之繞路路徑，即使仍可以連結供需節點，但就救災的時效性而言，也失去了繞路的意義。因此，繞路還須含有最短路徑的概念。而繞路模型提供了方法，以指出路網中某一路段之毀損，將造成最可觀的繞路成本；同理，該模型也可尋找出繞路成本最小之替代路徑，以便提高路網的效率，此即為繞路模型對防災路網貢獻之處。

本研究所採用之繞路模型，係由Nardelli等學者所提出。該研究探討最短路徑樹中任意兩點間之路線的繞路問題。若該兩點間之最短路徑中任一路段無法通行時，則必須從原路網中，找尋另一路徑來連通此兩點，此即為繞路；而某一路段無法通行，並造成原來兩點間之繞路最遠，此即為繞路之繞路敏感路段。本研究則利用此觀念，來構建系統繞路模型。

### 5.2.1 系統繞路之定義

「系統繞路」：任一路段被破壞的情形下，單一救援單位透過某一橋段，以繞達全部需求場所之路徑。由於地震災害破壞之不確定性高，若任一路段遭受地震破壞後，能立即明瞭取代之橋段之所在，則防災路網將依然明確，並能將責任分區內之救援單位，有系統地與全部的需求場所連接起來。

### 5.2.2 系統繞路之陳述

存活路網為探討路網失靈之議題，成本最小的存活路網結構就是二邊連通圖，即移除任一路段，仍可透過替代路徑使原來任意兩個節點相互連通。然而，此時連通的路徑，並不隱含繞路成本趨小化的概念；因此，此存活路網結構下的繞路成本，有可能與先前未繞路之路徑成本高出許多。系統繞路模型秉持兩個觀點：1.預期道路可能被災害破壞、然未設定何路段將失靈的情形下，考量繞路成本趨小化；2.未預先設定何需求點於災時會有真實需求的立場下，考量供給點與整體需求點之間的總繞路成本趨小化。

前述之責任分區路網模型，具有未需繞路時，最具時間效率之結構。其最短路徑樹是規劃單一供給點至其範圍可及所有需求點最有效率的路網規劃方法。然而，樹結構有「樹的任一節線都是割線（cut edge）」的特性，因此沒有路段的容錯（fault tolerance）能力。所以，任一路段被破壞，都會導致樹中一個節點以上必須繞路的情形發生；而這些需要透過繞路才能與供給點連通之節

點，其個別繞路成本的總合，本研究定義為「系統繞路成本」。系統繞路模型，即為在最短路徑樹結構下，以系統繞路成本最小化之尋優方式，應用繞路模型所構建之存活路網。

### 5.2.3 系統繞路之原理

系統繞路模型如圖5.6，以及下述符號與定義說明。

假若令  $T$  為圖形  $G$  的最短路徑樹，其根（root）為  $r$ ； $d(s, r)$  表示  $G$  中任一節點  $s$  至根  $r$  最短路徑  $P_G(s, r)$  的距離。假若除去某一邊  $e$ ，令  $U_{T, e}$  為擁有根的子樹（subtree），而  $L_{T, e} = T - U_{T, e} - e$  表示為剩下的樹。根據Nardelli等人的研究，在  $e$  移除後， $U_{T, e}$  中所有的點與  $r$  之間的距離都不會改變，然而  $L_{T, e}$  中的點與  $r$  之間的距離卻有可能增加。令  $B_{T, e}(m, a)$  為  $L_{T, e}$  與  $U_{T, e}$  之間的橋段（ $m$  在  $L_{T, e}$  而  $a$  在  $U_{T, e}$ ）， $d(B_{T, e}(m, a))$  表示橋段的成本；則此時  $r$  即可透過含有  $B_{T, e}(m, a)$  的路徑到達  $L_{T, e}$  中的任何一點，而且不再需要經過  $e$ 。值得注意的是，此時  $L_{T, e}$  中所有的點，都一定要有屬於  $T_r$  以外的邊，才有能力與  $r$  連結。這裡我們定義了一個指標，來衡量  $L_{T, e}$  中所有的點繞路到達  $r$  的總成本，也就是系統繞路成本：

$$SDC_{T_r - e} = \sum_{s, m \in L_{T, e}, a, r \in U_{T, e}} \left\{ d(s, m) + d(B_{T, e}(m, a)) + d(a, r) \right\} \quad (5-1)$$

系統繞路成本為用來衡量由於  $e$  所引起所有節點的繞路成本總合。然而實際上所謂「所有節點」僅指  $L_{T, e}$  中的需求節點，因為  $U_{T, e}$  中的需求節點無需繞路即可到達。然而，很明顯地，如果每一個  $L_{T, e}$  中的節點為達最短繞路的目的，都想擁有自己的橋段  $B_{T, e}(m, a)$  來到達  $r$ ，則這將是可觀成本的路網結構，在此本研沿用Tarjan橋段的概念，究提出「共同橋段」的做法以簡化路網結構，當  $e$  被移除，利用  $L_{T, e}$  中原來的路網結構，讓所有其中的節點依循原路網結構，到達其中特定的一個集合點  $m$ ，並透過相同的橋段  $B_{T, e}(m, a)$  來到達  $r$ ，則成本將可大幅降低。而此時的繞路成本問題便簡化為共同橋段的選擇問題。本研究中，將這樣的問題稱作為系統繞路問題（SD Problem）。系統繞路問題具有「共同集合點、共同橋段」之特性，因而在此訂了另一個參數來衡量共同集合的成本，也就是聚集成本（Merging Cost）：

$$MC_G(m) = \sum_{s \in G} d(s, m) \quad (5-2)$$

聚集成本為  $L_{T, e}$  中所有節點到達其中特定聚集點  $m$  的總距離；因此，若令  $V_{L_{T, e}}$  表示為  $L_{T, e}$  的節點集合，則前述之系統繞路成本  $SDC_{T_r - e}$  就可以轉換成：

$$SDC_{T_r - e} = MC_{L_{T, e}}(m) + |V_{L_{T, e}}| \times \left[ d(B_{T, e}(m, a)) + d(a, r) \right] \quad (5-3)$$

其中第一項表示為  $L_{T, e}$  節點在聚集點  $m$  的聚集成本，第二項則表示由聚集點  $m$  到根  $r$  的進入成本，其中包含了橋段的權數與進入點  $a$  至  $r$  的距離加總，再乘上  $L_{T, e}$  中的節點數目。

系統繞路的成本為子樹  $L_{T, e}$  中的每一個節點，繞路至根  $r$  的距離總合。上述方式可將系統繞路成本分為聚集成本、進入成本兩項；從其中可以發現，聚集成本若要降低，則聚集點傾向靠近子樹的中心位置；然而， $L_{T, e}$  中的節點數量亦左右著進入成本，其可視為單一節點進入成本的權數（此因每一個節點的進入成本皆相同之故），因此子樹的需求節點數量越多，進入成本就越顯得重要，

聚集成本並非在構建樹的時候就已獲得，直覺的計算方式，則是在獲知破壞路段位置後，由其子樹每一個節點為出發點，進行深度優先搜尋、或再一次透過最短路徑演算來獲得；但事實上並不需如此大費周章。既然初始已獲得樹的結構，即表示已熟知樹結構中任何一點到達根的路徑；透過這個特性，並利用交集、聯集的演算，就可以快速獲得聚集成本，如式（5-4）。透過上述方式以及圖5.6之說明，則可容易而快速的獲得聚集成本。

The diagram illustrates the construction of a tree  $T_r$  from a graph  $G$ . The graph  $G$  is represented by a large triangle containing a path of nodes. A specific node  $a$  is highlighted. A subtree  $T_r$  is shown above  $G$ , containing a path of nodes ending at  $a$ . A subtree  $T_{r,e}$  is shown below  $G$ , containing a path of nodes starting from  $a$  and ending at  $s$ . The diagram shows how  $T_r$  and  $T_{r,e}$  are constructed from  $G$ , with  $T_r$  being a subtree of  $G$  and  $T_{r,e}$  being a subtree of  $T_r$ .

因此，成本最小化之系統繞路問題即成為下式：

而滿足成本最小化之系統繞路問題之繞路路段即為：

因此，探討最短路徑樹中每一路段預期破壞的情形，則所有的繞路路段即為：

$$DE_{T_r} = \bigcup_{e \in T_r} MSDE_{T_r-e} \quad (5-7)$$

最後，平衡路網結構成本、以及繞路成本之路網，亦即滿足系統繞路成本最小的二連通圖，即為：

$$2ECON_{T_r} = T_r \cup DE_{T_r} \quad (5-8)$$

#### 5.2.4 系統繞路之演算法

系統繞路演算法包含下列步驟：

##### 1. 步驟一：確認基礎路網

由確認之供給點為根，發展最短路徑樹  $T_r$ ，並記錄供給點至任一需求點之間的最短路徑。

##### 2. 步驟二：基礎路網路段毀損模擬

掃描基礎路網之最短路徑樹的所有邊，依次視為毀損路段，將原基礎路網分為包含供給單位之上半樹  $U_{T_r, e}$ 、以及包含需求單位之下半樹  $L_{T_r, e}$ 。

##### 3. 步驟三：尋找系統繞路成本最小之連接橋段

透過式 (5-3) 來尋找系統繞路成本，並求取擁有最小系統繞路成本之橋段  $B_{T_r, e}$ 。

##### 4. 步驟四：系統繞路路網之構成

掃描基礎路網所有邊後，所有繞路橋段之集合，與基礎路網之集合，即為最小系統繞路路網  $T_r \cup \left( \bigcup_{e \in T_r} B_{T_r, e} \right)$ 。

### 5.3 互援路網模型

責任分區路網模型，提供所有需求點與救援單位之間最快捷的路徑；系統繞路模型，確保任一路段毀損之時，責任分區內之路網具有最佳橋段使總體繞路成本最小。而互援路網模型，以最具效率的方式，將自己責任分區內之需求點，橋接至不同責任分區的救援單位。

#### 5.3.1 互援路網之定義

「互援」：地震災害的特性為不確定性，責任分區中的救援單位，因救援量能不足、或自身受到地震損害，致無法擔起自負責任的情況時，此時須由其分區外的其他救援單位互相支援，以替代自身的救援責任。救援能力將被確保，假若每個救援單位在災害發生的時刻，都立即知道該循什麼路線、去支援哪些區域。

### 5.3.2 互援路網之陳述

若將單一救援單位之責任分區路網視為微觀路網、多救援單位之多責任分區為巨觀路網，則互援路網乃利用交互救援的方式，來滿足巨觀最佳路網連結之條件。

互援路網與責任分區路網之差異，在於以系統繞路模行構建之責任分區路網，為預期道路破壞的路網；而以巨觀最小擴張樹之互援路網，為預期救援單位失靈的路網。互援路網之目標與責任分區路網之目標，同為救援單位與需求點之間的最小救援時間；不同之處，在於責任分區路網為一對多之組合，而互援路網為多對多之情境。然而，理論上，就使用觀點而言，如能不採用繞路的替代路線、而直接走最短路線，才應是最佳的策略；因此，在每個分區中討論連結度時，不應直接以連結度的思考邏輯來討論；反之，應先保留路網中，任意供需節點對之間的最短路徑之路網結構，再來考慮連結度的問題。因此，系統繞路路徑之定位，次於責任分區之最短路徑樹結構；而互援路網之定位，次於系統繞路路徑。此乃由於最具時間效率之路網為責任分區內之最短路徑樹結構，而交互救援行為，發生於責任分區內救援單位失靈或供給量能不足之情況。

由於路網中任一節點，必歸屬於某一責任分區；因此，巨觀之防災路網具有分區層級的結構。責任分區層級內之路網為系統繞路模型所構建，因而「道路破壞預期」的觀點使各責任分區層級內的路網，皆具有至少二邊連結度以上的品質。交互救援採取同級分區層級間之路網為「設施破壞預期」的觀點，因此各分區層級間的路網，將對分區層級內的需求點而言，提供了具有至少二供給點連結度以上的品質。由以上觀點，巨觀防災路網的結構雛形，可以如下概念圖所示。

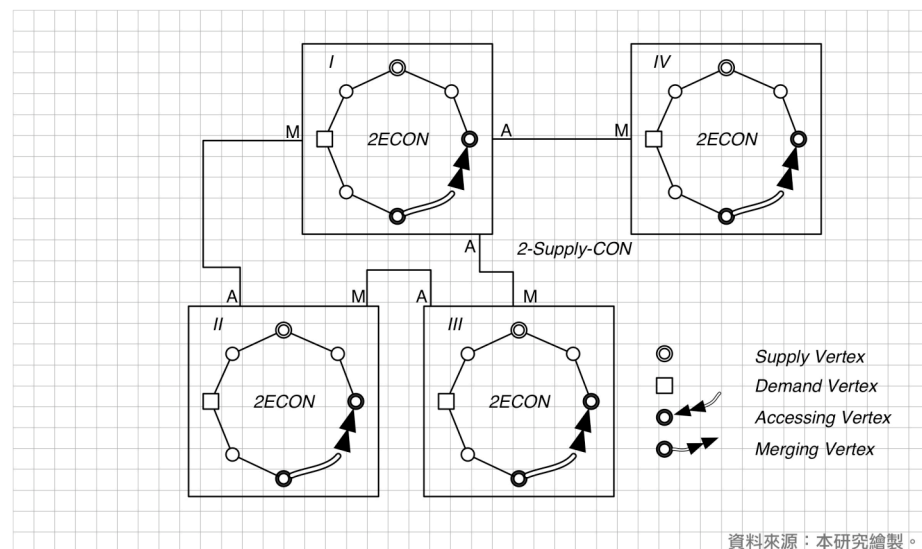


圖5.7 防災路網結構

在每個責任分區內，都存在有以供給點為根而發展、最短路徑樹的結構，以及套用系統繞路模型，所構建的橋段之結構；另一方面，責任分區之間，以各責任分區為單位，套用互援路網模型，構建分區間的橋段結構。因此在圖5.7中，區域I、II、III、與IV被視為同樣的層級，因此在連結四個區域的時候，將之視為獨立無關的區域，來進行同層級的路網連結。

### 5.3.3 互援路網之原理

互援路網之目的，乃在提高救援單位與不同責任分區間之連接。以巨觀路網角度，責任分區被簡化為單一節點，目標即在求取連接各責任分區之最有效率的路網。因此，每一責任分區將可能扮演兩種角色：**1.**於其他責任分區之救援單位供給量不足時，交互救援其他責任分區的供給單位角色；**2.**於自己供給量不足時，待其他供給單位救援的需求單位角色。以圖5.8為例，在甲責任分區內，自有一救援單位；此救援單位正常運作時，將還可作額外為其他乙區、丙區、及丁區等責任分區之候選交互救援單位。然而，連結至其他分區之方法，則需透過區內特定聯外點。聯外點係為區內具有連接另外分區之特定點。作為供給單位角色時，該聯外點以單線圓圈表示之；作為需求單位角色時，該點則以雙線圓圈表示之。兩種性質之聯外點亦可能為同一點。

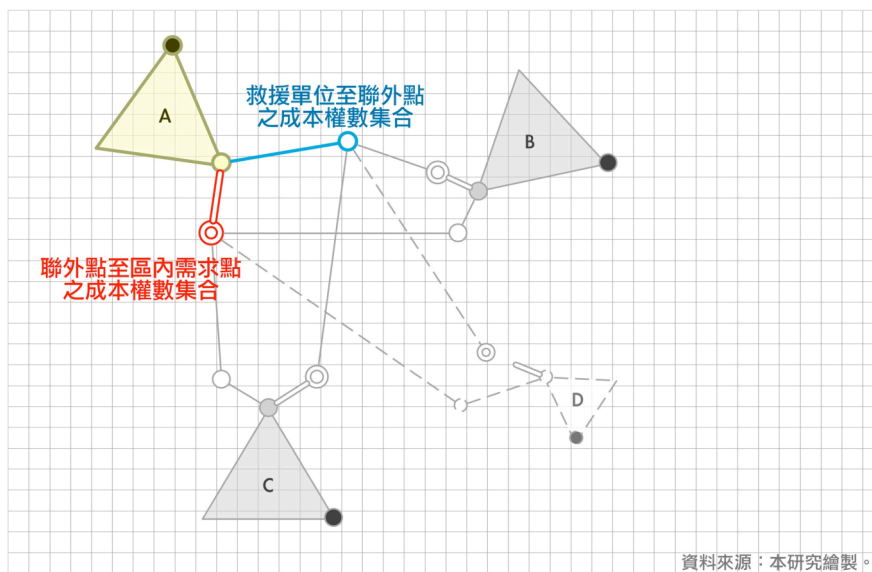


圖5.8 互援路網原理

互援路網之原理，係以最小旅行時間成本之目標，找出各責任分區間之合適橋段，以使各責任分區間能有效連結。基於各責任分區之兩種相異角色，連接橋段也有所不同。由於災害未發生時，無法確切知道災點所在位置，因此作為需求單位角色設計路網時，採以總體觀點。即先求取聯外點至區內所有可能需求點之總成本，再考量聯外橋段成本。作為供給單位角色時，係指責任分區內明確之救援單位，即先求取聯外點至區內救援單位之總成本，再考量聯外橋段成本。最後，將至區內所有可能需求點之總成本、聯外橋段成本、至區內救援單位之成本此三方成本同時考量，最小之成本組合，即建構出供給、需求兩種角色，成本皆最小之互援路網。

### 5.3.4 互援路網之演算法

互援路網之演算法，共分為五個步驟。

#### 1. 步驟一：確認候選聯外點與區間橋段

由5.1節之責任分區路網模型程序，可將路網區分為獨立不相連的多個分區。此時，找出各分區內可連接到其他分區的節點，以作為各責任分區間進出的候選聯外點。各聯外點之間的路網，為不歸屬任何責任分區之路網；尋找彼此的最短路徑並記錄其成本，以作為各責任分區間的候選橋段。其候選橋段  $B(C_{i,k}, C_{j,k'})$ ，為由責任分區  $S_i$  的聯外點  $C_{i,k}$ ，依成本最佳化原則循經責任分區間之路網，而到達責任分區  $S_j$  的聯外點  $C_{j,k'}$  之最短路徑。

#### 2. 步驟二：求取候選聯外點指標

首先，建立兩個指標來顯示候選聯外點之特性。包括供給面、需求面之指標。

(1) 聯外點供給面指標：用以衡量各聯外點至區內供給點距離。

$$SI_{S_i, k} = d_{2ECON_{S_i}}(C_{i, k}, S_i) \quad (5-9)$$

(2) 聯外點需求面指標：用以衡量各聯外點至區內所有需求點最短路徑權數總和。

$$DI_{S_i, k} = \sum_n d_{2ECON_{S_i}}(C_{i, k}, D_n) \quad (5-10)$$

#### 3. 步驟三：尋找最佳配對組合

尋找責任分區  $S_i$  的最佳互援單位  $S_j$ ；透過  $S_j$  責任分區的聯外點  $C_{j, k'}$ ，經由橋段  $B(C_{j, k'}, C_{i, k})$  連接至自己責任分區內之聯外點  $C_{i, k}$ ，再連接至區內的各需求點，求得責任分區  $S_i$  之最有效率之互援旅行成本  $MAC_{S_i}$ 。

$$MAC_{S_i} = \min_{k', k, j} \left[ SI_{S_j, k'} + d(B(C_{j, k'}, C_{i, k})) + DI_{S_i, k} \right] \quad (5-11)$$

#### 4. 步驟四：建構互援路網

互援路網  $MA$ ，即為所有區間橋段之集合路網。

$$MA = \bigcup_{i, j, k, k'} B(C_{i, k}, C_{j, k'}) \quad (5-12)$$

## 5.4 防災存活路網評估模式

本研究之路網模型，其廣泛性之原理，可運用於不同地域城市；然而，由於各地區之先天環境條件不同，若無統一之比較基準，難以判斷其路網規劃之優劣，未來資源之投入，無依據可循，方向亦難明辨。因而，本節以可靠、快捷、全面之目標，發展出可靠、快捷、全面效率衡量指標，可審視路網特性，並作為一套評估防災路網的模式，以助於判斷評估防災路網規劃配置方案之優劣比較。

### 5.4.1 防災存活路網模型

由5.1至5.3節可獲得防災路網模型 *RESCUE*；包含了，系統繞路之責任分區路網 *2ECON*，以及互援路網 *MA*。

$$RESCUE = 2ECON \cup MA \quad (5-13)$$

### 5.4.2 防災存活路網評估指標

#### 1. 可靠

(1) 區內最長繞路成本 (Longest Detour, LD)：在不使用最短路徑的情況下，每對供需點間，使用其他路段所需繞路的時間，最長的繞路時間表示著最差的替代效率。

$$LD = \max_{S_i \in S} \left\{ \sum_{D_k \in V(2ECON_{TS_i})} d_{2ECON_{TS_i}}(P_{2ECON_{TS_i}-e}(S_i, D_k)) \mid D_k \in D, e \in E(SPT_{S_i}) \right\} \quad (5-14)$$

(2) 區間平均互援成本 (Average Mutual Assistance Cost, AMAC)：互援條件係指區間救援的狀況。假若區內的救援單位發生失靈的現象，則需透過其它區域的救援單位來進行區內救援，因此互援條件可看做兩個不同救援單位間彼此的旅行時間距離。

$$AMAC = \frac{\sum_{S_i} MAC_{S_i}}{|\{S_i\}|} \quad (5-15)$$

#### 2. 快捷

(1) 平均旅行成本 (Average Travel Cost, ATC)：每對供需點間，其最短旅行成本的平均值。

$$ATC = \frac{\sum_{D_k \in D} \left( \min_{S_i \in S} \{d_{RESCUE}(D_k, S_i)\} \right)}{|\{D_k\}|} \quad (5-16)$$



(2) 最大旅行成本 (Maximum Travel Cost, MTC)：每對供需點間，其最小旅行成本路徑，即代表了救援的效率，而路網中，最大的旅行成本表示著最差的救援效率，因此作為衡量救援效率的重要指標。

$$MTC = \max_{S_i \in S} \left\{ d_{2ECON_{T_{S_i}}} (S_i, D_k) \mid D_k \in V(2ECON_{T_{S_i}}) \right\} \quad (5-17)$$

### 3. 全面效率

(1) 路網成本 (Network Cost, NC)：防災路網的考量中，路網成本並非首要考量的條件，然在相同防災路網品質條件下，成本愈低的路網，愈能付諸實現，因此路網成本為全面效率的參考。

$$NC = \sum_{e \in E(G)} w_e x_e, x_e = 1 \text{ if } e \in RESCUE, x_e = 0 \text{ otherwise} \quad (5-18)$$

#### 5.4.3 防災存活路網之綜合評估

由於各地區條件有所不同，各準則相對權重亦應視環境改變，此處以平均法為之。總體評估結果值  $E$  如下式。

$$E = 1/3 \left[ \frac{f(LD) + f(AMAC)}{2} + f(NC) + \frac{f(ATC) + f(MTC)}{2} \right] \quad (5-19)$$

其中， $f(x)$  為標準化函數。

$$f(x) = \begin{cases} 1, & \text{if } x \text{ is better than } RS_{l,x} \\ \frac{x - RS_{u,x}}{RS_{l,x} - RS_{u,x}}, & \text{if } x \text{ is between } RS_{l,x} \text{ and } RS_{u,x} \\ 0, & \text{if } x \text{ is worse than } RS_{u,x} \end{cases} \quad (5-20)$$

$RS_{l,x}$ ：專家針對  $x$  指標最理想的參考尺度值。

$RS_{u,x}$ ：專家針對  $x$  指標無法接受的參考尺度值。



## 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。

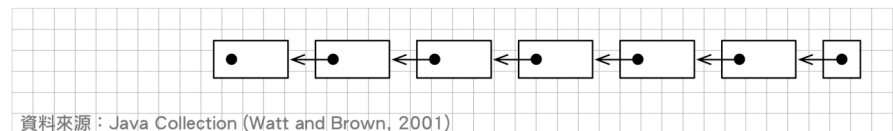


圖6.1 單連列

##### 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：

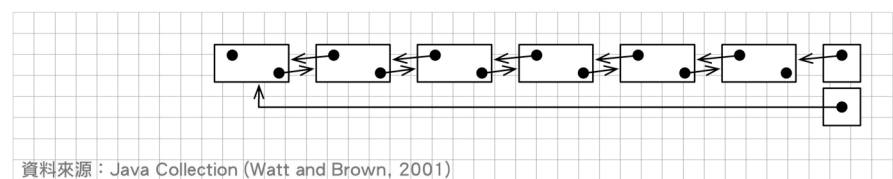


圖6.2 雙連列

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。

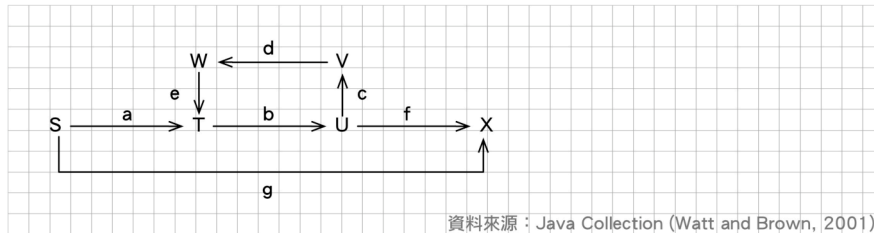


圖6.3 路網範例

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。

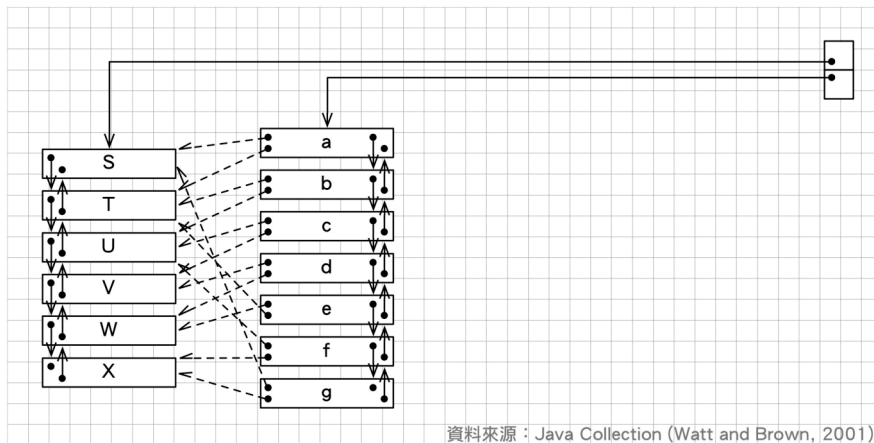
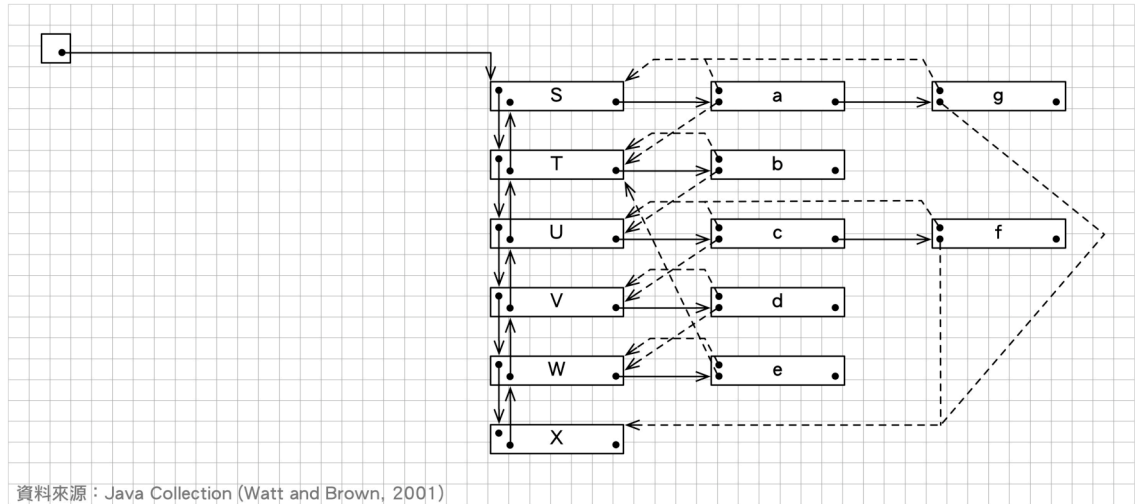


圖6.4 邊集和

## 2. 鄰接集合

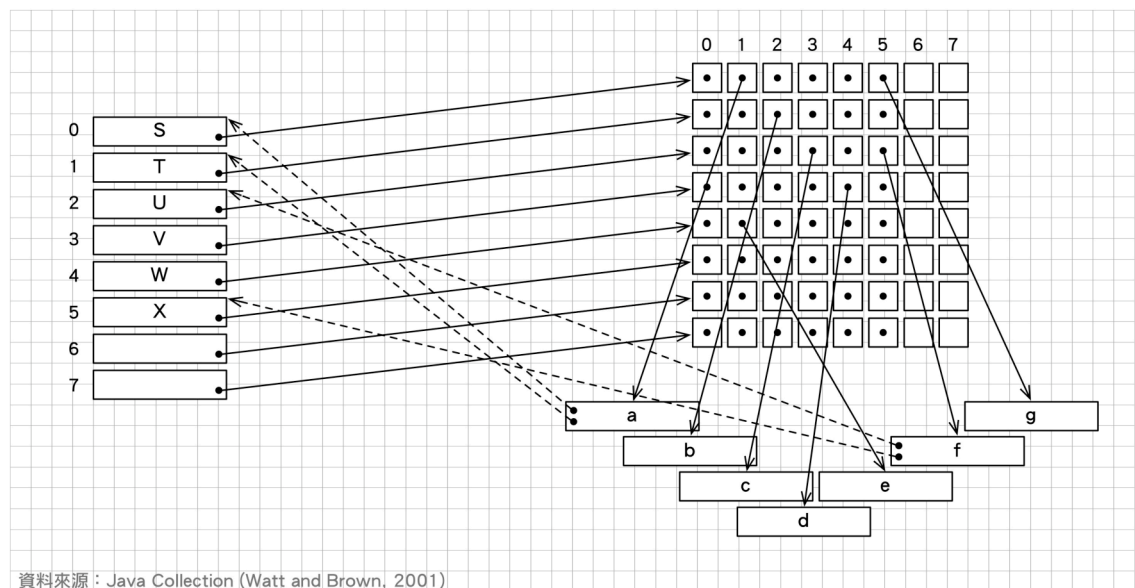
所謂鄰接集合表示法，就是將圖形中的所有點，單獨視為一個集合，並將每個點的鄰接邊集合，與該點作關係的對應。



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

圖6.5 鄰接集合

## 3. 鄰接矩陣



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

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。

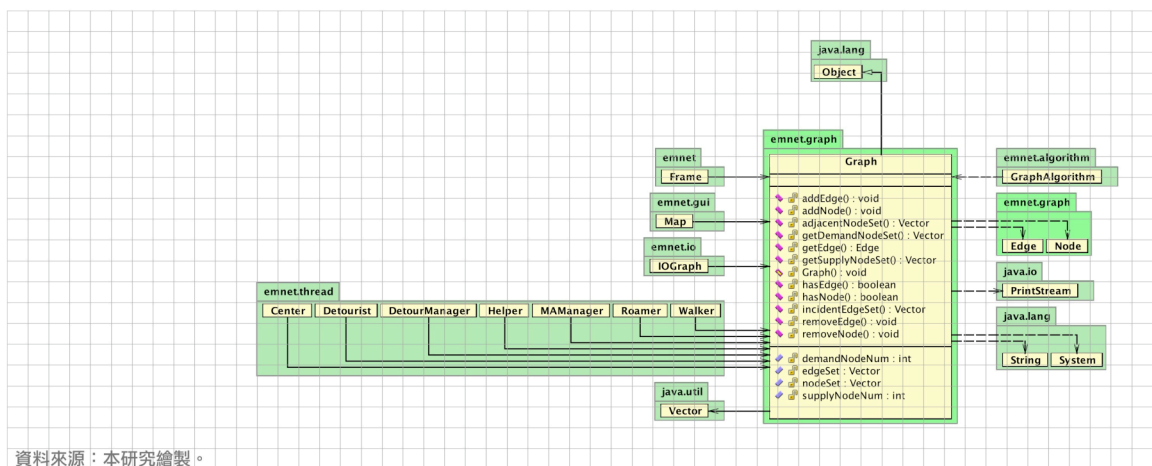


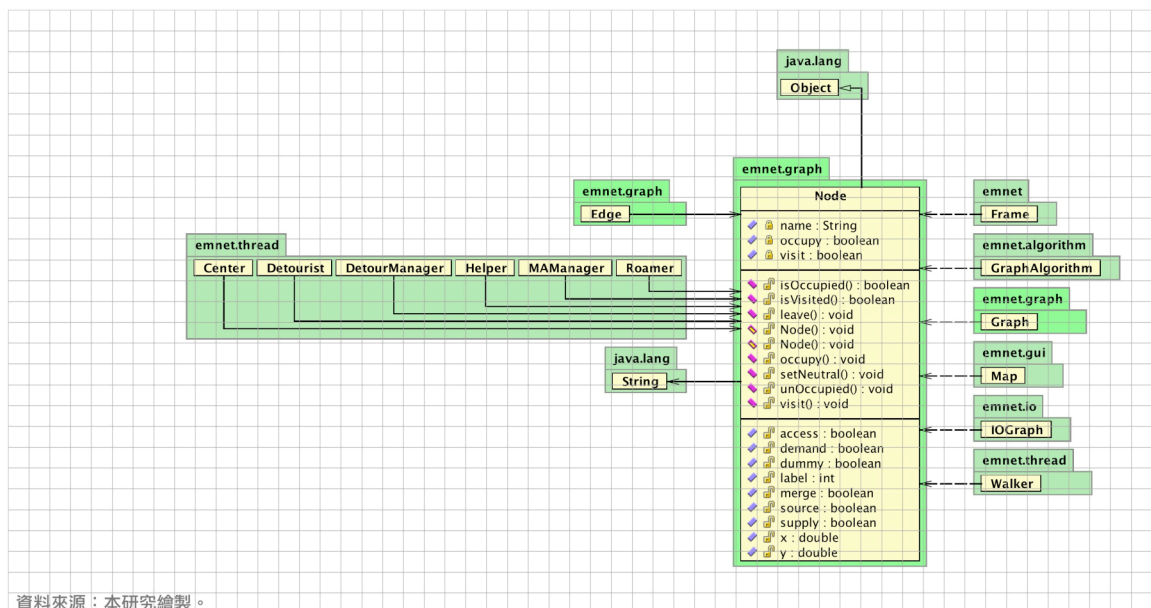
圖6.7 圖形類別

## 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。



資料來源：本研究繪製。

圖6.8 點類別

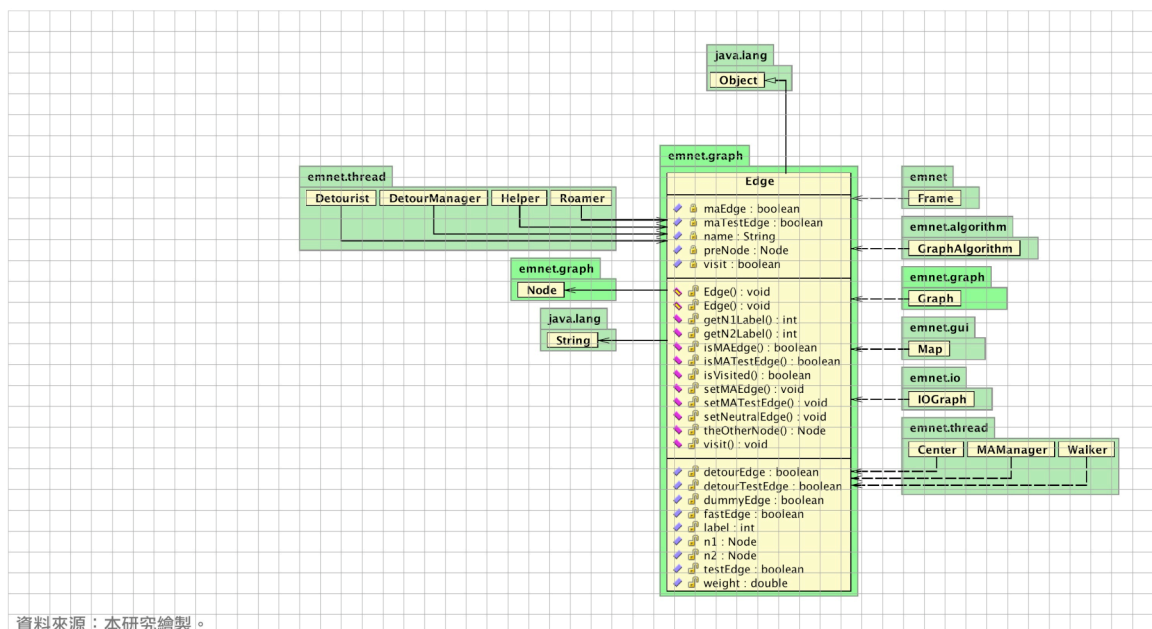


### 3. 邊

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。



資料來源：本研究繪製。

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。

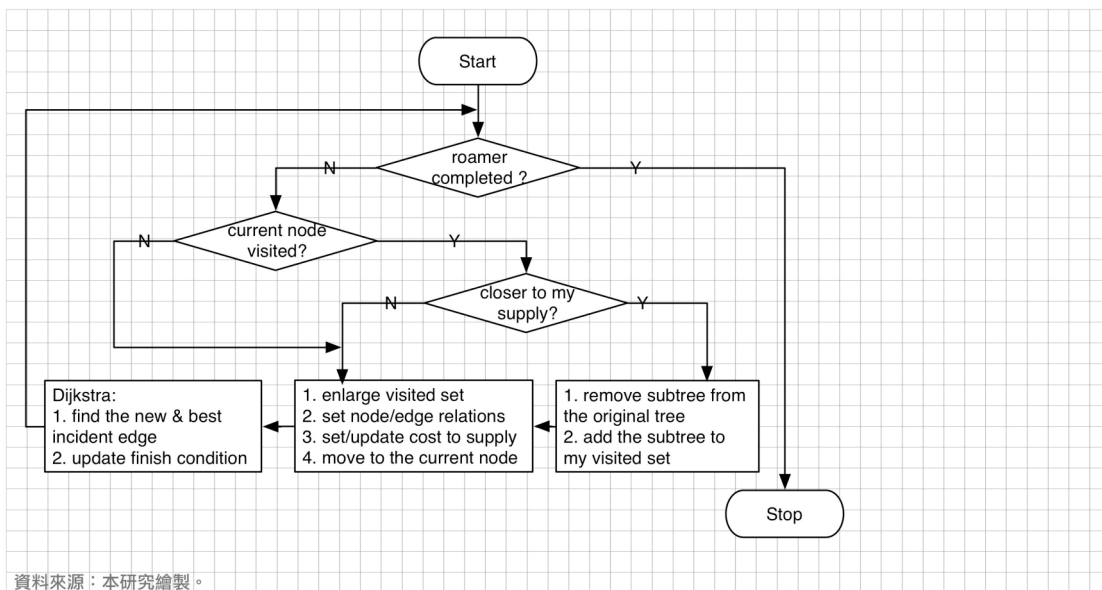


圖6.10 責任分區之演算流程

## 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。

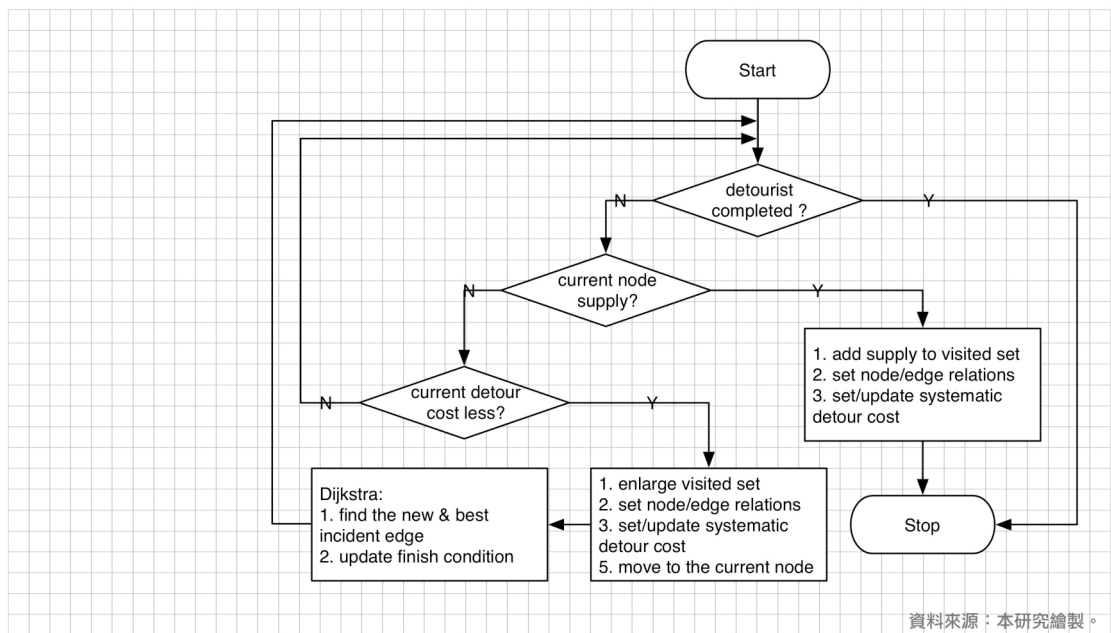


圖6.11 系統繞路之演算流程

## 6.4 互援路網

### 6.4.1 演算概念

區間互援路網之目標，在降低區內單一供給單位失靈所造成之風險；首先走路者於區內系統繞路路網中，找出某一界面節點至所有潛在需求節點的集成本，再透過命名為救援者（**helper**）之Java物件，從該點向區外供給單位求取其與橋接的組合成本，進而持續向互援管理者予以確認其組合成本是否為最佳組合，進行直至所有界面點皆測試結束；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，以作為區間最適互援路徑。

### 6.4.2 演算流程

首先，先取得整體路網與區域路網之介面節點，以做為互援路網之初始起點，救援者將採取區內聚集成本、區間橋接成本最小的尋優方式，循環比較，直至連結至區外供給單位的成本最小為止，則獲得原供給單位失靈時，區間互援之最佳路徑，其演算流程架構如圖6.12所示。

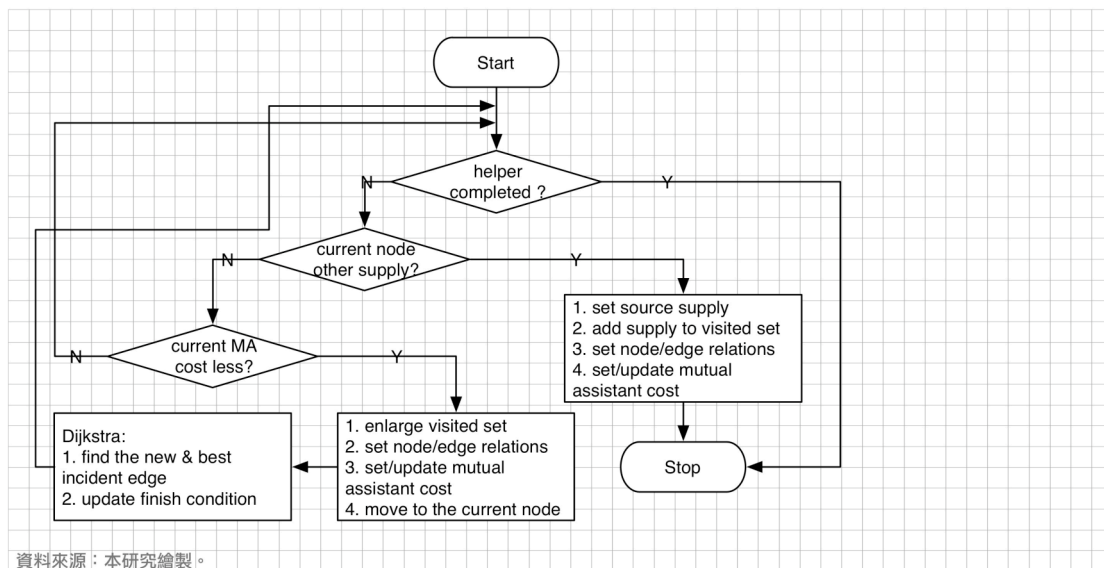
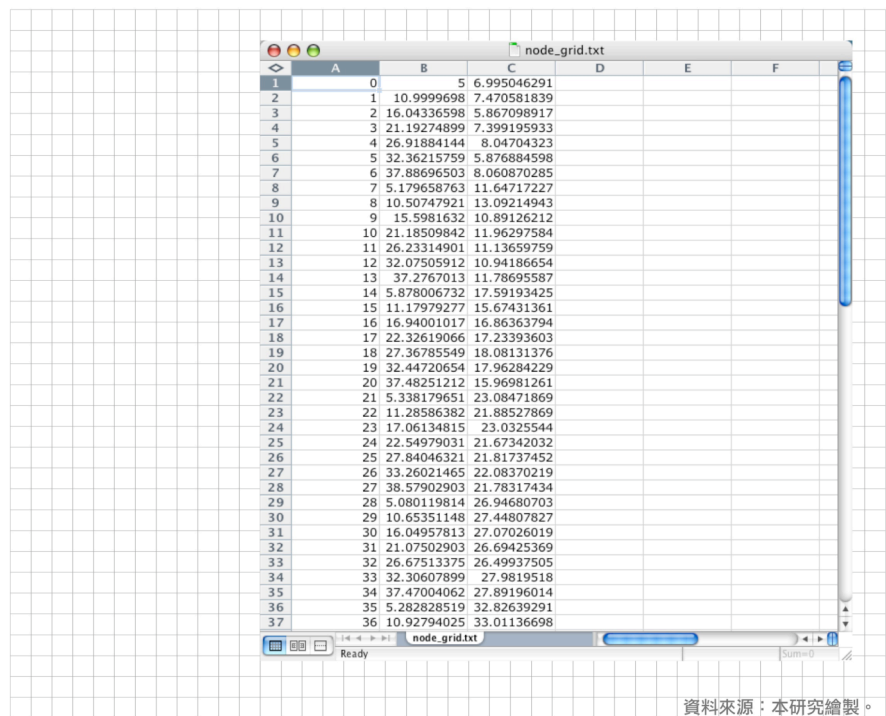


圖6.12 互援路網之演算流程

## 6.5 路網範例與操作

### 6.5.1 路網構建

由前述6.1節之點、線邊集和結構特性，首先須建立出點的獨立屬性，再透過邊的關係，將整個路網串連。為了易於直觀，應用程式將採用圖形化的方式呈現；點的外顯屬性，即為其座標。採用電腦螢幕第四象限為正的座標系統，點的輸入值包括點的標籤名稱、X座標、Y座標等三項；輸入方式可採用各類工程統計領域之商業軟體（本研究採用Microsoft Excel），以欄為屬性單位、列為資料單位，最後輸出成以「tab」為間隔之純文字檔；或可於任何文字編輯器中，以tab間隔屬性、換行區分單筆資料的方式，直接存成純文字檔案，以作為本分析軟體之輸入資料。



	A	B	C	D	E	F
1	0	5	6.995046291			
2	1	10.9999698	7.470581839			
3	2	16.04336598	5.867098917			
4	3	21.19274899	7.399195933			
5	4	26.91884144	8.04704323			
6	5	32.36215759	5.876884598			
7	6	37.88696503	8.060870285			
8	7	5.179658763	11.64717227			
9	8	10.50747921	13.09214943			
10	9	15.5981632	10.89126212			
11	10	21.18509842	11.96297584			
12	11	26.23314901	11.13659759			
13	12	32.07505912	10.94186654			
14	13	37.2767013	11.78695587			
15	14	5.878006732	17.59193425			
16	15	11.17979277	15.67431361			
17	16	16.94001017	16.86363794			
18	17	22.32619066	17.23393603			
19	18	27.36785549	18.08131376			
20	19	32.44720654	17.96284229			
21	20	37.48251212	15.96981261			
22	21	5.338179651	23.08471869			
23	22	11.28586382	21.88527869			
24	23	17.06134815	23.0325544			
25	24	22.54979031	21.67342032			
26	25	27.84046321	21.81737452			
27	26	33.26021465	22.08370219			
28	27	38.57902903	21.78317434			
29	28	5.080119814	26.94680703			
30	29	10.65351148	27.44807827			
31	30	16.04957813	27.07026019			
32	31	21.07502903	26.69425369			
33	32	26.67513375	26.49937505			
34	33	32.30607899	27.9819518			
35	34	37.47004062	27.89196014			
36	35	5.282828519	32.82639291			
37	36	10.92794025	33.01136698			

資料來源：本研究繪製。

圖6.13 點資料之輸入界面

首先，透過Excel之亂數函式功能，輸入一個約以5（無因次）為間隔的棋盤式佈設點資料。如圖6.13所示，欄位A為點的標籤名稱、欄位B為點的X座標、欄位C為點的Y座標，以一列為一筆資料，依序輸入直至所有的點資料輸入完畢，並建議將檔案儲存名為node\_grid.xls的格式；其中檔名前段node顯示該檔為點資料，檔名後段grid描述點資料之特性，副檔名xls則表示該檔案在此階段是由何軟體製作、以及是否已可作為本研究分析軟體之資料輸入格式。

本研究採以Microsoft Excel軟體介面做為點、線資料的輸入工具，之後再轉存為以tab為間隔之純文字檔。操作流程如圖6.14所示：

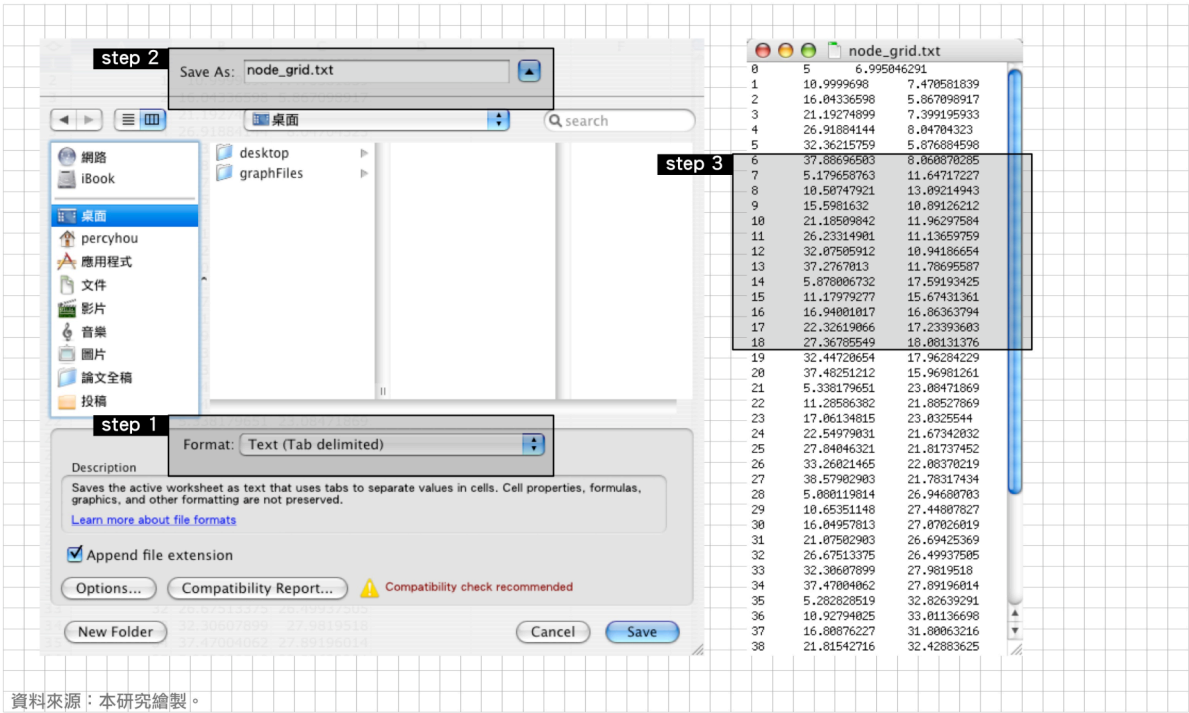


圖6.14 點資料轉檔方式

本分析軟體採用Java技術製作，輸入資料並採取純文字檔之通用格式，以便未來可應用於各種平台、各類作業系統環境之中。上圖之順序步驟，表示由Microsoft Excel製作之點資料，可透過格式轉換存成以tab分隔的純文字資料格式；也就是本分析軟體可輸入資料之格式。

為了縮減點、邊關係矩陣的大小，以提高輸入操作之正確性，本研究採取邊集合表示法來建構路網；因此邊的輸入值有邊的標籤名稱、邊某一側端點的點標籤名稱、邊另一側端點的點標籤名稱等三項屬性。輸入方式同樣採用各類工程統計領域之商業軟體，以欄為屬性單位、列為資料單位，最後輸出成以tab為間隔之純文字檔；也可於任何文字編輯器中，以tab間隔屬性、換行區分單筆資料的方式，直接存成純文字檔案，以作為本分析軟體之輸入資料。圖6.15即透過邊集合表示法，來展現出棋盤式路網之特性。

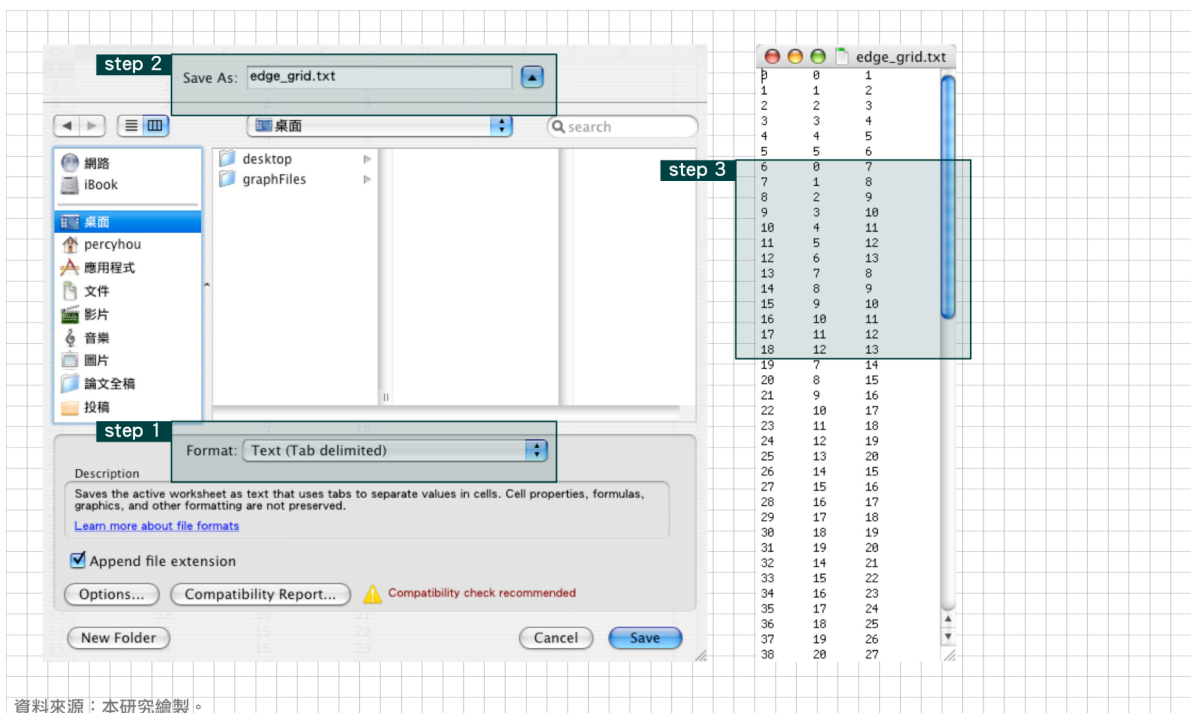
	A	B	C	D	E	F
1	0	0	1			
2	1	1	2			
3	2	2	3			
4	3	3	4			
5	4	4	5			
6	5	5	6			
7	6	0	7			
8	7	1	8			
9	8	2	9			
10	9	3	10			
11	10	4	11			
12	11	5	12			
13	12	6	13			
14	13	7	8			
15	14	8	9			
16	15	9	10			
17	16	10	11			
18	17	11	12			
19	18	12	13			
20	19	7	14			
21	20	8	15			
22	21	9	16			
23	22	10	17			
24	23	11	18			
25	24	12	19			
26	25	13	20			
27	26	14	15			
28	27	15	16			
29	28	16	17			
30	29	17	18			
31	30	18	19			
32	31	19	20			
33	32	14	21			
34	33	15	22			
35	34	16	23			
36	35	17	24			
37	36	18	25			

資料來源：本研究繪製。

圖6.15 邊資料之輸入界面

同樣地，本研究透過Excel之表格界面，輸入一個佈設為棋盤式關係的邊資料，欄位A為邊的標籤名稱、欄位B為該邊某一側端點的點標籤名稱、欄位C為該邊另一側端點的點標籤名稱，以一系列為一筆資料，依序輸入直至所有的邊資料輸入完畢，並建議將檔案儲存名為edge\_grid.xls的格式；其中檔名前段edge顯示該檔為邊資料，檔名後段grid描述邊資料之特性，副檔名xls則表示該檔案在此階段是由何軟體製作、以及是否已可作為本研究分析軟體之資料輸入格式。

接著，同樣採取Microsoft Excel製作之邊資料、透過格式轉換存成以tab分隔的純文字資料格式，以作為本分析軟體輸入之純文字檔通用格式資料；如圖6.16之順序步驟。



資料來源：本研究繪製。

圖6.16 邊資料轉檔方式



## 6.5.2 應用操作

### 1. 檔案輸入

首先，將對應路網的點線檔案放在同一資料夾之中，並於「common dir」欄位中輸入其位於電腦中之適當的位址；然後於「node file」欄位中輸入點的檔案名稱、「edge file」欄位中輸入邊的檔案名稱；最後鍵下輸入按鈕「import」；如圖6.17之左半部所示。此時，對應於路網結構的兩個檔案，將會以邊集合的圖形表示法架構出路網結構，並檢查該結構是否符合二邊連結（2ECON）條件，再透過點的座標屬性、以圖形的方式顯示於上方面板、以表格文字的方式顯示於左下面板；如圖6.17之右半部所示。

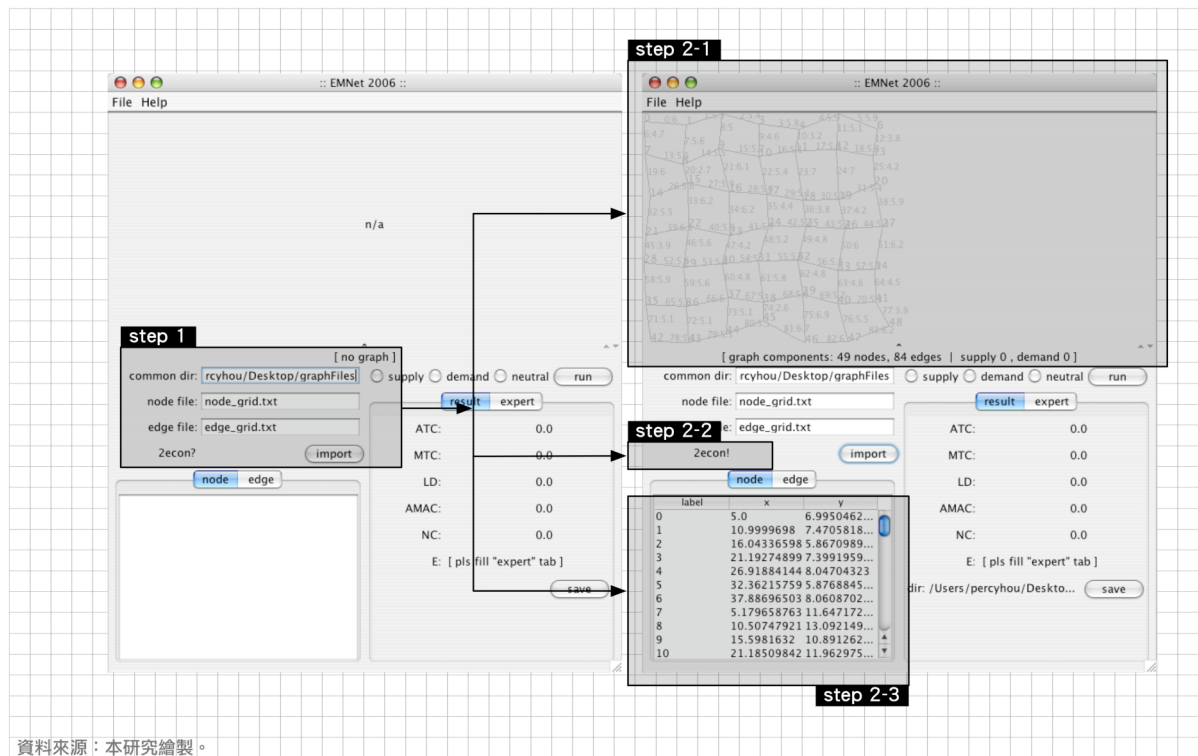


圖6.17 點邊資料輸入

## 2. 供需點設定

透過界面上供給（supply）、需求（demand）的選項設定後，以滑鼠直接於圖面之路網節點鍵入左鍵，便可設定供給點（紅色圓點）與需求點（藍色方點）；如圖6.18所示；此後再鍵下執行按鈕「run」，則責任分區、系統繞路、互援路網等三個路網模型演算法，便會依序執行。

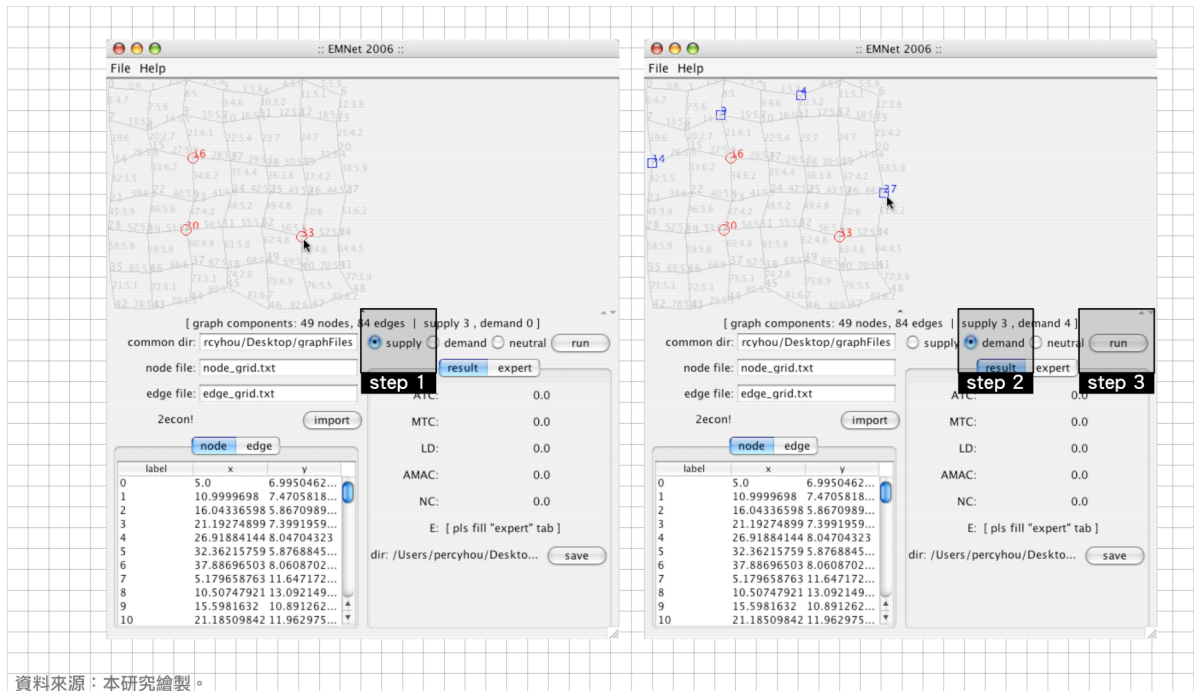
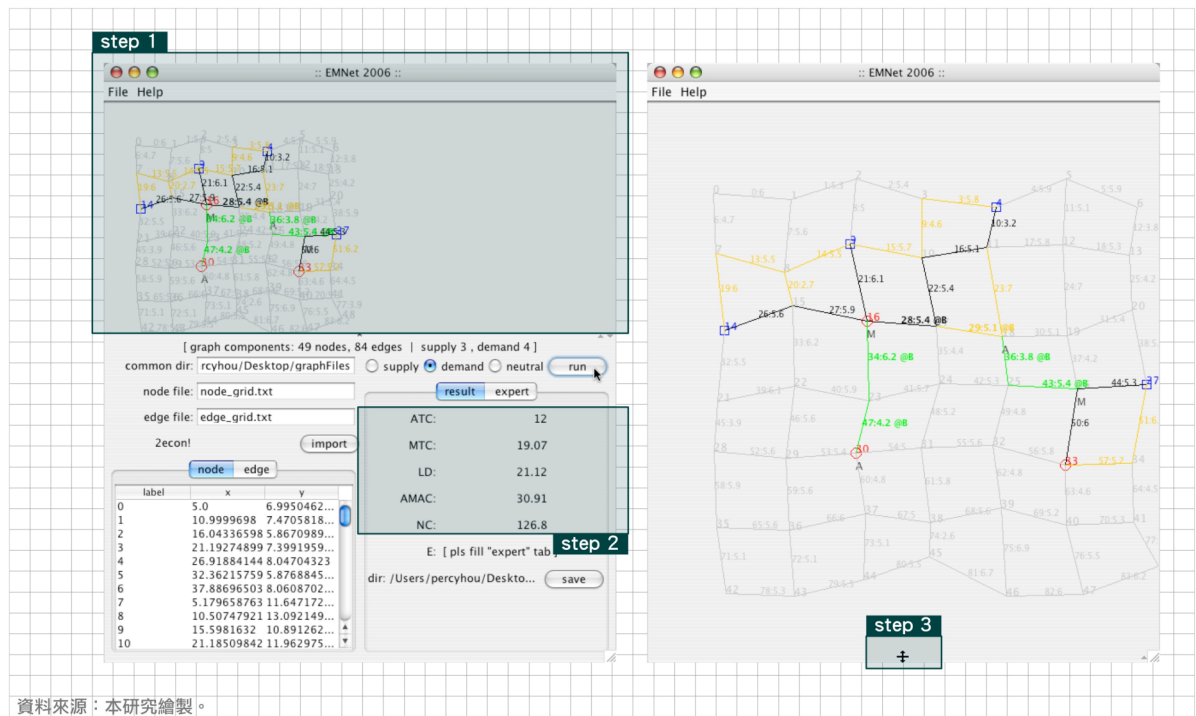


圖6.18 供需點設定方式

### 3. 路網演算與路網特性指標

透過第五章之防災路網模型與相關發展演算方法，根據所匯入之路網點線資料，則以黑色路網之責任分區路網、橙色路網之系統繞路路網，展開以救援單位為中心之各區域防災路網結構；最後不同供給單位所形成區域之間，則以綠色路網之互援路網結構連結；如圖6.19所示。

在演算結束時，同時計算獲致第五章所提出之路網相關特性指標值，如圖6.19左半部所示。



資料來源：本研究繪製。

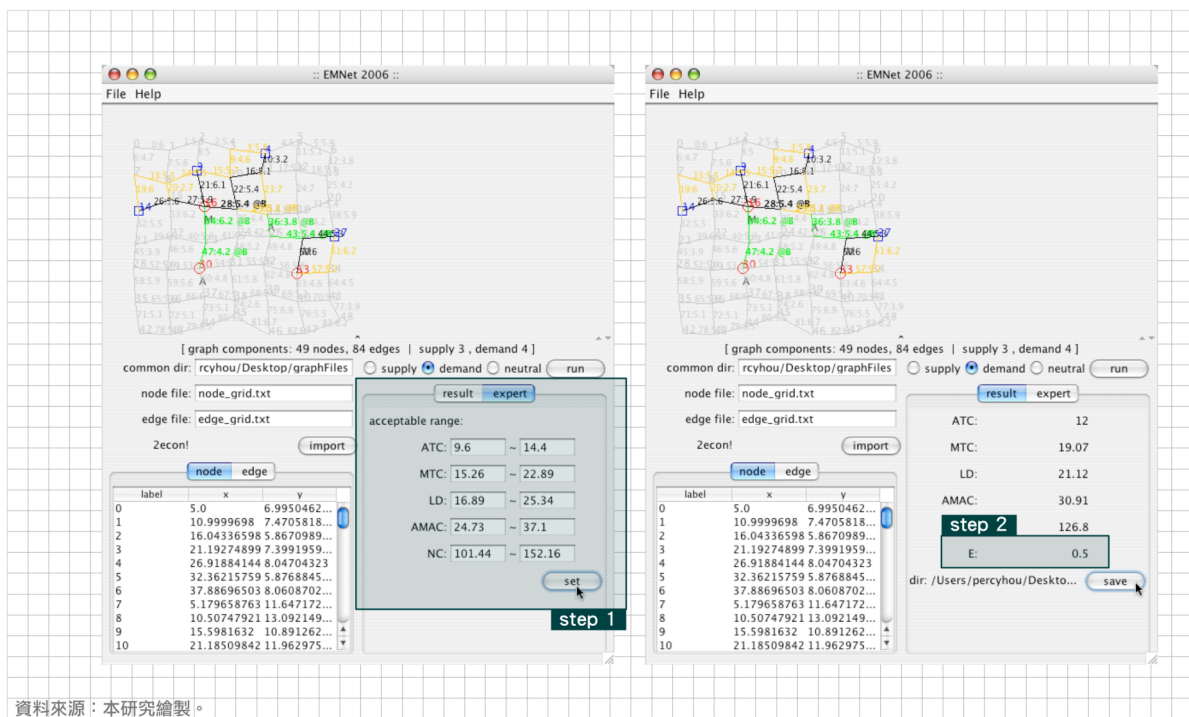
圖6.19 路網演算結果

#### 4. 專家意見與評估綜合指標

路網特性之綜合評估指標，係反應出可靠、全面效率、快捷三個構面，及其五個指標之特性；視各指標於其構面中具有相同權重，且各構面權重亦相同，並採用幾何平均之方法，加總平均以獲致綜合評估指標值。該指標值係採用：超過上界滿分、未足下界零分、上下界之間線性關係的方式來計算。而上下界限預設值為演算結果之120%與80%，因此在此預設上下界限範圍，路網綜合評估值為0.5；如圖6.20之左半部所示。然而，真正合適之上下界數值，應由專家綜合評估判斷後，分別取得並輸入適當欄位之中。

當專家完成所有指標上下界的輸入之後，最後再鍵下設定按鈕「set」；則路網的綜合評估就會依照各指標之上下界，透過線性關係計算出0~1的評估綜合指標值，而三構面中的各指標採以同比重之加權平均計算，三構面間亦採以等比重之加權平均計算。最後，回復到「result」的標籤頁面，便可得到專家對路網整體的綜合評價值「E」；數值愈高者，代表專家評價愈高，如圖6.20之右半部所示。

相關不同供給點情境之路網範例，請詳見附錄二。



資料來源：本研究繪製。

圖6.20 專家意見之輸入介面

## 第七章 棋盤式路網特性分析

### 7.1 測試路網

考量情境組合所能展現的精緻度、以及運作時間的可行性等因素，本研究採以7\*7之正方棋盤路網作為測試路網範例，來探討各種路網設計的效果，路網中點與點之間間隔採取5無因次單位。路網建構之方法同樣採取邊集合表示法，如圖7.1。邊集合中的每一邊都具有四種連結關係，包含了每一邊的兩個端點關係，以及為了掃描邊集合而存在與前、後邊元素的關係。然而另一方面，點集合中的每一個點，關係數將大於邊的關係數。由於本研究採用之路網皆為二邊連通圖，因此每一點至少存在與兩個邊以上的關係，以及為了掃描點集合而存在與前、後點元素的關係。

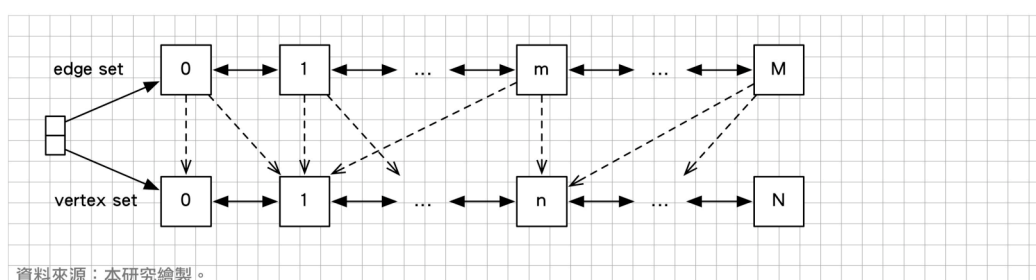


圖7.1 邊集合表示法

以下範例為測試7\*7正方棋盤測試路網的點、邊資料；此為理論路網，而針對實際路網則可採用實際座標。測試路網中，點與點之間採用等間距5無因次單位。構成邊集合表示法的點、邊資料如表7.1、表7.2所示。其中點資料包含三種主要屬性。表7.1中顯示，標籤為點的識別方法，並用以指認該點在路網中的唯一性；第四象限為正的 X、Y 座標，用以圖形化於地圖之中。表7.2中顯示，標籤為邊的辨識方法，並用以指認該邊在路網中的唯一性；邊兩端點的點標籤，用以建立該邊。

表7.1 棋盤式測試路網之點資料

點標籤	X 座標	Y 座標	點標籤	X 座標	Y 座標	點標籤	X 座標	Y 座標
0	5	5	17	20	15	34	35	25
1	10	5	18	25	15	35	5	30
2	15	5	19	30	15	36	10	30
3	20	5	20	35	15	37	15	30
4	25	5	21	5	20	38	20	30
5	30	5	22	10	20	39	25	30
6	35	5	23	15	20	40	30	30
7	5	10	24	20	20	41	35	30
8	10	10	25	25	20	42	5	35
9	15	10	26	30	20	43	10	35
10	20	10	27	35	20	44	15	35
11	25	10	28	5	25	45	20	35
12	30	10	29	10	25	46	25	35

13	35	10	30	15	25	47	30	35
14	5	15	31	20	25	48	35	35
15	10	15	32	25	25			
16	15	15	33	30	25			

表7.2 棋盤式測試路網之邊資料

邊標籤	點標籤1	點標籤2	邊標籤	點標籤1	點標籤2	邊標籤	點標籤1	點標籤2
0	0	1	28	16	17	56	32	33
1	1	2	29	17	18	57	33	34
2	2	3	30	18	19	58	28	35
3	3	4	31	19	20	59	29	36
4	4	5	32	14	21	60	30	37
5	5	6	33	15	22	61	31	38
6	0	7	34	16	23	62	32	39
7	1	8	35	17	24	63	33	40
8	2	9	36	18	25	64	34	41
9	3	10	37	19	26	65	35	36
10	4	11	38	20	27	66	36	37
11	5	12	39	21	22	67	37	38
12	6	13	40	22	23	68	38	39
13	7	8	41	23	24	69	39	40
14	8	9	42	24	25	70	40	41
15	9	10	43	25	26	71	35	42
16	10	11	44	26	27	72	36	43
17	11	12	45	21	28	73	37	44
18	12	13	46	22	29	74	38	45
19	7	14	47	23	30	75	39	46
20	8	15	48	24	31	76	40	47
21	9	16	49	25	32	77	41	48
22	10	17	50	26	33	78	42	43
23	11	18	51	27	34	79	43	44
24	12	19	52	28	29	80	44	45
25	13	20	53	29	30	81	45	46
26	14	15	54	30	31	82	46	47
27	15	16	55	31	32	83	47	48

## 7.2 供需點情境設計

由於實際路網之中，需求點位置不易預測；因此，在測試路網中，則假設需求點平均散布在路網之中，並以，即需求點所位在點標籤為0、3、6、21、24、27、42、45、與48之位置。表7.3為雙供給點之情境設計配置；採用雙供給點設計為互援路網之最低設計條件，因而採取之；區位設計因測試路網方正特性，可簡化雙供給點配置關係，實際組合情形，考量路網之對稱性，刪除重複之相對組合，篩選出總共136種不同之配置（詳見附錄一、附錄二）。圖7.2為棋盤式路網之樣態。

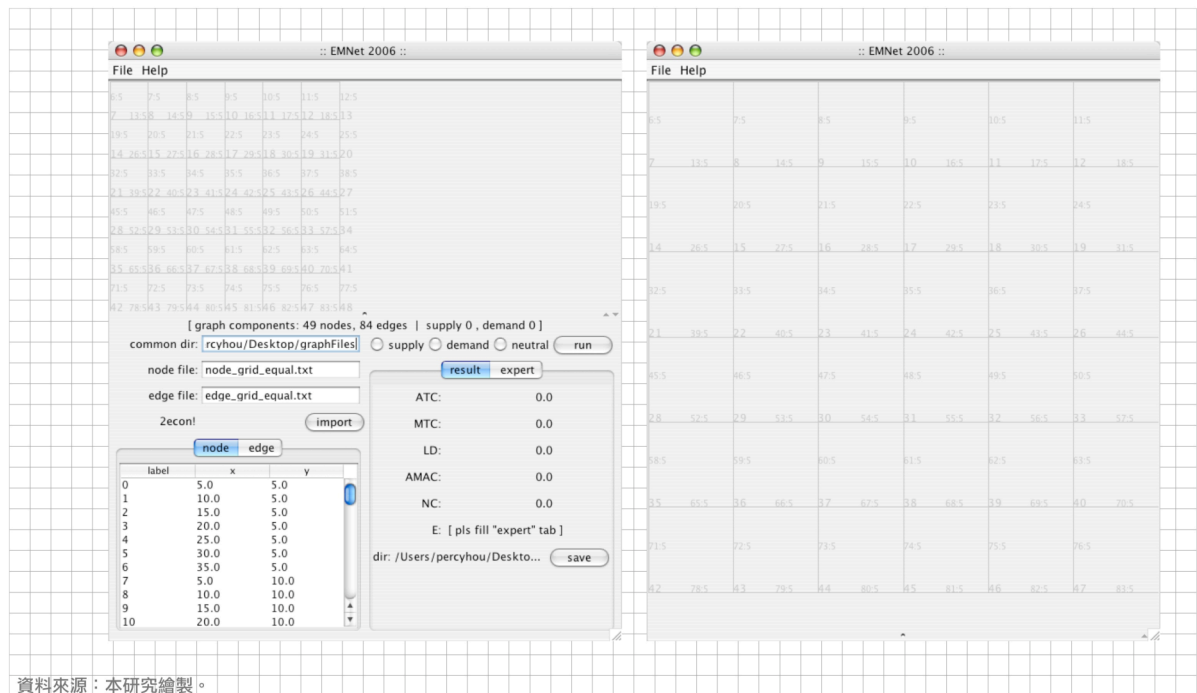


圖7.2 棋盤式測試路網

表7.3 雙供給點情境設計

供給1	供給2	供給1	供給2	供給1	供給2	供給1	供給2	供給1	供給2
40	8	7	10	7	41	14	8	14	43
40	15	7	11	7	43	14	9	14	44
40	16	7	12	7	44	14	11	14	46
40	29	7	13	7	46	14	12	14	47
40	36	7	14	7	47	14	13	17	1
40	30	7	15	10	1	14	15	17	2
40	37	7	16	10	2	14	16	17	8
40	32	7	17	10	8	14	18	17	9
40	39	7	18	10	9	14	19	17	10
33	9	7	19	10	14	14	20	17	14
33	11	7	20	10	15	14	22	17	15
33	15	7	22	10	16	14	23	17	16
33	16	7	23	10	22	14	25	17	22
33	18	7	25	10	23	14	26	17	23
33	19	7	26	10	28	14	28	17	28
33	29	7	28	10	29	14	29	17	29
33	30	7	29	10	30	14	30	17	30
33	32	7	30	10	31	14	31	17	31
33	37	7	31	10	35	14	32	17	35
33	39	7	32	10	36	14	33	17	36
32	16	7	33	10	37	14	34	17	37
32	18	7	34	10	38	14	35	17	38
7	1	7	35	10	43	14	36	17	43
7	2	7	36	10	44	14	37	17	44
7	4	7	37	14	1	14	38		
7	5	7	38	14	2	14	39		
7	8	7	39	14	4	14	40		
7	9	7	40	14	5	14	41		



### 7.3 測試路網之設計結果

本研究將雙供給設計情境之組合，將平均旅行成本、最大旅行成本、最大繞路成本、平均互援成本、路網成本趨勢，分別繪製如圖7.3、圖7.4、圖7.5、圖7.6、及圖7.7，由卡式（Cartesian）系統X軸表示供給點至路網中心點最短路徑之於總路網成本之比率，Y軸表示雙供給點之間的最短路徑之於總路網成本之比率，分別以此供給點至中心點的距離、以及彼此之間遠近的兩者關係，來共同表示出救援單位在整體路網之中的離散程度；Z軸單位與兩橫座標單位相同，分別為各指標所代表之意涵。

從圖7.3的趨勢可發現，平均旅行成本會隨著雙供給點之間的距離逐漸增加而遞減，這即表示相距愈遠的供給點配置，可使供給點到達各需求點的總成本減少，因此供給點分散，對於平均散布的需求點區域，相對而言更為快速。此外，平均旅行成本在雙供給點彼此距離近、而與中心點距離遠時，與彼此距離近、距中心點也近的情況相比，有增加的趨勢。

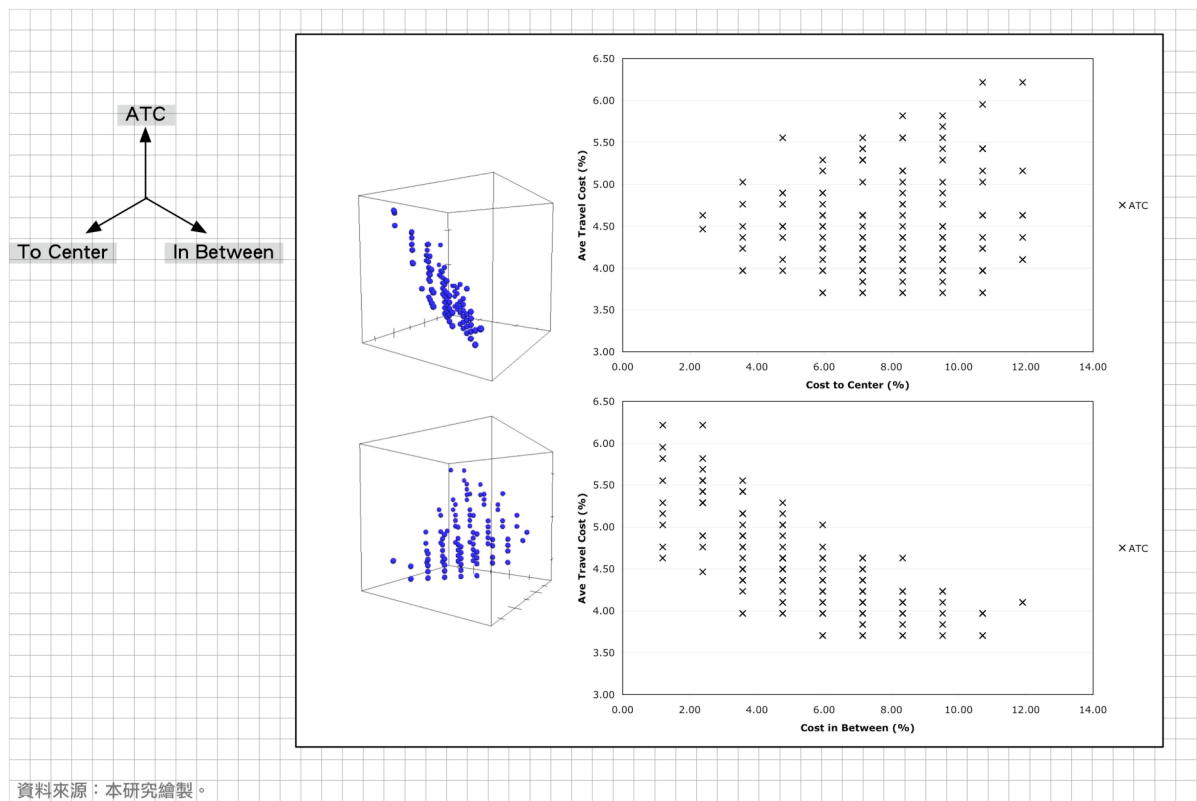


圖7.3 ATC之趨勢

圖7.4顯示，雖然最大旅行成本也有隨著雙供給點之間的距離增大而減少的趨勢，然而最佳的旅行成本在供給點相距5.95%時就已經出現，且供給點之間距離持續增加的情況下，仍有最佳的旅行成本的路網配置方案，分布較廣、趨勢較不明顯；而當兩供給點之間的距離達到原路網成本的10.71%左右（區域邊長的1.5倍）時，最大旅行成本減少趨勢漸緩，這顯示出此二供給點過於分散，已至整體路網的邊緣地帶，相對所有需求點而言，位處偏遠，反而容易產生某些需求點與供給點之間距離增加的可能性。而圖7.3中，平均旅行成本在距離達到10.71%（區域邊長的1.5倍）時，也同樣發生遞增的現象，顯示出雙供給點位處於路網邊界時，即使彼此距離分散，卻反而無法達到服務供給點最佳的位置。而由於對於救災旅次而言，最大旅行成本反應出某些供給地點的救災風險，因而在路網設計過程之中，應該特別關注。

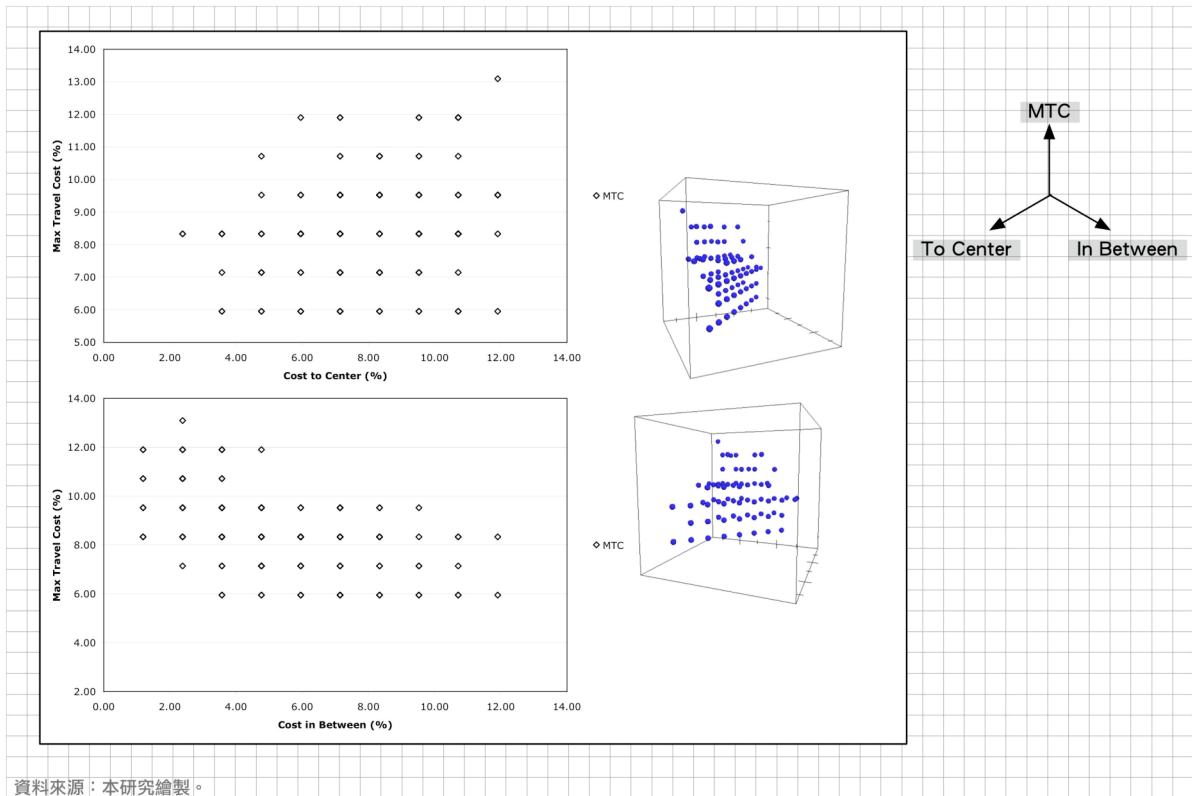


圖7.4 MTC之趨勢

圖7.5顯示雙供給組合設計下，繞路成本的變化趨勢。繞路成本隨著二供給點兩者之間位置的距離增加至4.76%時，該指標有整體驟降的現象，爾後減少的幅度則較不顯著。此顯示出，繞路成本之降低在二供給配置距離相近時，有很高的效果，隨著兩者供給位置逐漸擴大，雖仍可微幅降低繞路成本，卻沒有很明顯的效益。而在兩供給點位置相近時，散亂程度較高，而無論兩供給點距離中心點之遠近，變異的程度都很高，顯示出繞路成本與供給點距離中心點之遠距較沒有明顯關係，而與二供給點彼此之間的距離比較有關。

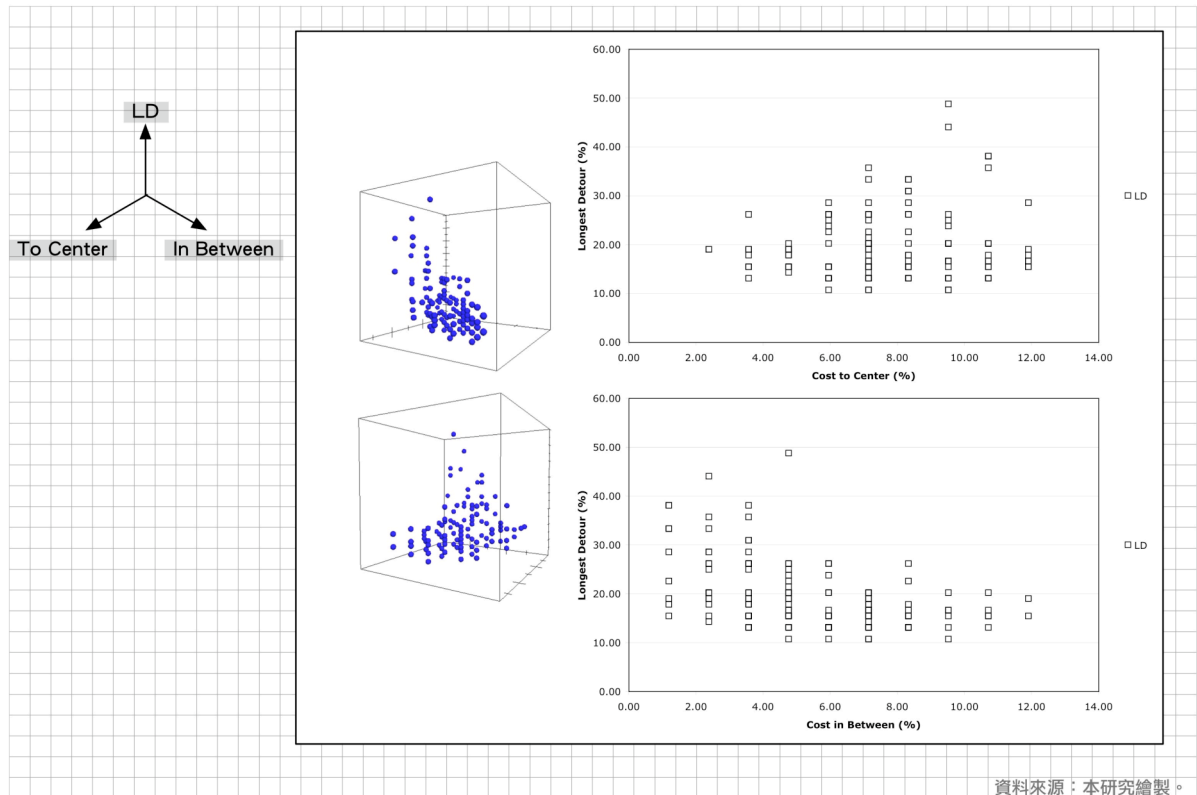
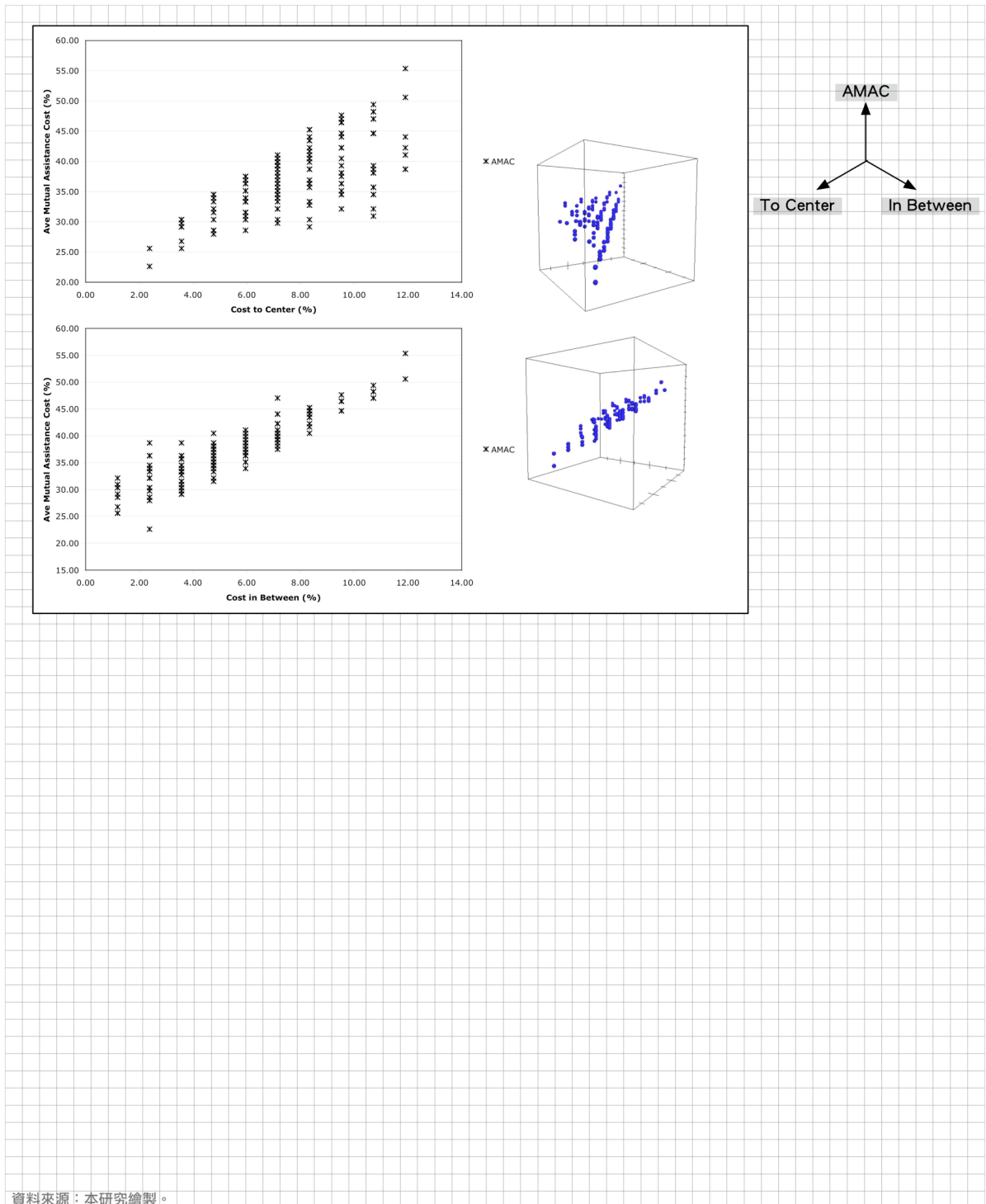


圖7.5 LD之趨勢

圖7.6顯示出互援成本與二供給點之間清楚的脈絡關係，隨著供給點之間的距離增加，互援成本呈現正相關的穩定增長。互援路網乃透過備援供給點連結至不同區域的需求點，以達成路網供給連結度之功能；因此，互援成本會隨著供給點遠離其他需求區域，而逐漸增加。同樣的趨勢也發生在供給點距離中心位置，距中心點之距離（ $X$ ）、供給點之間的距離（ $Y$ ）與互援成本指標（ $Z$ ）之間，具有 $Z = c - aX - bY$ 之勢，顯示相同互援成本下，二供給點彼此距離以及距中心點距離，具有負向關係。



資料來源：本研究繪製。

圖7.6 AMAC之趨勢

圖7.7顯示，路網成本隨著二供給點之間距離的增加，漸有穩定減小並收斂之趨勢。這樣的結果，可以顯示出當兩個供給點之間距離較近時，兩者配置位置之組合對路網結構影響顯著，因此無法輕易確認比較出優劣，但隨著二供給點相距較遠時，供給點彼此之間配置關係對路網的影響就逐漸減少。因此，若要獲得節省的路網結構，在供給點設置距離較近時，應測試並考量多種位置的組合，才能得到較好的結果。

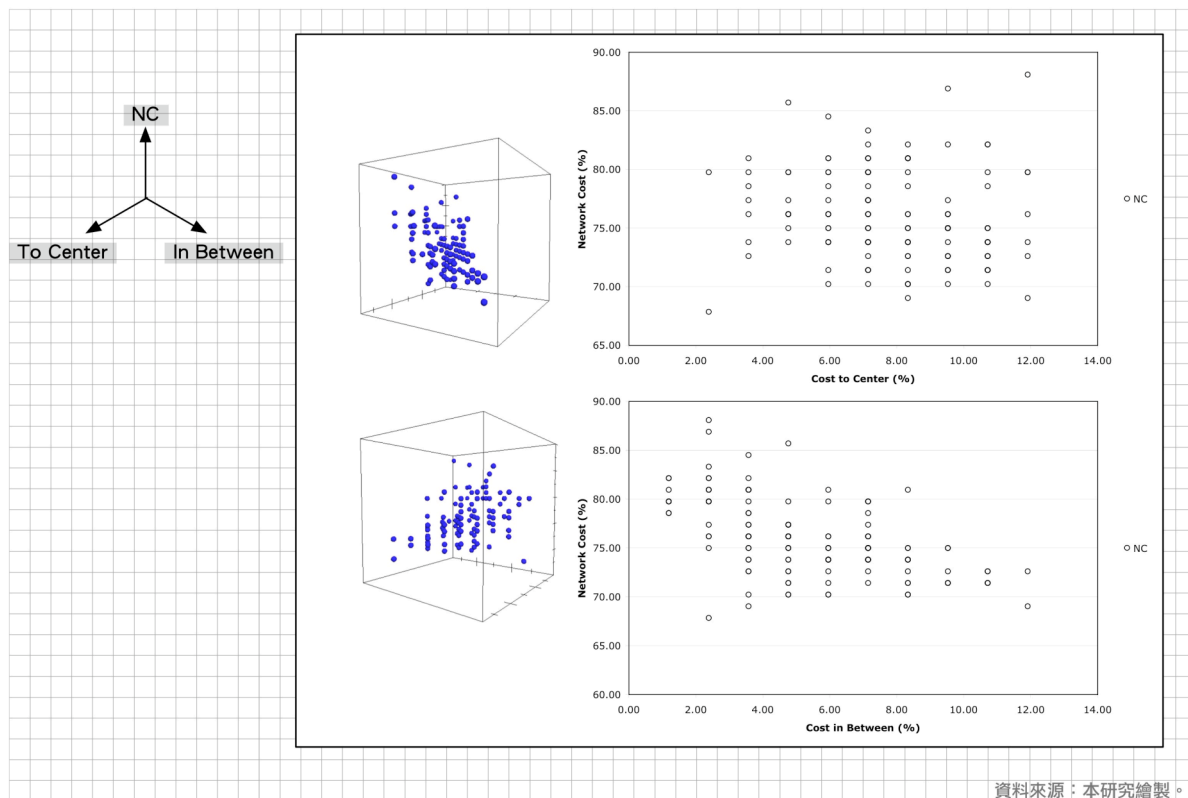


圖7.7 NC之趨勢

綜合以上之觀察發現，隨著供給點相距愈遠，平均旅行成本及最大旅行成本逐漸降低，而繞路成本也會隨之下降；平均互援時間隨著雙供給點之間的距離增加而遞增，兩者之間有正相關的影響，此乃由於平均互援時間為量測不同區內所有需求點至區外供給點的總成本；而供給點乃為影響區域所在的主要因素，最短路徑森林之路網結構會使得需求點儘可能的趨近其供給點，因此，供給點也相對能代表眾多需求點之所在，所以兩供給點之間的距離增加，也就某種程度表示了平均互援時間的增加。

此外，在二供給點之間的距離達到5.95%（此測試正方路網邊長的六分之五）以後，最大旅行成本與繞路成本都有趨向變化減小的勢態，由於供給點假設均勻分布該路網之中，因此幾何特性應該為造成路網特性的因素之一；至於平均旅行成本、路網成本，仍會隨著二供給點之間的距離增加而持續降低，平均互援時間則隨之持續增高，此情境模擬之供給點配置方式結果，可提供未來在考量方正區域、需求點均勻散布之情形之參考。

## 7.4 綜合建議

總結以上結果，雙供給點、均勻需求、正方棋盤式區域之防災存活路網設計，在特定情形下，如欲獲得較好的效果，可採用以下簡要建議：

- （1）平均旅行成本：欲規劃整體區域最小的平均旅行成本，供給單位之間的距離，約為該正方區域邊長的0.83~1.5倍之間。
- （2）最大旅行成本：欲控制最大旅行成本，供給單位之間的距離，在該正方區域邊長約為0.5倍以上時，即會出現該成本最小之情形。
- （3）繞路成本：繞路成本最小的情形，發生在供給單位之間的距離達該正方區域邊長的0.67~1.33倍之間，而0.67倍之距離效益最佳。
- （4）互援成本：互援成本隨供給單位之間的距離成正關係，供給單位彼此之間愈靠近，該成本就愈小。
- （5）路網成本：整體規劃路網成本最小發生在供給單位之間距離為該正方區域邊長0.33倍時，而隨著供給單位彼此之間愈靠近，該成本平均而言也會跟著降低。

## 第八章 結論與建議

目前尚無可準確預測地震的機制，因此，透過路網規劃的方法，來減低災害所帶來的不確定性便相形重要。本研究首先採取了道路的觀點，以關鍵性、空間評估的方式，找出空間良好、並具有重要性的個別路段，以作為納入為防災道路的候選路段之法，再以整體路網的觀點，採用容錯的存活路網設計結構，以達到地震減災的目的；在整體路網中，並引進了許多想法，例如路網連結度、共同繞路橋段、多供給互援單位等；本研究並分析路網結構，建立了路網模型，並提出其演算法，最後，並採用棋盤式路網進行分析，以顯現其實用性。

本研究之獨特性之一，在於提出系統繞路的觀點，來設計存活路網。地震減災的路網設計問題，對於提供可靠、快捷的運輸服務將扮演很重要的角色；在本研究中，共同橋段可用以確認最小的系統繞路成本，此亦為一新的方法來構建二邊連接圖。該路網結構顯示了許多優點，並可建立更為安全的路網系統，保護民眾免於地震災害所帶來的許多不確定性。

### 8.1 結論

（1）本研究提出一套規劃防災路網之流程，提供相關政府規劃部門，以應用於長期建構安全防災都市之規劃與評估。對於現存都市路網，透過路段關鍵性與道路空間評量等方法，篩選綜合出重要適宜之初始防災路網，再經由路網設計模型與專家決策系統等過程，提出達成多目標向度防災路網結構之建議；該路網設計模型並可運用於新興都市之防災路網結構。

（2）本研究提出評估路段關鍵性之方法。城市路網之中，每個路段可被使用的方式與頻次不同，針對救災單位與避難點考量所可能採用的救災路線，透過適當指標評估出可能使用頻次較高之路段，以作為篩選防災路網之考量因素。

（3）本研究提出三維道路空間評估概念。將路段空間劃分為道路空間和近鄰空間，近鄰空間再分為上部空間與側部空間，透過空間屬性區分以利進行現有道路之調查評量；提出二次災害風險衡量因子，包括道路寬度、電線密度、建築物高度、人行道寬度等，以作為篩選防災路網之考量因素。

（4）本研究構建防災路網之設計模型。防災城市之中，防災路網為聯繫各避難系統間運作之重要媒介；近年來因災害議題逐受重視，並訂定相關防災道路準則，然而防災路網供需點間的設計模型，宜以整體性之考量，因此，本研究以可靠性、快捷性、全面性三目標，提出路網設計模型以供規劃參考。

（5）最短路徑森林演算法劃分多供給點責任分區。本研究依救援路徑成本方式來考量合理之路網分區，將比依行政區劃分救援分區更具時間效率。在演算法上，運用 **Dijkstra** 最短路徑演算法，採取平行多工演算法，縮減重複演算時間，以快速計算出多供給點至多需求點的最短路徑森林。

(6) 容錯設計。以原則性之路網規劃設計準則來決定防災路網，將較無法保障替代道路之存在；然而本研究將問題以限制式性質加以控制，則可確保路段毀損情形下，存在替代路徑。此外，除因此存在路網容錯特性外，本研究並納入繞路成本考量，以避免雖有替代路徑，卻繞路費時之情形發生；且，改進往昔僅考量單點對單點之繞路成本研究，路網演算過程納入系統繞路成本指標，此乃先前研究之所未見，以建構涵蓋全面多需求點之繞路成本最小化之路網結構。

(7) 互援路網設計。防災路網設計，結合供給點與需求點之考量，使防災路網非獨立設計，而使其結構與救援單位、避難據點之關連緊密結合；此外，本研究提出二供給點連結之觀念，落實，以確保防災路網分區間之供給點連結性，使防災路網設計中，除了考量路段容錯問題外，也提升了供給點容錯的可靠性。

(8) 本研究建立防災路網決策工具。不同城市之路網具有不同特性，所設立的標準也應有所差異，本研究針對可靠、快捷、全面效率等目標，擬定相關準則指標，將路網特性量化觀察，以數據的方式，讓決策者更容易了解路網結構，將有助於規劃決策之過程。

(9) 本研究之防災路網工具拓展理論路網與實務路網之應用性。基本輸入資料為以邊集合架構之點、邊的兩個純文字檔；點資料文字檔為以Tab為間隔，包含點標籤、X座標、Y座標三個欄位，邊資料文字檔為以Tab為間隔，包含邊標籤、該邊兩端點的點標籤三個欄位；互動輸入資料為供給點、需求點之個數與位置，以及專家針對不同指標所設定之上下界限值；輸出資料為圖形化之防災路網結構，平均旅行成本、最大旅行成本、區內最長繞路成本、區間平均互援成本、路網成本等五個指標值，以及針對專家所設定上下界限值後所獲得的防災路網總體評估值。

(10) 在雙供給均勻需求的棋盤式路網情境中，本研究模擬136種不同的防災存活路網之設計，並發現需求點均勻分布且供給點之設置相互逐漸遠離時，平均旅行成本趨勢下降。隨著供給點相距愈遠，各供給點之責任分區界限漸為明確，平均旅行成本逐漸降低，表示供給單位處於需求點中心區位。

(11) 相同情境中，隨供給點彼此逐漸遠離，繞路成本有陡降漸緩之趨勢。此乃顯示供給點之間配置距離，短距離內對繞路成本的敏感度高，亦即表示此範圍內擴大供給點間的距離，對減少繞路成本有明顯的效果；互援成本則隨著雙供給點之間的距離增加而遞增，因此當救援單位供給量能較小時，應特別考量互援單位之設置位置，以避免供給上之高風險。

(12) 雙供給點組合情境之路網結果顯示，需求點均勻分布且供給點相近時，供給點配置組合影響路網成本明顯。透過本研究之情境模擬分析結果，顯示出當供給點之間的距離較近時，配置位置之組合對路網結構影響較顯著，因此如要獲得成本較為低廉的路網結構，應要考量多種配置組合。



(13) 模擬結果顯示，欲規劃整體區域最小的平均旅行成本，供給單位之間的距離，約為該正方區域邊長的0.83~1.5倍之間；供給單位之間的距離，在該正方區域邊長約為0.5倍以上時，最大旅行成本即會出現該成本最小之情形；繞路成本最小的情形，發生在供給單位之間的距離達該正方區域邊長的0.67~1.33倍之間、0.67倍之距離效益最佳；互援成本隨供給單位之間的距離成正關係；路網成本最小發生在供給單位之間距離為該正方區域邊長0.33倍時，而隨著供給單位彼此之間愈靠近，該成本平均而言也會跟著降低。

## 8.2 建議

囿於本研究之主題性及篇幅，下列為本研究認為未來值得探究之防災路網設計與規劃相關之議題：

(1) 供給單位、需求單位之規模評估。目前本研究採取各供給點、需求點之供給與需求量能相同之模型，然而未來可研究評估各供給單位救援規模之方法，進一步考量其救援能力之差異性，並根據周圍土地使用強度、人口密度、日夜活動之不同，探討需求點之屬性差異，並整合於路網規劃之中，將可使救援分區具權數衡量之特性，對於路網隱含之救援意義，多有助益。

(2) 防災道路的基礎設施設備之規劃與落實。關於避難地點或收容場所，指揮中心、臨時安置空間、醫療運作空間、儲備水源、物資輸送內部路線等準備，都是相當重要的，然而，防災道路與一般道路的設施設備應有何差異，卻鮮少被討論，例如，應將防災路網之功能與供給單位、救災中心之運作緊密結合，且如能透過針對災害蒐集防災路網動態情報的相關設備，將可進一步掌握防災路網的交通資訊狀況，並於災害期間，則可對救災路線規劃、旅次的控管，快速提出更有效用之決策。

(3) 評量防災路網使用面的方法。本研究提出道路三維空間、路段關鍵性、以及諸多路網指標，於實際使用面及理論幾何面，來量化防災路網空間及路線之品質表現；而未來對於更細緻的運行操作與管制面，如與平日交通量對路網在救援上的影響，值得後續研究深入探討。此外，防災道路設計標準之訂定亦相當重要，防災路網為防災都市之骨幹，其結構、高程、寬度、車道佈設等，都應更加提高其救災時之可靠專用性，乃至於平日交通管理策略，如停車管理、三維空間障礙物管制等，應提出一套明確可依循的準則，以加強都市之防災機能。



## 參考文獻

1. 丁育群、蔡綽芳，「九二一震災對都市空間防災規劃問題探討」，工程月刊，第七十三卷第九期，25~36頁，民國89年。
2. 土井幸平，「阪神大震災復興計劃之概要與都市、建築之課題」，第十六屆中日工程技術研討會，4-1~4-7頁，台北，民國84年。
3. 毛利正光、塚口博斯、中村俊策，「住宅區內步行路網之評估」，土木學會關西支部年次學術演講會演講概要，日本，1981。
4. 王元元、王慶瑞、黃紀麟，組合數學，中央圖書，台北，民國89年。
5. 內政部建築研究所，「921集集震災都市防災調查研究報告」，民國88年。
6. 台北市建築師工會，兵庫縣南部（阪神）地震「驗證」報導彙編，民國84年。
7. 台灣科技大學建築系，「台北市實質環境防災機能之研究」，國科會，民國88年。
8. 交通部公路局，九二一地震後中橫公路谷關至德基路段功能定位，民國89年。
9. 交通部公路局，中橫公路開放通車前之各替代道路方案改善勘查報告彙編，民國89年。
10. 任善強、周寅亮，數學模型，中央圖書，台北，民國86年。
11. 吳永隆、葉光毅，「地區性道路網便利性、舒適性、防災性基準之研究」，建築學報，第二十六卷，19~34頁，民國87年。
12. 吳永隆、葉光毅，「地區性道路網便利性、舒適性、防災性基準之研究」，建築學報，第二十六期，19~34頁，民國87年。
13. 何明錦、李威儀，從都市防災系統檢討實質空間之防災功能—（一）防救災交通動線系統及防救據點，內政部建築研究所，民國87年。
14. 何明錦、黃定國，都市計劃防災規劃作業之研究，內政部建築研究所，民國86年。
15. 李威儀、錢學陶，從都市防災系統中實質空間防災功能檢討—（二）學校、公園及大型公共設施等防救據點，內政部建築研究所，民國88年。
16. 李威儀、何明錦，台北市實質環境防災機能之研究，行政院國家科學委員會，民國88年。
17. 李威儀、錢學陶、李咸亨，台北市都市計劃防災系統之規劃，台北市政府都發局，民國86年。
18. 李威儀，都市計劃防災空間系統規劃之研究，都市防災論文集，內政部建築研究所，民國88年。
19. 李得全、謝佩砒、朱南玉，都市計劃防災規劃實務案例-以台北市南港區通盤檢討為例，都市防災論文集，內政部建築研究所，民國88年。
20. 宋增民、殷翔，作業研究，中央圖書，台北，民國90年。
21. 林峰田，「公共設施檢討空間分析方法」，都市與計劃，第二十四卷第二期，171~192頁，民國86年。
22. 林峰田，「國土城鄉防災綱要計畫」，內政部營建署，民國92年。
23. 林峰田、李佳昀，「地震防救災文獻案例式查詢系統」，都市與計劃，第二十七卷第一期，65~80頁，民國89年。
24. 林峰田、陳亮全，都市災害危險度評估網格資訊系統之建立—避難空地配置評估方法，內政部建築研究所，民國87年。
25. 林建元、張璠，「台北市消防站系統之區位分析」，規劃學報，第十八期，29~51頁，民國80年。
26. 周寅亮，離散數學，中央圖書，台北，民國87年。
27. 室崎益輝，「阪神大震災後的復興計劃」，第十六屆中日工程技術研討會，3-1~3-11頁，台北，民國84年。
28. 施鴻志、周士雄，都市計劃，建都文化，民國85年。

29. 施鴻志、陳長庚，我國都市防災因應對策之研究，都市防災論文集，內政部建築研究所，民國88年。
30. 徐淵靜、侯鵬曦，「震災時都市道路系統運輸功能評估與防災路網之研擬」，國立交通大學，碩士論文，民國90年。
31. 徐淵靜、侯鵬曦，「防災路網模式構建」，第十八屆中華民國運輸研討會，新竹，民國92年。
32. 徐淵靜、侯鵬曦，「高雄市防災路網研擬之研究—地理資訊系統之應用」，中華道路，第四十二卷第二三四期，10-20頁，民國92年。
33. 孫志鴻、詹仕堅、鄒明城、謝奇峰，「地理資訊系統在防救災上的應用」，土木技術，第一卷第二期，111~127頁，民國87年。
34. 陳坤章，高雄市推動建築物防火管理制度及實施成效之研究，高雄市政府消防局，民國88年。
35. 陳亮全、林文苑、賴美如，我國災害境況模擬評估系統之現況與課題，都市防災論文集，內政部建築研究所，民國88年。
36. 陳建忠、詹士樑，都市地區避難救災路徑有效性評估之研究，內政部建築研究所，民國88年。
37. 陳建忠、黃定國、黃志弘，都市計劃通盤檢討有關防災規劃作業程序及設計準則之研究，內政部建築研究所，民國88年。
38. 張益三，都市防災規劃之研究，台灣省政府住宅與都市發展處市鄉規劃局，民國88年。
39. 都市計劃作業規劃手冊第捌冊都市防災計劃，台灣省政府住宅與都市發展處市鄉規劃局，民國87年。
40. 馮正民、林楨家，都市及區域分析方法，建都文化，民國89年。
41. 馮正民、解鴻年，「緊急設施區位模式之評述」，交通運輸，第十一期，27~38頁，民國78年。
42. 葉光毅、吳永隆，地區性交通計劃，滄海書局，民國87年。
43. 黃定國，安全都市之建立與防災都市整備計劃之研究，都市防災論文集，內政部建築研究所，民國88年。
44. 曾國雄、林楨家，「淡海新市鎮消防隊佈設區位之研究」，都市與計劃，第二十四卷第一期，81~98頁，民國86年。
45. 韓復華、卓裕仁，「緊急疏散公用車輛調派之研究」，運輸計劃季刊，第二十三卷第三期，247~272頁，民國83年。
46. 韓復華、胡大瀛，「路網疏散模式研究與微電腦決策輔助系統之建立」，運輸計劃季刊，第十六卷第三期，民國76年。
47. Adam, D., *Data Structures and Algorithms in Java*, 2001.
48. Amin, A.T., Siegrist, K.T. and Slater, P.J. "Pair-connected reliability of communication networks with vertex failures", Congressus Numerantium, 67, pp. 233-42, 1988.
49. Asakura, Y. "Reliability measures of an origin and destination pair in a deteriorated road network with variable flow", Transportation Networks: Recent Methodological Advances (ed. Bell, M.G.H.), Pergamon Press, Oxford, 1996.
50. Bagga, K.S., Beineke, L.W., Lipman, M.J. and Pippert, R.E. A survey of integrity. Technical Report. Department of Mathematical Sciences, Indiana University-Purdue University at Fort Wayne, 1989.
51. Baiou, M. "On the dominant of the Steiner 2-edge connected subgraph polytope", Discrete Applied Mathematics, 112 (1-3), pp.3-10, 2001.
52. Ball, M.O., Golden, B.L. and Vohra, R.V. "Finding the most vital arcs in a network", Operation Research Letters, 8, pp. 73-6, 1989.
53. Barefoot, C.A., Entringer, R. and Swart, H. "Vulnerability in graphs – a comparative survey",

- Journal of Combinatorial Mathematics and Combinatorial Computing, 1, pp. 12-22, 1987.
54. Barlow, R.E. "Mathematical theory of reliability: a historical perspective", IEEE Transactions on Reliability, R-33, pp. 16-20, 1984.
  55. Barlow, R.E. and Singpurwalla, N.D. "Assessing the reliability of computer software and computer networks: an opportunity for partnership with computer scientists", The American Statistician, 39, pp. 88-94, 1985.
  56. Beineke, L.W. Explorations into graph vulnerability. Technical Report. Department of Mathematical Sciences, Indiana University-Purdue University at Fort Wayne, 1989.
  57. Beineke, L.W. and Harary, F. "The connectivity function of a graph", Mathematika, 14, pp. 197-202, 1967.
  58. Bell, M.G.H. "A game theory approach to measuring the performance reliability of transport networks", Transportation Research Part B, 34, pp. 533-545, 2000.
  59. Bell, M.G.H., Cassir, C., Iida, Y. and Lam, W.H.K. "A sensitivity based approach to network reliability assessment", Proceedings of the 14th International Symposium on Transportation and Traffic Theory (ed. Ceder, A.), pp. 283-300, Pergamon, Oxford, 1999.
  60. Bellman, R. "On a Routing Problem", Quarterly of Applied Mathematics, 16 (1), pp. 87-90, 1958.
  61. Berdica, K. "An introduction to road vulnerability: what has been done, is done and should be done", Transport Policy, 9, pp. 117-127, 2002.
  62. Boesch, F.T. "Synthesis of reliable networks - a survey", IEEE Transactions on Reliability, R-35, pp. 240-6, 1986.
  63. Boesch, F.T., Harary, F. and Kabell, J.A. "Graphs as models of communication network vulnerability: connectivity and persistence", Networks, 11, pp. 57-63, 1981.
  64. Bruno, R.P. Data structures and algorithms with object-oriented design patterns in Java, 2000.
  65. Chen, A., Yang, H., Lo, H.K. and Tang, W. "A capacity related reliability for transportation networks", Journal of Advanced Transportation, 33 (2), pp. 183-200, 1999.
  66. Chvatal, V. "Tough graphs and Hamiltonian circuits", Discrete Mathematics, 5, pp. 215-28, 1973.
  67. Colbourn, C.J. "Network resilience", SIAM Journal on Algebraic and Discrete Methods, 8, pp. 404-9, 1987.
  68. Corley, H.W. and Sha, D.Y. "Most vital links and nodes in weighted networks", Operation Research Letters, 1 (4), pp. 157-160, 1982.
  69. Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. Introduction to Algorithms, McGraw Hill, USA, 2001.
  70. David, M.G. Graphic Java 2: Mastering the JFC. Swing, 1999.
  71. D'Este, G.M. and Taylor, M.A.P. "Network vulnerability: an approach to reliability analysis at the level of national strategic transport networks", The Network Reliability of Transport (eds. Bell, M.G.H. and Iida, Y.), pp. 23-44, Pergamon Press, Oxford, 2003.
  72. D'Este, G.M. and Taylor, M.A.P. "Modelling network vulnerability at the level of the national strategic transport network", Journal of the Eastern Asia Society for Transportation Studies, 4 (2), pp. 1-14, 2001.
  73. Dijkstra, E. W. (1959) "A note on two problems in connexion with graphs", Numerische Mathematik, 1, pp. 269-271, 1959.
  74. Douglas, B.W. Introduction to graph theory, Prentice-Hall, USA, 2001.

75. Feng, C.M. and Wang, T.C. "Highway emergency rehabilitation scheduling in post-earthquake 72 hours", Journal of the Eastern Asia Society for Transportation Studies, 5, pp. 3276-85, 2003.
76. Feng, C.M. and Wen, C.C. "Traffic control management for earthquake-raided area", Journal of the Eastern Asia Society for Transportation Studies, 5, pp. 3261-75, 2003.
77. Floyd R.W. "Algorithm 97 (SHORTEST PATH)", Communications of the ACM, 5 (6), p. 345, June 1962.
78. Garey, M.R. and Johnson, D.S. Computers and intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, California, 1979.
79. Gibbons, A. Algorithmic Graph Theory, Cambridge University Press, New York, 1985.
80. Glenn, R. An introduction to data structures and algorithms with Java, 1997.
81. Goemans, M.X. and Bertsimas, D.J. "Survivable networks: linear programming relaxations and the parsimonious property", Math. Program., 60 (2), pp. 145-166, 1993.
82. Grotschel, M., Monma, C.L. and Stoer, M. "Design of survivable networks", Handbooks in Operation Research and Management Science Volume 7 (eds. Ball, M.O., Magnanti, T.L., Monma, C.L. and Nemhauser, G.L.), Elsevier Science, Amsterdam, 1995.
83. Gunay, B. "Modeling lane discipline on multilane uninterrupted traffic flow", Traffic and Engineering Control, 40 (9), pp. 440-447, 1999.
84. Hart, P.E., Nilsson, N. J. and Raphael, B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on Systems Science and Cybernetics, SSC 4 (2), pp. 100-107, 1968.
85. Hart, P.E., Nilsson, N.J., Raphael B. "Correction to A Formal Basis for the Heuristic Determination of Minimum Cost Paths", SIGART Newsletter, 37, pp. 28-29, 1972.
86. Hou, P.H. and Hsu, Y.C. "Finding dominant links of emergency network with respect to earthquake disaster", Journal of the Eastern Asia Society for Transportation Studies, 6, pp. 77-90, 2005.
87. Hou, P.H. and Hsu, Y.C. "Rescue network design with respect to earthquake", Proceedings of International Symposium City Planning, 2004, September, Sapporo, Japan, pp. 445-52, 2004.
88. Iida, Y., Kurauchi, F. and Shimada, H. "Traffic management system against major earthquakes", IATSS Review, 24 (2), 2000.
89. Iida, Y. and Wakabayashi, H. "An approximation method of terminal reliability of a road network using partial minimal path and cut set", Proceedings of the 5th WCTR, Yokohama, Japan, pp. 367-80, 1989.
90. John, Z. Definitive guide to Swing for Java 2, 2000.
91. Johnson, D.B. "Efficient algorithms for shortest paths in sparse networks", Journal of the ACM, 24 (1), pp. 1-13, 1977.
92. Kim, T. Core Swing: advanced programming, 2000.
93. Konak A., Kulturel-Konak, S. and Smith, A.E. "Minimum cost 2-edge-connected Steiner graphs in rectilinear space: an evolutionary approach", Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, 1, pp. 97-103, 2000.
94. Kulturel-Konak, S., Konak, A. and Smith, A.E. "Minimum cost 2-edge-connected Steiner graphs in rectilinear space: an evolutionary approach", Proceedings of the 2000 Congress on Evolutionary Computation, July, La Jolla, USA, pp. 97-103, 2000.
95. Lee, Y.L., Yeh, K.Y. and Shiao, J.P. "Urban road network risk analysis after earthquake –

- Nan-Tou city case study", Conference Proceedings of the 2nd International Symposium on Transport Network Reliability, 2004, August, Christchurch & Queenstown, New Zealand, pp. 116-9, 2004.
96. Lipman, M.J. and Pippert, R.E. "Towards a measure of vulnerability II. The ratio of disruption", Graph theory with applications to algorithms and computer science (eds. Alavi, Y. Chartrand, G., Lesniak, L., Lick, D.R. and Wall C.E.), pp. 507-17. Wiley, New York, 1985.
  97. Lo, C.Y. "Developing simultaneous engineering dimension content for construction industry", Journal of National Taipei University of Technology, 31 (1), pp. 183-208, 1998.
  98. Lo, C.Y., Lee, L. and Huang, K.C. "Creating VE/VM simultaneous learning and teaching model", Development of Value Engineering and Management, pp. 130-135, Beijing, China, 2004.
  99. Lovasz, L. and Plummer, M.D. Matching theory", Annals of Discrete Mathematics, 29, North-Holland, Amsterdam, 1986.
  100. Mahjoub A. "2-edge connected spanning subgraphs and polyhedra", Mathematical Programming, 64 (2), pp. 199-208, 1994.
  101. Malik, K., Mittal, A.K. and Gupta, S.K. "The k most vital arcs in the shortest path problem", Operation Research Letters, 8 (4), pp. 223-7, 1989.
  102. Michael, T.G. & Roberto, T. Data structures and algorithms in Java, 2001.
  103. Mitch, G. Hardcore JFC: conquering the swing architecture, 2001.
  104. Mori, M., and Tsukaguchi, H. "A new method for evaluation of level of service in pedestrian facilities", Transportation Research-A, 21A (3), pp. 223-234, 1987.
  105. Nardelli, E., Proietti, G. and Widmayer, P. "Finding the detour-critical edge of a shortest path between two nodes", Information Processing Letters, 67, pp. 51-4, 1998.
  106. Nardelli, E., Proietti, G. and Widmayer, P. "A faster computation of the most vital edge of a shortest path", Information Processing Letters, 79, pp. 81-5, 2001.
  107. Nicholson, A. and Dalziell, E. "Risk evaluation and management: A road network reliability study", The Network Reliability of Transport (eds. Bell, M.G.H. and Iida, Y.), pp. 45-59, Elsevier Science, UK, 2003.
  108. Peyrat, C. "Diameter invulnerability of graphs", Discrete Applied Mathematics, 9, pp. 245-50, 1984.
  109. Ringeisen, R.D. and Lipman, M.J. "Cohesion and stability in graphs", Discrete Mathematics, 46, pp. 191-8, 1983.
  110. Pippert, R.E. and Lipman, M.J. "Towards a measure of vulnerability I. The edge-connectivity vector", Graph theory with applications to algorithms and computer science (eds. Alavi, Y. Chartrand, G., Lesniak, L., Lick, D.R. and Wall C.E.), pp. 651-7, Wiley, New York, 1985.
  111. Sakakibara, H., Kajitani, Y. and Okada, N. "Road network robustness for avoiding functional isolation in disasters", Journal of Transportation Engineering, 130 (5), pp. 560-7, 2004.
  112. Samit, S. and Hasan, P. "Design of survivable networks with connectivity requirements", Telecommunication Systems, 20 (1-2), pp. 133-149, 2002.
  113. Satyanarayana, A. "A unified formula for analysis of some network reliability problems", IEEE Transactions on Reliability, R-31, pp. 23-32, 1982.
  114. Spragins, J.D., Sinclair, J.C., Kang, Y.J. and Jafari, H. "Current telecommunication network reliability models: a critical assessment", IEEE Journal on Selected Areas in Communications, SAC-4, pp. 1168-73, 1986.
  115. Tarjan, R.E. "A note on finding the bridges of a graph", Information Processing Letters, 2, pp.

- 160-1, 1974.
116. Taylor, M.A.P and D'Este, G.M. "Critical infrastructure and transport network vulnerability: developing a method for diagnosis and assessment", The 2nd International Symposium on Transport Network Reliability, pp. 96-102, Christchurch & Queenstown, New Zealand, August 2004.
  117. Tsukaguchi, H. and Jung, H.Y. "Occupancy-a new concept in residential street planning", TEC, 43 (6), pp. 233-7, 2002.
  118. Tsukaguchi, H., Vandebona, U. and Li, Y. "Planning of residential street network for disaster prone urban areas", WCTR, 1999.
  119. Warshall, S. "A theorem on Boolean matrices", Journal of the ACM, 9 (1), pp. 11-2, January 1962.
  120. West, D.B. Introduction to Graph Theory, Prentice Hall, NJ, USA, 2001.
  121. Whitney, H. "Congruent graphs and the connectivity of graphs", Amer. J. Math., 54, pp. 150-68, 1932.
  122. Yee, A., Leung, S.K. and Wesemann, L. "The 1994 Northridge Earthquake: A transportation impact overview", Transportation Research Board, Research Circular, 462, pp. 7-19, 1996.
  123. Zemel, E. "Polynomial algorithms for estimating network reliability", Networks, 12, pp. 439-52, 1982.
  124. Zhang, L. and Levinson, D. "Investing for robustness and reliability in transportation networks", The 2nd International Symposium on Transport Network Reliability, pp. 160-6, Christchurch & Queenstown, New Zealand, August 2004.



## 附錄一 模擬數據結果

本研究針對雙供給平均需求點散布之路網條件，去除了因路網對稱性而相對相同的配置情形，進行了136種完全不同的雙供給配置組合（S1、S2之組合）情境，並根據研究所發展之指標：平均旅行成本（Average Travel Cost, ATC）、最大旅行成本（Maximum Travel Cost, MTC）、繞路成本（Longest Detour, LD）、平均互援成本（Average Mutual Assistance Cost, AMAC）、路網成本（Network Cost, NC），以觀察各種情境下之路網特性，並用以描繪主研究「防災存活路網設計模型」中圖7.3～圖7.7之趨勢，所有指標並已與初始路網總長度相除，轉換為百分率數值概念（%）。

S1	S2	To Center	Between	ATC	MTC	LD	AMAC	NC
40	8	9.52	9.52	4.23	7.14	15.48	46.43	71.43
40	15	8.33	8.33	4.10	7.14	13.10	44.05	71.43
40	16	7.14	7.14	4.23	7.14	13.10	40.48	71.43
40	29	8.33	5.95	4.36	7.14	26.19	40.48	70.24
40	36	9.52	4.76	4.76	8.33	23.81	40.48	70.24
40	30	7.14	4.76	4.50	7.14	26.19	38.10	72.62
40	37	8.33	3.57	4.76	8.33	30.95	36.31	69.05
40	32	7.14	2.38	5.29	9.52	28.57	32.14	79.76
40	39	8.33	1.19	5.55	10.71	33.33	30.36	82.14
33	9	7.14	7.14	4.36	8.33	14.29	39.88	75.00
33	11	7.14	4.76	4.63	8.33	25.00	36.90	71.43
33	15	7.14	7.14	4.10	5.95	16.67	38.69	73.81
33	16	5.95	5.95	4.23	7.14	10.71	37.50	75.00
33	18	5.95	3.57	4.90	8.33	25.00	33.33	76.19
33	19	7.14	2.38	5.42	10.71	26.19	33.33	80.95
33	29	7.14	4.76	4.36	7.14	20.24	36.31	71.43
33	30	5.95	3.57	4.36	7.14	26.19	33.33	70.24
33	32	5.95	1.19	5.16	9.52	28.57	28.57	78.57
33	37	7.14	4.76	4.63	8.33	21.43	36.31	76.19
33	39	7.14	2.38	5.42	10.71	33.33	30.36	79.76
32	16	4.76	4.76	4.50	7.14	17.86	34.52	76.19
32	18	4.76	2.38	4.76	7.14	17.86	28.57	76.19
7	1	11.90	2.38	6.22	13.10	28.57	38.69	88.10
7	2	10.71	3.57	5.42	11.90	35.71	34.52	75.00
7	4	10.71	4.76	4.63	9.52	20.24	38.10	75.00
7	5	11.90	7.14	4.63	9.52	16.67	44.05	79.76
7	8	10.71	1.19	6.22	11.90	38.10	30.95	79.76
7	9	9.52	2.38	5.42	10.71	26.19	34.52	77.38
7	10	8.33	3.57	4.36	9.52	15.48	35.71	76.19
7	11	9.52	4.76	4.36	8.33	13.10	37.50	72.62
7	12	10.71	5.95	4.36	8.33	13.10	39.29	73.81
7	13	11.90	7.14	4.36	9.52	15.48	41.07	73.81
7	14	10.71	1.19	5.95	11.90	38.10	32.14	82.14

S1	S2	To Center	Between	ATC	MTC	LD	AMAC	NC
7	15	9.52	2.38	5.69	10.71	44.05	36.31	75.00
7	16	8.33	3.57	5.03	9.52	28.57	36.31	80.95
7	17	7.14	4.76	4.10	8.33	13.10	35.12	77.38
7	18	8.33	5.95	4.10	7.14	13.10	36.90	75.00
7	19	9.52	7.14	4.10	7.14	13.10	42.26	73.81
7	20	10.71	8.33	3.97	8.33	15.48	44.64	72.62
7	22	8.33	3.57	5.55	9.52	30.95	32.74	72.62
7	23	7.14	4.76	4.63	8.33	26.19	35.12	73.81
7	25	7.14	7.14	3.84	5.95	13.10	41.07	73.81
7	26	8.33	8.33	3.70	5.95	13.10	42.26	73.81
7	28	10.71	3.57	5.16	9.52	17.86	35.71	73.81
7	29	9.52	4.76	4.90	8.33	16.67	35.12	76.19
7	30	8.33	5.95	4.50	8.33	13.10	39.88	73.81
7	31	7.14	7.14	4.10	8.33	16.67	39.88	75.00
7	32	8.33	8.33	4.10	7.14	17.86	45.24	80.95
7	33	9.52	9.52	3.84	5.95	16.67	44.64	72.62
7	34	10.71	10.71	3.70	5.95	15.48	47.02	72.62
7	35	11.90	4.76	5.16	9.52	16.67	38.69	79.76
7	36	10.71	5.95	5.03	8.33	16.67	38.69	73.81
7	37	9.52	7.14	4.36	8.33	16.67	42.26	72.62
7	38	8.33	8.33	4.10	9.52	22.62	45.24	75.00
7	39	9.52	9.52	4.10	8.33	16.67	44.64	71.43
7	40	10.71	10.71	3.97	7.14	20.24	48.21	71.43
7	41	11.90	11.90	4.10	8.33	19.05	55.36	69.05
7	43	11.90	7.14	4.63	9.52	17.86	42.26	76.19
7	44	10.71	8.33	4.23	8.33	13.10	44.64	70.24
7	46	10.71	10.71	3.97	8.33	16.67	47.02	72.62
7	47	11.90	11.90	4.10	5.95	15.48	50.60	72.62
10	1	8.33	3.57	4.90	9.52	15.48	33.33	78.57
10	2	7.14	2.38	5.29	9.52	15.48	32.14	80.95
10	8	7.14	2.38	5.29	11.90	20.24	33.93	83.33
10	9	5.95	1.19	5.29	9.52	22.62	30.36	80.95
10	14	7.14	4.76	4.23	9.52	15.48	34.52	77.38
10	15	5.95	3.57	4.90	9.52	15.48	30.95	75.00
10	16	4.76	2.38	5.55	10.71	20.24	28.57	76.19
10	22	4.76	4.76	4.50	9.52	15.48	33.33	77.38
10	23	3.57	3.57	4.23	8.33	26.19	29.17	77.38
10	28	7.14	7.14	4.23	9.52	15.48	39.29	79.76
10	29	5.95	5.95	4.23	8.33	15.48	36.31	79.76
10	30	4.76	4.76	4.50	7.14	15.48	32.14	85.71
10	31	3.57	3.57	3.97	5.95	15.48	30.36	73.81
10	35	8.33	8.33	3.97	8.33	15.48	43.45	73.81

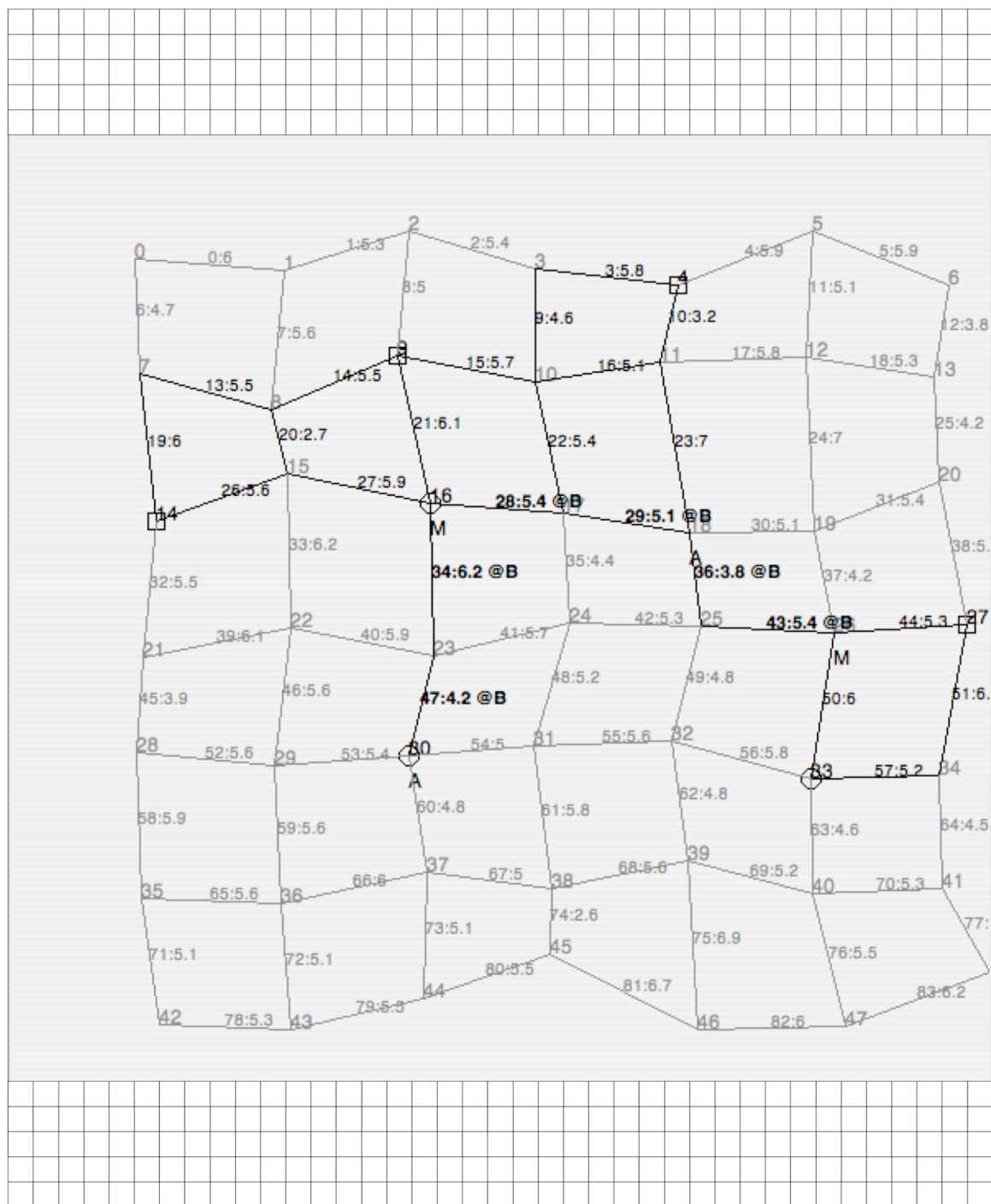
S1	S2	To Center	Between	ATC	MTC	LD	AMAC	NC
10	36	7.14	7.14	3.97	7.14	15.48	40.48	73.81
10	37	5.95	5.95	3.97	5.95	15.48	37.50	73.81
10	38	4.76	4.76	3.97	5.95	15.48	34.52	73.81
10	43	8.33	8.33	3.70	5.95	15.48	43.45	75.00
10	44	7.14	7.14	3.70	5.95	15.48	39.29	76.19
14	1	10.71	3.57	5.42	11.90	38.10	34.52	82.14
14	2	9.52	4.76	5.29	11.90	48.81	38.10	75.00
14	4	9.52	7.14	4.50	9.52	20.24	44.05	75.00
14	5	10.71	8.33	4.23	8.33	13.10	44.64	71.43
14	8	9.52	2.38	5.82	11.90	25.00	36.31	86.90
14	9	8.33	3.57	5.16	10.71	17.86	38.69	80.95
14	11	8.33	5.95	4.23	8.33	13.10	36.31	76.19
14	12	9.52	7.14	4.23	8.33	13.10	38.10	76.19
14	13	10.71	8.33	3.97	8.33	13.10	44.64	73.81
14	15	8.33	1.19	5.82	10.71	33.33	29.17	79.76
14	16	7.14	2.38	5.29	11.90	20.24	33.93	80.95
14	18	7.14	4.76	3.97	7.14	10.71	35.71	70.24
14	19	8.33	5.95	3.97	7.14	15.48	38.69	72.62
14	20	9.52	7.14	3.97	8.33	10.71	47.02	75.00
14	22	7.14	2.38	5.55	9.52	35.71	29.76	82.14
14	23	5.95	3.57	4.63	8.33	26.19	33.93	76.19
14	25	5.95	5.95	3.70	5.95	13.10	36.31	73.81
14	26	7.14	7.14	3.70	5.95	10.71	39.29	73.81
14	28	9.52	2.38	5.55	9.52	20.24	32.14	82.14
14	29	8.33	3.57	5.16	9.52	15.48	33.33	80.95
14	30	7.14	4.76	5.03	9.52	22.62	34.52	77.38
14	31	5.95	5.95	3.97	8.33	26.19	33.93	80.95
14	32	7.14	7.14	3.97	7.14	17.86	39.29	77.38
14	33	8.33	8.33	3.84	5.95	16.67	41.67	73.81
14	34	9.52	9.52	3.70	5.95	10.71	46.43	75.00
14	35	10.71	3.57	5.42	10.71	20.24	35.71	78.57
14	36	9.52	4.76	5.03	9.52	20.24	34.52	76.19
14	37	8.33	5.95	4.76	9.52	20.24	41.07	70.24
14	38	7.14	7.14	4.23	9.52	20.24	39.88	79.76
14	39	8.33	8.33	4.63	9.52	26.19	40.48	70.24
14	40	9.52	9.52	3.97	7.14	13.10	46.43	71.43
14	41	10.71	10.71	3.70	5.95	13.10	49.40	71.43
14	43	10.71	5.95	4.63	9.52	20.24	38.10	75.00
14	44	9.52	7.14	4.50	9.52	20.24	39.29	75.00
14	46	9.52	9.52	4.23	9.52	20.24	47.62	75.00
14	47	10.71	10.71	3.97	8.33	15.48	48.21	71.43
17	1	7.14	4.76	4.63	8.33	13.10	37.50	76.19

S1	S2	To Center	Between	ATC	MTC	LD	AMAC	NC
17	2	5.95	3.57	4.50	8.33	13.10	31.55	79.76
17	8	5.95	3.57	4.50	8.33	13.10	31.55	77.38
17	9	4.76	2.38	4.90	8.33	14.29	27.98	79.76
17	10	3.57	1.19	4.76	8.33	15.48	26.79	78.57
17	14	5.95	3.57	4.76	11.90	20.24	33.33	84.52
17	15	4.76	2.38	4.90	8.33	19.05	30.36	79.76
17	16	3.57	1.19	5.03	8.33	17.86	25.60	79.76
17	22	3.57	3.57	4.50	8.33	13.10	29.17	76.19
17	23	2.38	1.19	4.63	8.33	19.05	25.60	79.76
17	28	5.95	5.95	4.10	8.33	13.10	36.90	76.19
17	29	4.76	4.76	4.36	8.33	19.05	33.93	76.19
17	30	3.57	3.57	4.36	7.14	19.05	29.76	80.95
17	31	2.38	2.38	4.46	8.33	19.05	22.62	67.86
17	35	7.14	7.14	4.10	8.33	19.05	39.88	78.57
17	36	5.95	5.95	4.63	8.33	23.81	35.12	71.43
17	37	4.76	4.76	4.10	5.95	19.05	31.55	75.00
17	38	3.57	3.57	3.97	5.95	19.05	30.36	72.62
17	43	7.14	7.14	3.84	5.95	19.05	37.50	75.00
17	44	5.95	5.95	3.70	5.95	13.10	35.12	73.81

## 附錄二 路網範例

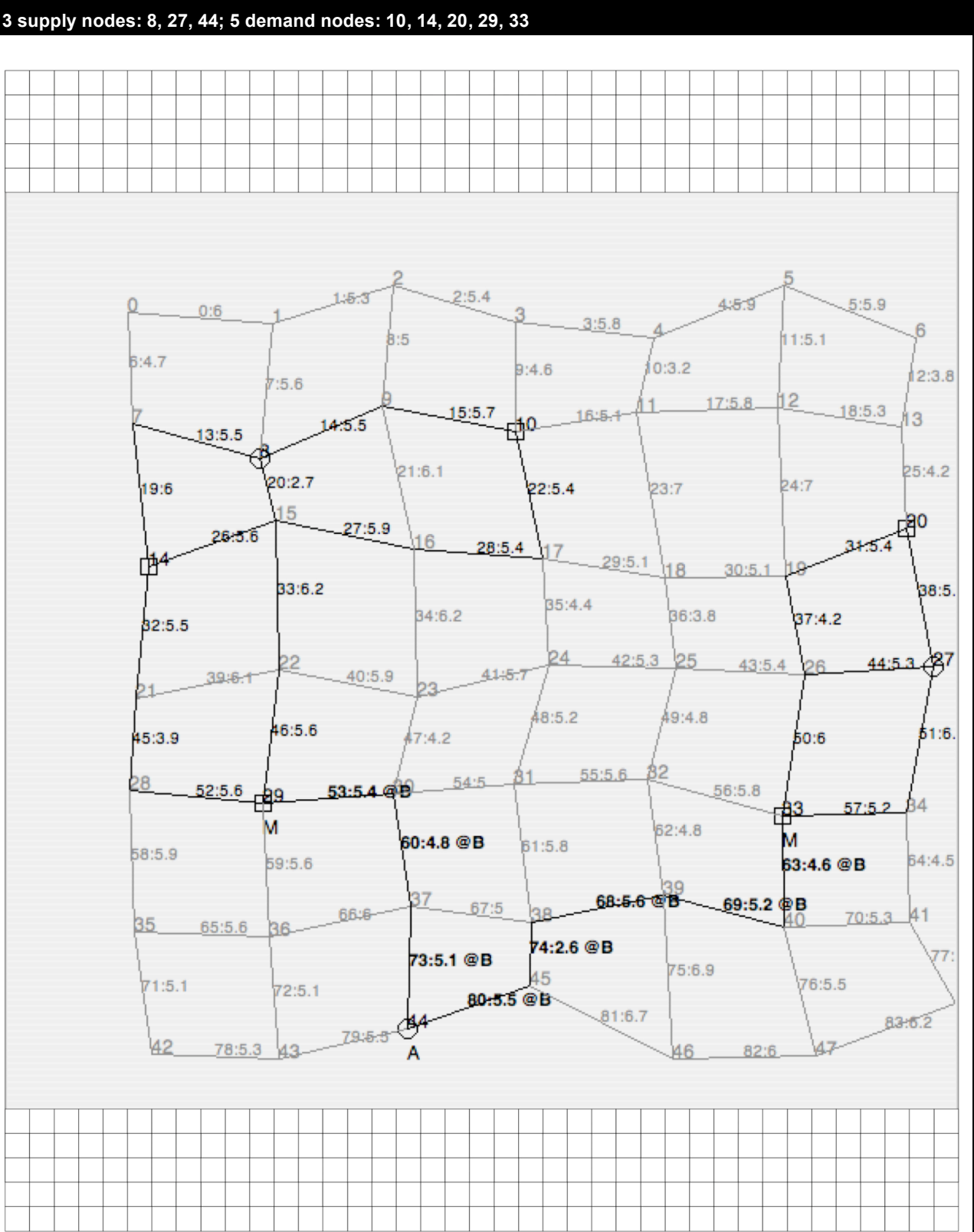
以下為利用Excel之亂數函式功能，輸入一個約以5（無因次）為間隔的棋盤式路網資料，並以三個供給點（點標籤16、點標籤30、點標籤33）、四個需求點（點標籤4、點標籤9、點標籤14、點標籤27）進行存活路網模型演算所獲得之圖形，用以顯示主研究「防災存活路網設計模型」中圖6.19路網範例之演算結果。

3 supply nodes: 16, 30, 33; 4 demand nodes: 4, 9, 14, 27

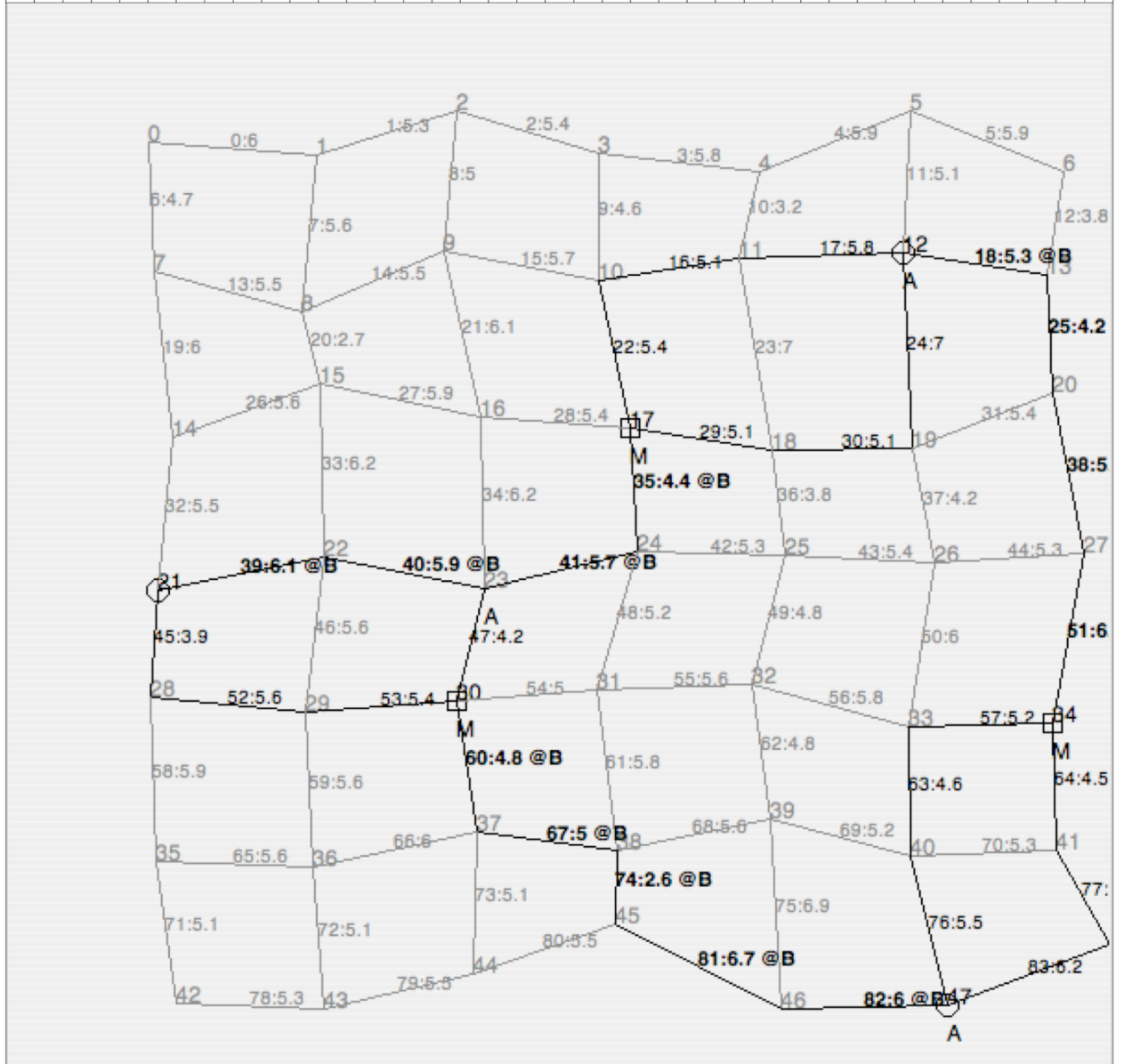


資料來源：本研究繪製。

該存活路網模型適用條件為：雙供給點以上、單需求點以上的任何二邊連接路網圖形。以下為各種舉例情形。

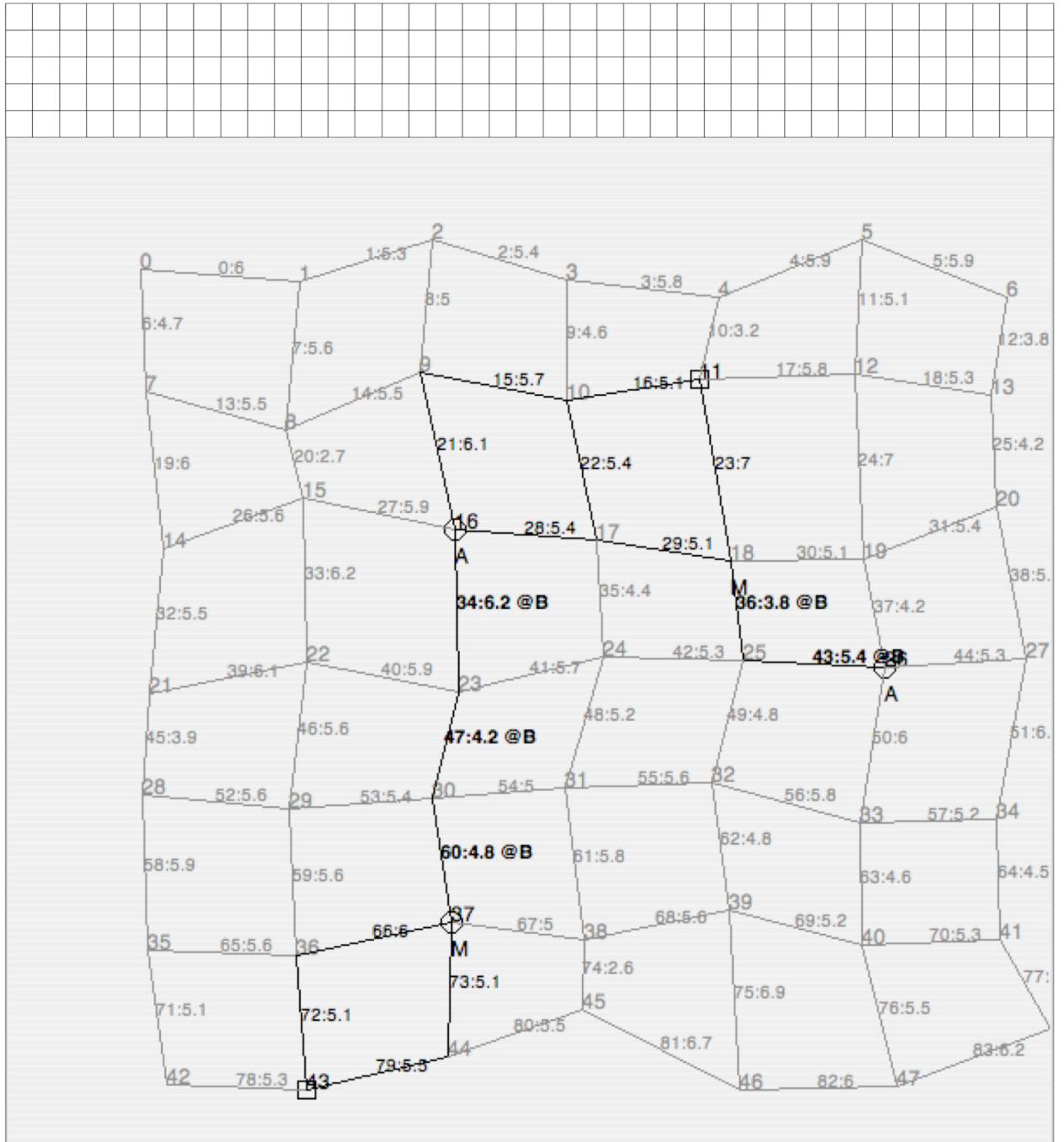


3 supply node: 12, 21, 47; 3 demand nodes: 17, 30, 34



資料來源：本研究繪製。

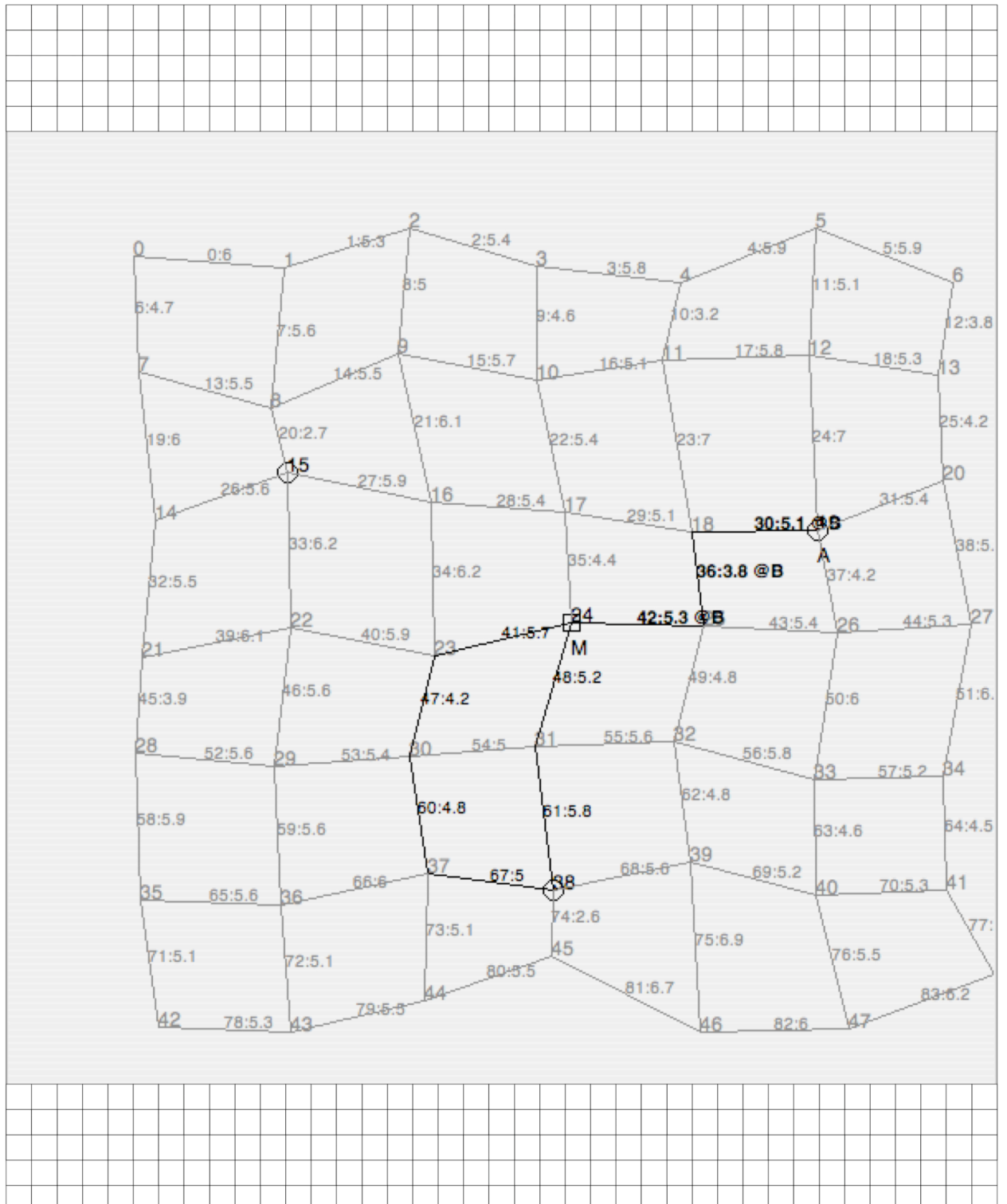
3 supply nodes: 16, 26, 37; 2 demand nodes: 11, 43



資料來源：本研究繪製。



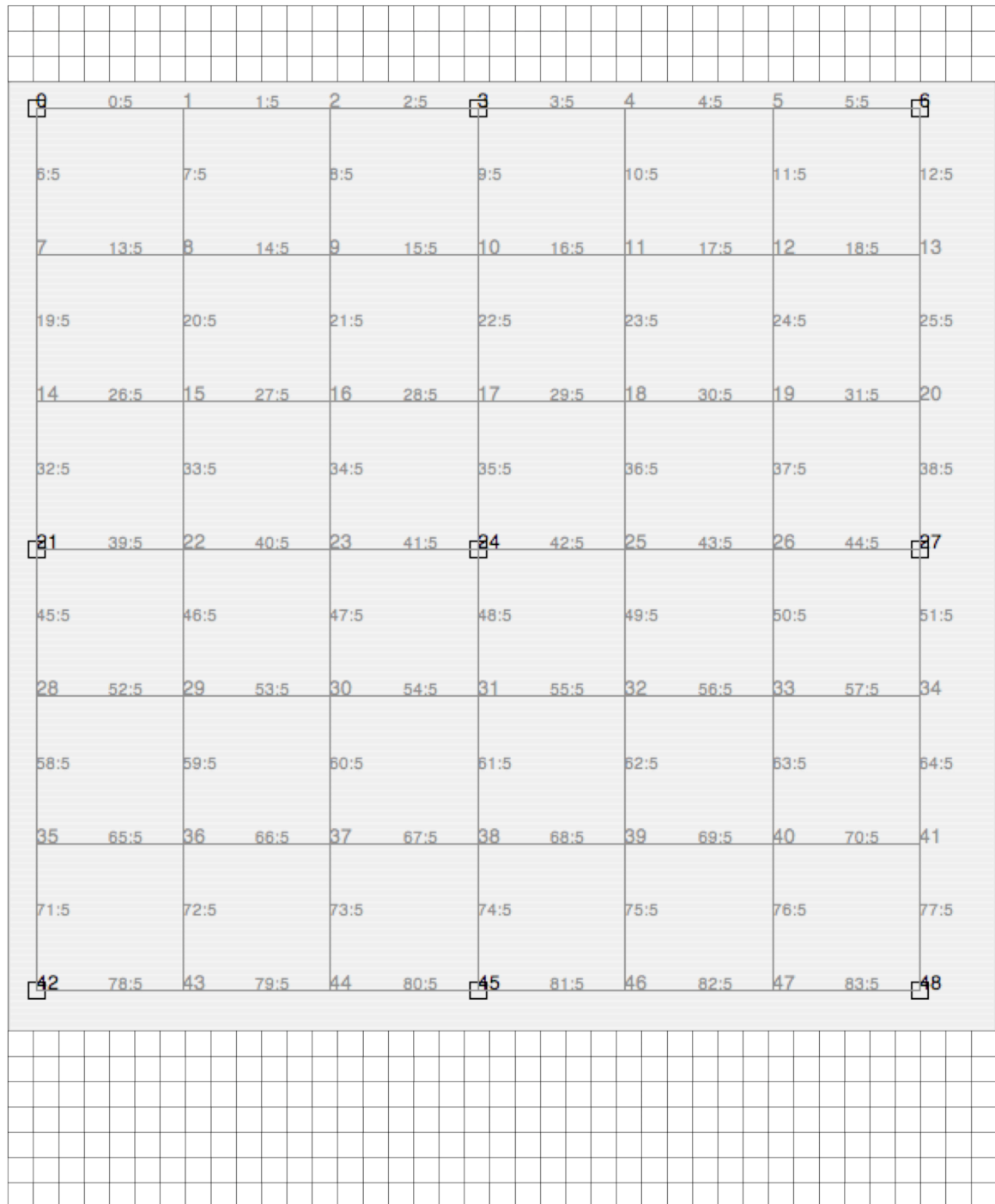
3 supply nodes: 15, 19, 38; 1 demand node: 24



資料來源：本研究繪製。

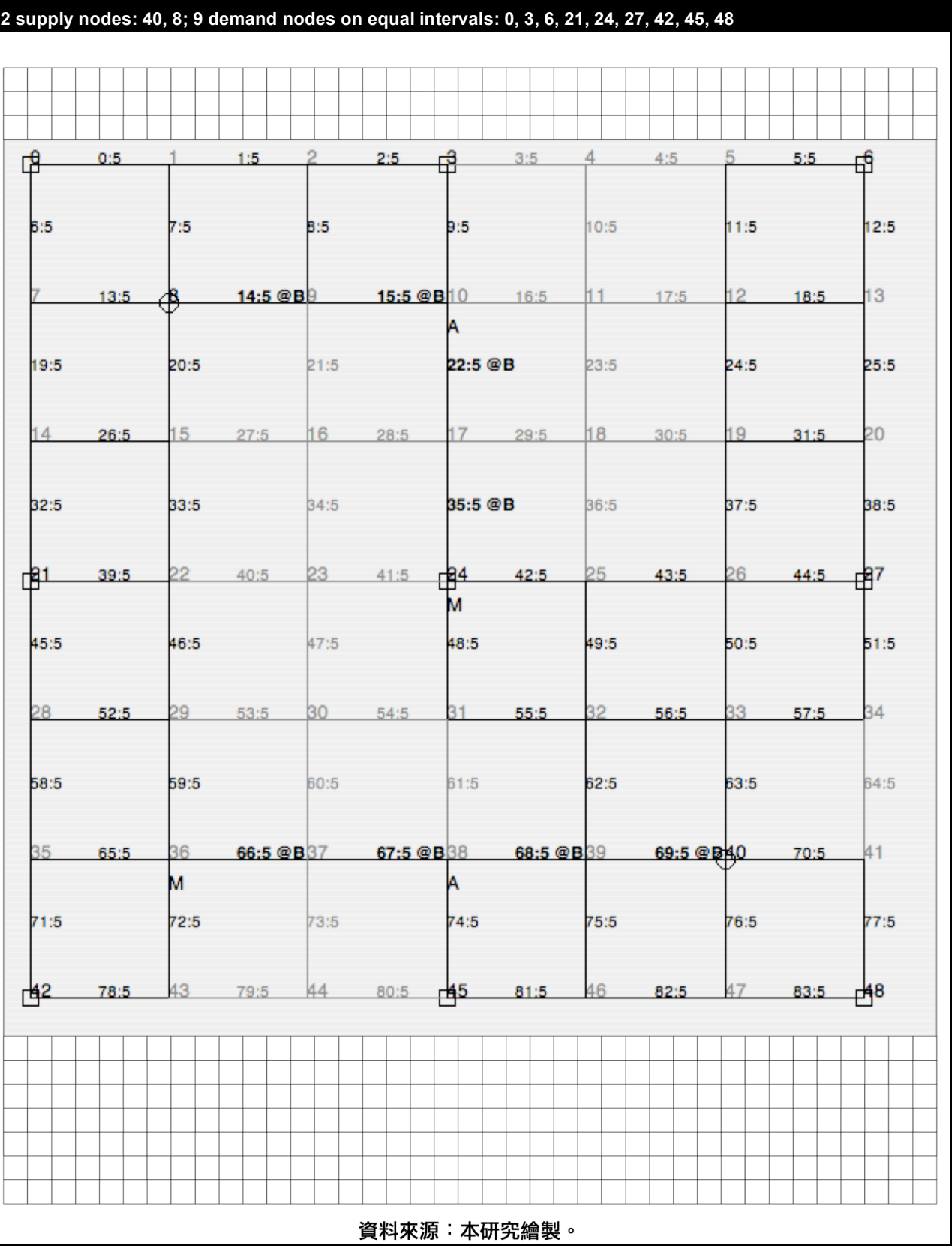
以下為來自主研究「防災存活路網設計模型」中表7.1及表7.2之點、邊資料所構築之7\*7正方棋盤測試路網（主研究之圖7.2），該假設情境為需求點（包括：點標籤0、點標籤3、點標籤6、點標籤21、點標籤24、點標籤27、點標籤42、點標籤45、點標籤48）均勻散布在此路網之中，作為雙供給點設計之基礎條件。

2 supply nodes of 136 scenarios; 9 demand nodes on equal intervals: 0, 3, 6, 21, 24, 27, 42, 45, 48

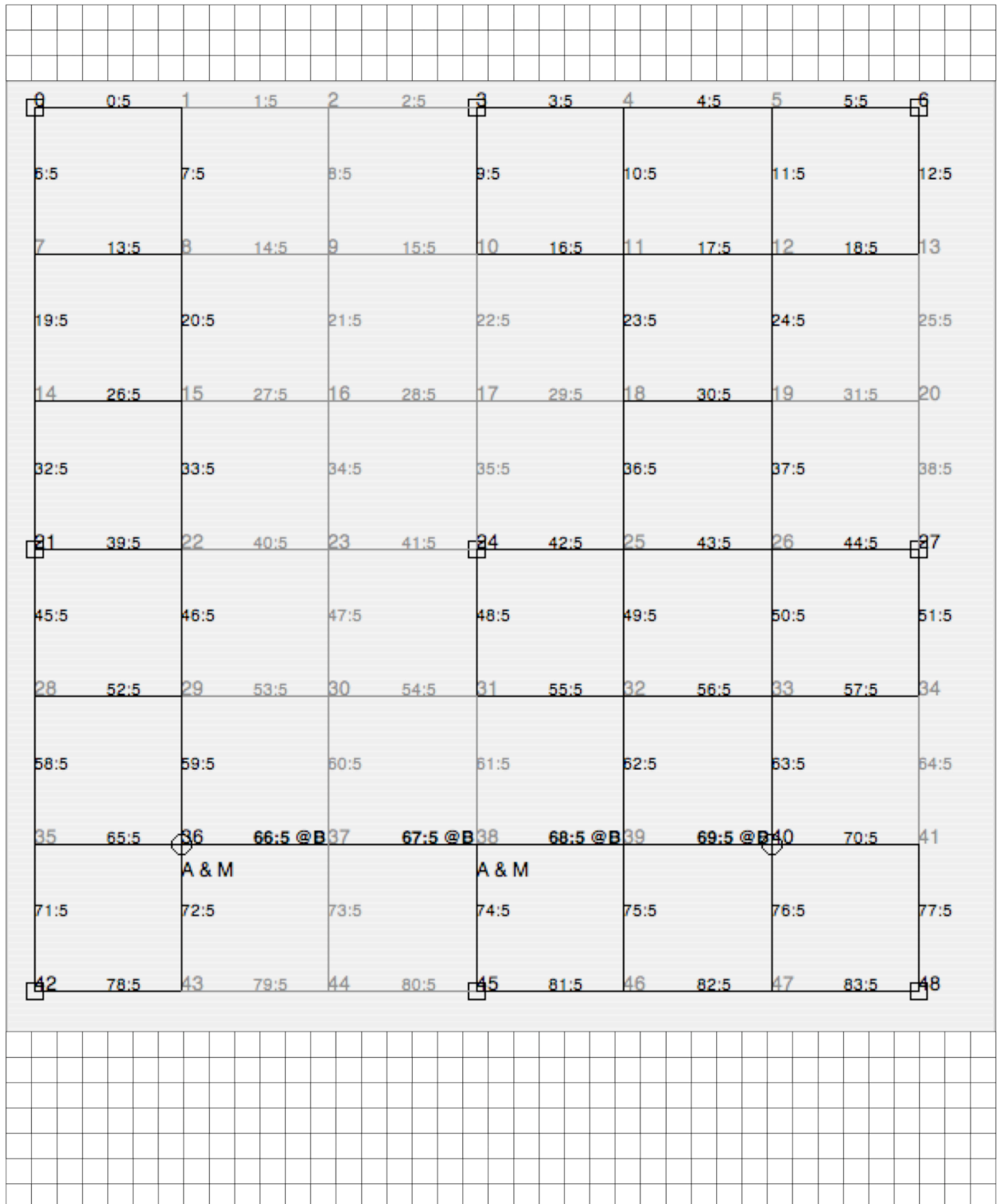


資料來源：本研究繪製。

主研究「防災存活路網設計模型」中共模擬136種完全不同相對位置的雙供給點佈設情境，以下列舉其中幾種佈設情形。

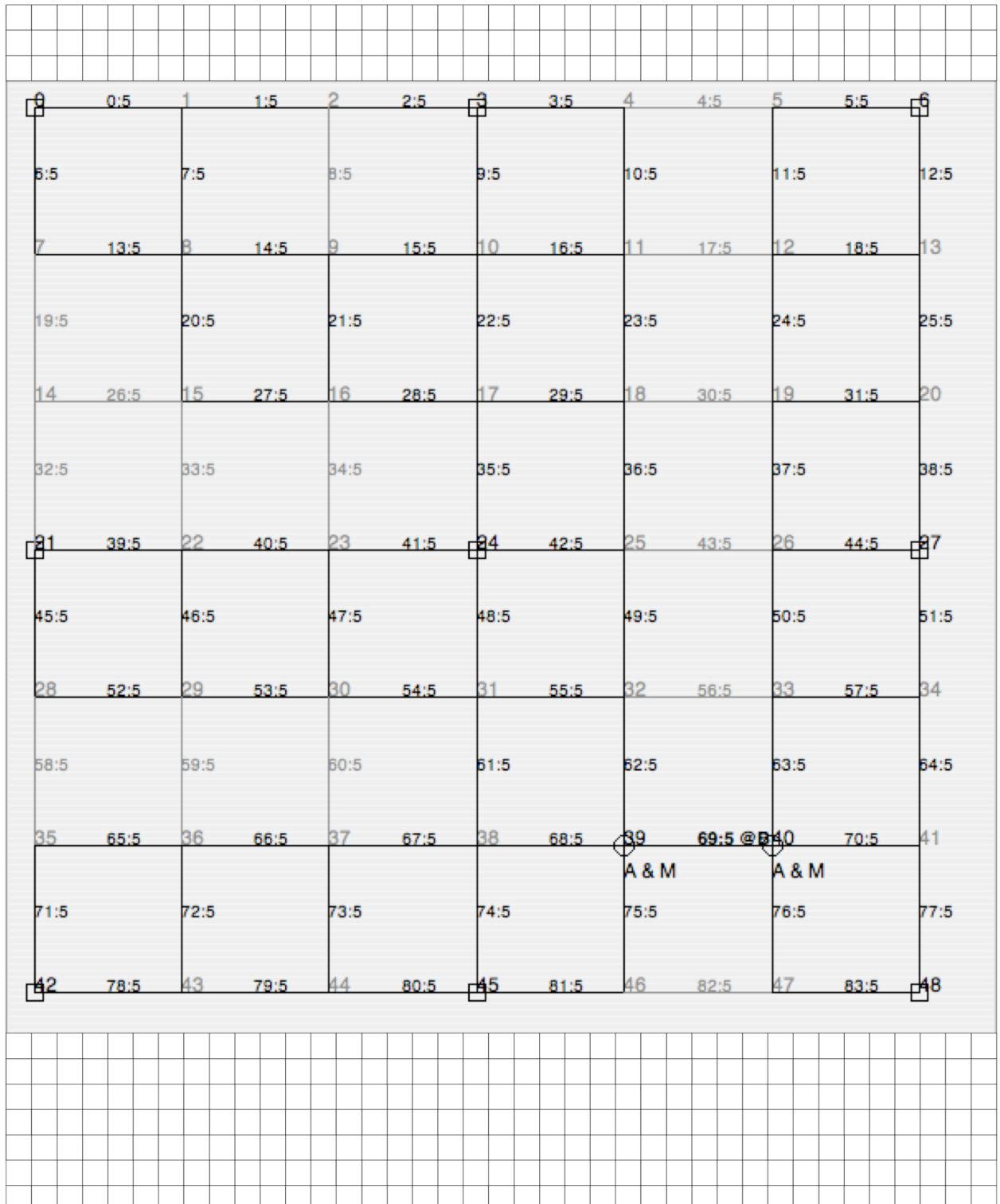


2 supply nodes: 40, 36; 9 demand nodes on equal intervals: 0, 3, 6, 21, 24, 27, 42, 45, 48



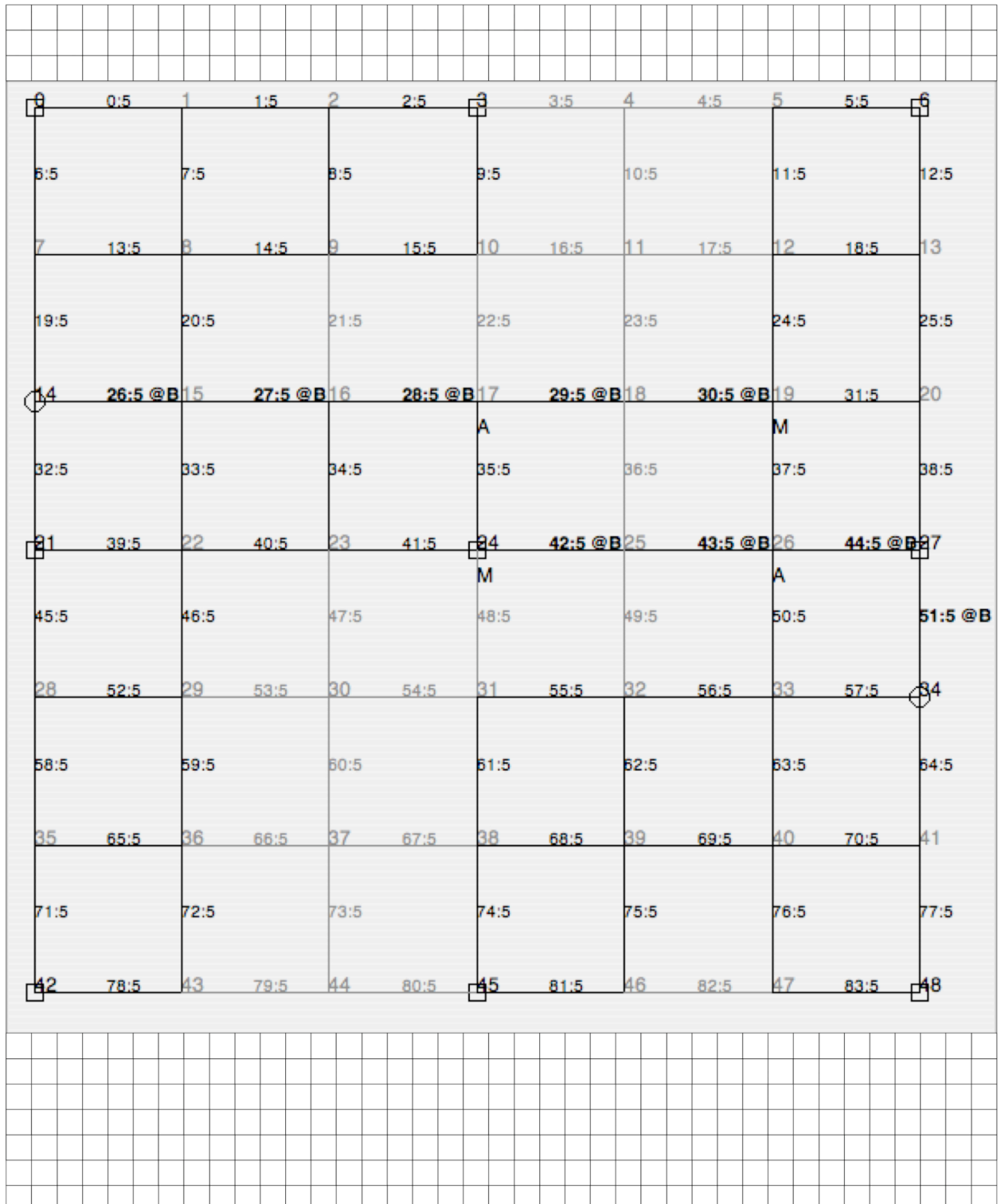
資料來源：本研究繪製。

2 supply nodes: 40, 39; 9 demand nodes on equal intervals: 0, 3, 6, 21, 24, 27, 42, 45, 48



資料來源：本研究繪製。

2 supply nodes: 14, 34; 9 demand nodes on equal intervals: 0, 3, 6, 21, 24, 27, 42, 45, 48



資料來源：本研究繪製。

### 附錄三 程式碼

#### GraphAlgorithm.java

```
package emnet.algorithm;

import java.util.Vector;
import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Iterator;
import emnet.thread.Center;
import emnet.thread.Roamer;
import emnet.thread.Detourist;
import emnet.thread.DetourManager;

public class GraphAlgorithm {
    public GraphAlgorithm() {
        try {
            jblnit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public static Graph copyGraph(Graph oldGraph){
        Vector oldNodeSet,oldEdgeSet;

        oldNodeSet=oldGraph.getNodeSet();
        oldEdgeSet=oldGraph.getEdgeSet();

        return copyGraph(oldNodeSet,oldEdgeSet);
    }

    public static Graph copyGraph(Vector oldNodeSet,Vector oldEdgeSet){
        Vector newNodeSet=new Vector(),newEdgeSet=new Vector();

        for(int i=0;i<oldNodeSet.size();i++){

            // new node setting:
            // 1. check label
```

```
// 2. check supply/demand
// 3. check visit
Node tempOldNode=(Node)oldNodeSet.elementAt(i);
Node tempNewNode=new Node(i);

if(tempOldNode.isSupply()){
    tempNewNode.setSupply();
}else if(tempOldNode.isDemand()){
    tempNewNode.setDemand();
}

if(tempOldNode.isVisited()){
    tempNewNode.visit();
}

newNodeSet.addElement(tempNewNode);
}

for(int i=0;i<oldEdgeSet.size();i++){

    // new edge setting:
    // 1. check label
    // 2. check n1/n2
    // 3. set weight
    // 4. set previous node
    // 5. check fast
    // 6. check detour
    // 7. check visit
    Edge tempOldEdge=(Edge)oldEdgeSet.elementAt(i);
    int n1=tempOldEdge.getN1Label(),n2=tempOldEdge.getN2Label();
    double weight=tempOldEdge.getWeight();
    Edge tempNewEdge=new
Edge(i,(Node)newNodeSet.elementAt(n1),(Node)newNodeSet.elementAt(n2),weight);

    if(tempOldEdge.isFastEdge()){
        tempNewEdge.setFastEdge();
    }

    if(tempOldEdge.isDetourEdge()){
        tempNewEdge.setDetourEdge();
    }
}
```



```
        if(tempOldEdge.isVisited()){
            tempNewEdge.visit();
        }

        newEdgeSet.addElement(tempNewEdge);
    }
    return new Graph(newNodeSet,newEdgeSet);
}

public static Graph getSubtreeWithCertainNode(Graph tree,Node certainNode,Edge ruinedEdge){

    Vector nodeSet=new Vector(),edgeSet=new Vector();
    Vector currAdjacentNodes,currIncidentEdges;
    Node currNode=certainNode;

    int maxLabel=0;
    Vector tempTreeNodeSet=tree.getNodeSet();
    Node tempTreeNode;
    for(int i=0;i<tempTreeNodeSet.size();i++){
        tempTreeNode=(Node)tempTreeNodeSet.elementAt(i);
        if(tempTreeNode.getLabel()>maxLabel)
            maxLabel=tempTreeNode.getLabel();
    }

    Vector preNodes=new Vector(maxLabel+1);
    for(int i=0;i<(maxLabel+1);i++)
        preNodes.addElement(null);

    boolean finish=false;

    do{
        currAdjacentNodes=tree.adjacentNodeSet(currNode);
        currIncidentEdges=tree.incidentEdgeSet(currNode);

        if(currIncidentEdges.contains(ruinedEdge)){
            currIncidentEdges.removeElement(ruinedEdge);
            currAdjacentNodes.removeElement(ruinedEdge.theOtherNode(currNode));
        }

        Iterator itrAdjacentNodes=currAdjacentNodes.iterator();

        if(!nodeSet.contains(currNode))
```

```

        nodeSet.addElement(currNode);

        int i=0;
        currNodeAssign:
        do{
            if(itrAdjacentNodes.hasNext()){
                Node tempNode=(Node)itrAdjacentNodes.next();

                if(!nodeSet.contains(tempNode)){
                    preNodes.setElementAt(currNode,tempNode.getLabel());
                    currNode=tempNode;
                    break currNodeAssign;
                }else{
                    i++;
                    if(i==currAdjacentNodes.size()){
                        if(currNode==certainNode){
                            finish=true;
                            break currNodeAssign;
                        }
                        Edge
tempEdge=tree.getEdge(currNode,(Node)preNodes.elementAt(currNode.getLabel()));
                        if(!edgeSet.contains(tempEdge)){
                            edgeSet.addElement(tempEdge);
                        }
                        currNode=(Node)preNodes.elementAt(currNode.getLabel());
                    }
                }
            }else{
                finish=true;
                //break currNodeAssign;
            }
        }while(itrAdjacentNodes.hasNext());
    }while(!finish);
    return new Graph(nodeSet,edgeSet);
}

public static Vector getAdjacentNodes(Graph graph,Node currNode){
    return graph.adjacentNodeSet(currNode);
}

public static Vector getIncidnetEdges(Graph graph,Node currNode){
    return graph.incidentEdgeSet(currNode);
}

```

```

}
```

```

public static Graph getSubtreeWithoutSupply(Graph tree,Edge ruinedEdge){
    Graph subtreeWithoutSupply=null;
    Graph subtreeN1=getSubtreeWithCertainNode(tree,ruinedEdge.getN1(),ruinedEdge);
    Graph subtreeN2=getSubtreeWithCertainNode(tree,ruinedEdge.getN2(),ruinedEdge);
    Vector subtreeNodeSetN1=subtreeN1.getNodeSet();
    Node tempNode;
    for(int i=0;i<subtreeNodeSetN1.size();i++){
        tempNode=(Node)subtreeNodeSetN1.elementAt(i);
        if(tempNode.isSupply())
            subtreeWithoutSupply=subtreeN2;
    }
    if(subtreeWithoutSupply!=subtreeN2)
        subtreeWithoutSupply=subtreeN1;
    return subtreeWithoutSupply;
}
}
```

```

public static Graph getSubtreeWithSupply(Graph tree,Edge ruinedEdge){
    Graph subtreeWithSupply=null;
    Graph subtreeN1=getSubtreeWithCertainNode(tree,ruinedEdge.getN1(),ruinedEdge);
    Graph subtreeN2=getSubtreeWithCertainNode(tree,ruinedEdge.getN2(),ruinedEdge);
    Vector subtreeNodeSetN1=subtreeN1.getNodeSet();
    Node tempNode;
    for(int i=0;i<subtreeNodeSetN1.size();i++){
        tempNode=(Node)subtreeNodeSetN1.elementAt(i);
        if(tempNode.isSupply())
            subtreeWithSupply=subtreeN1;
    }
    if(subtreeWithSupply!=subtreeN1)
        subtreeWithSupply=subtreeN2;
    return subtreeWithSupply;
}
}
```

```

private void jblnit() throws Exception {
}
}
```

```

public static Vector getIncidentEdgeSet(Roamer roamer,Vector nodeSet){
    Center center=roamer.getCenter();
    Graph graph=center.getGraph();
}
```

```

    Edge myEdge=center.getMyEdge(roamer);
    Vector visitorSequence;

    Vector edgeSet=new Vector();
    Edge tempEdge;
    Node n1,n2;
    for(int i=0;i<nodeSet.size();i++){
        Vector tempEdgeSet=graph.incidentEdgeSet((Node)nodeSet.elementAt(i));
        for(int j=0;j<tempEdgeSet.size();j++){
            if(!edgeSet.contains(tempEdgeSet.elementAt(j))){
                edgeSet.addElement(tempEdgeSet.elementAt(j));

                tempEdge=(Edge)tempEdgeSet.elementAt(j);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                //remove edges included in the routeEdgeSet
                if((nodeSet.contains(n1) && nodeSet.contains(n2))
                    edgeSet.removeElement(tempEdge);
                //remove dummyEdges
                if(n1==center.getDummyNode() || n2==center.getDummyNode())
                    edgeSet.removeElement(tempEdge);
                //remove visited edge
                if((nodeSet.contains(n1)){
                    visitorSequence=center.getVisitorSequence(n2);
                    if(visitorSequence.contains(roamer))
                        edgeSet.removeElement(tempEdge);
                }else if((nodeSet.contains(n2)){
                    visitorSequence=center.getVisitorSequence(n1);
                    if(visitorSequence.contains(roamer))
                        edgeSet.removeElement(tempEdge);
                }
            }
        }
    }
    if(edgeSet.contains(myEdge))
        edgeSet.removeElement(myEdge);

    return edgeSet;
}

public static Vector getIncidentEdges(Detourist detourist,Vector nodeSet){
    Graph usableGraph=detourist.getUsableGraph();

```

## GraphAlgorithm.java

```
DetourManager dmr=detourist.getDetourManager();
Center center=dmr.getCenter();
Edge myEdge=dmr.getMyEdge(detourist);

Vector incidentEdges=new Vector();
Edge tempEdge;
Node n1,n2;

for(int i=0;i<nodeSet.size();i++){
    Vector tempEdgeSet=usableGraph.incidentEdgeSet((Node)nodeSet.elementAt(i));
    for(int j=0;j<tempEdgeSet.size();j++){
        tempEdge=(Edge)tempEdgeSet.elementAt(j);
        if(!incidentEdges.contains(tempEdge))
            incidentEdges.addElement(tempEdge);

        n1=tempEdge.getN1();
        n2=tempEdge.getN2();
        if(nodeSet.contains(n1) && nodeSet.contains(n2))
            incidentEdges.removeElement(tempEdge);
        if(n1==center.getDummyNode() || n2==center.getDummyNode())
            incidentEdges.removeElement(tempEdge);
    }
}
if(incidentEdges.contains(myEdge))
    incidentEdges.removeElement(myEdge);

return incidentEdges;
}

public static Vector getInterfaceNodes(Graph subject,Graph environment){
    Vector interfaceNodes=new Vector();
    Vector environmentEdgeSet=environment.getEdgeSet();

    Edge tempEdge;
    Node n1,n2;

    for(int i=0;i<environmentEdgeSet.size();i++){
        tempEdge=(Edge)environmentEdgeSet.elementAt(i);
        n1=tempEdge.getN1();
        n2=tempEdge.getN2();

        if(subject.hasNode(n1) && !subject.hasNode(n2)){
```

```
        if(!interfaceNodes.contains(n1))
            interfaceNodes.addElement(n1);
    }
    if(subject.hasNode(n2) && !subject.hasNode(n1)){
        if(!interfaceNodes.contains(n2))
            interfaceNodes.addElement(n2);
    }
}
return interfaceNodes;
}
```

```
public static double networkCost(Graph graph){
    Vector edgeSet=graph.getEdgeSet();
    Edge edge;
    double networkCost=0.0;
    for(int i=0;i<edgeSet.size();i++){
        edge=(Edge)edgeSet.elementAt(i);
        networkCost=networkCost+edge.getWeight();
    }
    return networkCost;
}
```

```
public static double pathCost(Vector edgeSet){
    double pathkCost=0.0;
    Edge edge;
    for(int i=0;i<edgeSet.size();i++){
        edge=(Edge)edgeSet.elementAt(i);
        pathkCost=pathkCost+edge.getWeight();
    }
    return pathkCost;
}
```

```
public static Vector intersection(Vector set0,Vector set1){
    Vector intersection=new Vector();
    Object temp;
    for(int i=0;i<set0.size();i++){
        temp=set0.elementAt(i);
        if(set1.contains(temp))
            intersection.addElement(temp);
    }
    return intersection;
}
```

```

    }

    public static Vector union(Vector set0, Vector set1){
        Vector union=new Vector();
        Object temp;
        for(int i=0;i<set0.size();i++){
            temp=set0.elementAt(i);
            union.addElement(temp);
        }
        for(int i=0;i<set1.size();i++){
            temp=set1.elementAt(i);
            if(!union.contains(temp))
                union.addElement(temp);
        }
        return union;
    }

    //detour
    public static double getMergeCost(Detourist detourist, Node node){

        Center center=detourist.getDetourManager().getCenter();
        double mergeCost=0.0;

        Graph downstream=detourist.getDownstream();
        Vector downstreamDemandNodeSet=downstream.getDemandNodeSet();

        Graph mergeNodeToSupplyPath=getPathToSupply(node, center);
        Vector mergeNodeToSupplyEdgeSet=mergeNodeToSupplyPath.getEdgeSet();

        Vector tempEdgeSetUnion=new Vector(), tempEdgeSetIntersection=new Vector();

        Node tempDemand;
        Graph tempDemandToSupplyPath;
        Vector tempDemandToSupplyEdgeSet;
        for(int i=0;i<downstreamDemandNodeSet.size();i++){
            tempDemand=(Node)downstreamDemandNodeSet.elementAt(i);

            tempDemandToSupplyPath=getPathToSupply(tempDemand, center);
            tempDemandToSupplyEdgeSet=tempDemandToSupplyPath.getEdgeSet();

            tempEdgeSetUnion=union(mergeNodeToSupplyEdgeSet, tempDemandToSupplyEdgeSet);

```

```
tempEdgeSetIntersection=intersection(mergeNodeToSupplyEdgeSet,tempDemandToSupplyEdgeSet);

        mergeCost=mergeCost+pathCost(tempEdgeSetUnion)-pathCost(tempEdgeSetIntersection);
    }
    return mergeCost;
}

public static Graph getPathToSupply(Node node,Center center){
    Vector nodeSet=new Vector();
    Vector edgeSet=new Vector();

    Node currNode=node;
    Edge preEdge;
    while(currNode!=center.getDummyNode()){
        preEdge=center.getPreEdge(currNode);

        if(center.getPreNode(currNode)!=center.getDummyNode()){
            if(!nodeSet.contains(currNode))
                nodeSet.addElement(currNode);
            if(!edgeSet.contains(preEdge))
                edgeSet.addElement(preEdge);
        }else{
            if(!nodeSet.contains(currNode))
                nodeSet.addElement(currNode);
        }
        currNode=preEdge.theOtherNode(currNode);
    }
    return new Graph(nodeSet,edgeSet);
}
```



## Graph.java

```
package emnet.graph;

import java.util.Vector;

public class Graph {
    private Vector nodeSet, edgeSet;
    int supplyNodeNum, demandNodeNum;

    public Graph(Vector nodeSet, Vector edgeSet){
        this.nodeSet=nodeSet;
        this.edgeSet=edgeSet;
    }

    public Vector getNodeSet(){
        return this.nodeSet;
    }

    public Vector getEdgeSet(){
        return this.edgeSet;
    }

    //incident edges of node n
    public Vector incidentEdgeSet(Node node){
        Vector incidentEdgeSet=new Vector();
        for(int i=0;i<this.edgeSet.size();i++){
            Edge tempEdge=(Edge)edgeSet.elementAt(i);
            if(tempEdge.getN1()!=node || tempEdge.getN2()!=node){
                if(!incidentEdgeSet.contains(tempEdge)){
                    incidentEdgeSet.addElement(tempEdge);
                }
            }
        }
        return incidentEdgeSet;
    }

    //adjacent nodes of node n
    public Vector adjacentNodeSet(Node n){
        Vector adjacentNodeSet=new Vector();
        for(int i=0;i<this.edgeSet.size();i++){
            Edge tempEdge=(Edge)edgeSet.elementAt(i);
            if(tempEdge.getN1()==n && !adjacentNodeSet.contains(tempEdge.getN2())){
```

```

        adjacentNodeSet.addElement(tempEdge.getN2());
    }else if(tempEdge.getN2()==n && !adjacentNodeSet.contains(tempEdge.getN1())){
        adjacentNodeSet.addElement(tempEdge.getN1());
    }
}
return adjacentNodeSet;
}

public Vector getSupplyNodeSet(){
    Vector supplyNodeSet=new Vector();
    for(int i=0;i<this.nodeSet.size();i++){
        Node tempNode=(Node)nodeSet.elementAt(i);
        if(tempNode.isSupply())
            supplyNodeSet.addElement(tempNode);
    }
    return supplyNodeSet;
}

public int getSupplyNodeNum(){
    return getSupplyNodeSet().size();
}

public Vector getDemandNodeSet(){
    Vector demandNodeSet=new Vector();
    for(int i=0;i<this.nodeSet.size();i++){
        Node tempNode=(Node)nodeSet.elementAt(i);
        if(tempNode.isDemand())
            demandNodeSet.addElement(tempNode);
    }
    return demandNodeSet;
}

public int getDemandNodeNum(){
    return getDemandNodeSet().size();
}

public Edge getEdge(Node n1,Node n2){
    Edge edge;
    for(int i=0;i<edgeSet.size();i++){
        edge=(Edge)edgeSet.elementAt(i);
        if(edge.getN1()==n1){
            if(edge.theOtherNode(edge.getN1())==n2)

```

```

        return edge;
    }else if(edge.getN1()==n2){
        if(edge.theOtherNode(edge.getN1())==n1)
            return edge;
    }
}
System.out.println("error: no edge can be returned!");
return null;
}

public void addNode(Node node){
    if(!nodeSet.contains(node))
        this.nodeSet.addElement(node);
}

public void addEdge(Edge edge){
    if(!edgeSet.contains(edge))
        this.edgeSet.addElement(edge);
}

public boolean hasNode(Node node){
    if(nodeSet.contains(node)){
        return true;
    }else{
        return false;
    }
}

public boolean hasEdge(Edge edge){
    if(edgeSet.contains(edge)){
        return true;
    }else{
        return false;
    }
}

public void removeNode(Node node){
    if(nodeSet.contains(node)){
        Vector incidentEdges=this.incidentEdgeSet(node);
        Edge tempIncidentEdge;
        for(int i=0;i<incidentEdges.size();i++){
            tempIncidentEdge=(Edge)incidentEdges.elementAt(i);

```

```
        if(edgeSet.contains(tempIncidentEdge))
            edgeSet.removeElement(tempIncidentEdge);
    }
    nodeSet.removeElement(node);
}

public void removeEdge(Edge edge){
    if(edgeSet.contains(edge))
        edgeSet.removeElement(edge);
}
}
```

## Node.java

```
package emnet.graph;

public class Node{
    private int label;
    private double x,y;
    private boolean demand,supply,merge,access,source,dummy;
    private boolean visit,occupy;

    public Node(int label){
        this.label=label;
        this.demand=false;
        this.supply=false;
        this.merge=false;
        this.access=false;
        this.source=false;
        this.dummy=false;
        this.visit=false;
        this.occupy=false;
    }

    public Node(int label,double x,double y){
        this(label);
        setX(x);
        setY(y);
    }

    public void setDemand(){
        this.demand=true;
    }

    public void setSupply(){
        this.supply=true;
    }

    public void setNeutral(){
        this.supply=false;
        this.demand=false;
    }

    public void setMerge(){
        this.merge=true;
    }
```

```
}

public void setAccess(){
    this.access=true;
}

public void setSource(){
    this.source=true;
}

public void setDummy(){
    this.dummy=true;
}

public synchronized void visit(){
    this.visit=true;
}

public synchronized void occupy(){
    this.occupy=true;
}

public synchronized void unOccupied(){
    this.occupy=false;
}

public synchronized void leave(){
    this.occupy=false;
}

public boolean isOccupied(){
    return this.occupy;
}

public boolean isDemand(){
    return this.demand;
}

public boolean isSupply(){
    return this.supply;
}
```

## Node.java

```
public boolean isMerge(){
    return this.merge;
}

public boolean isAccess(){
    return this.access;
}

public boolean isSource(){
    return this.source;
}

public boolean isDummy(){
    return this.dummy;
}

public boolean isVisited(){
    return this.visit;
}

public int getLabel(){
    return label;
}

public void setX(double x){
    this.x=x;
}

public void setY(double y){
    this.y=y;
}

public double getX(){
    return this.x;
}

public double getY(){
    return this.y;
}
}
```

## Edge.java

```
package emnet.graph;

public class Edge {
    private int label;
    private double weight;
    private Node n1,n2;
    private boolean fastEdge,detourEdge,testEdge,detourTestEdge,dummyEdge,maTestEdge,maEdge;
    private boolean visit;

    public Edge(int label,Node n1,Node n2){
        this.label=label;
        this.weight=0.0;
        this.n1=n1;
        this.n2=n2;
        this.fastEdge=false;
        this.detourEdge=false;
        this.testEdge=false;
        this.detourTestEdge=false;
        this.dummyEdge=false;
        this.maTestEdge=false;
        this.maEdge=false;
        this.visit=false;
    }

    public Edge(int label,Node n1,Node n2,double weight){
        this(label,n1,n2);
        this.weight=weight;
    }

    public synchronized void setFastEdge(){
        this.fastEdge=true;
    }

    public boolean isFastEdge(){
        return this.fastEdge;
    }

    public synchronized void setDetourEdge(){
        this.detourEdge=true;
    }
}
```



## Edge.java

```
public boolean isDetourEdge(){
    return this.detourEdge;
}

public synchronized void setTestEdge(){
    this.testEdge=true;
}

public boolean isTestEdge(){
    return this.testEdge;
}

public synchronized void setDetourTestEdge(boolean detourTestEdge){
    this.detourTestEdge=detourTestEdge;
}

public boolean isDetourTestEdge(){
    return detourTestEdge;
}

public synchronized void setDummyEdge(){
    this.dummyEdge=true;
}

public boolean isDummyEdge(){
    return dummyEdge;
}

public synchronized void setMATestEdge(boolean maTestEdge){
    this.maTestEdge=maTestEdge;
}

public boolean isMATestEdge(){
    return maTestEdge;
}

public synchronized void setMAEdge(){
    this.maEdge=true;
}

public boolean isMAEdge(){
    return maEdge;
}
```

```
}

public synchronized void setNeutralEdge(){
    this.fastEdge=false;
    this.detourEdge=false;
    this.maEdge=false;
    this.testEdge=false;
    this.detourTestEdge=false;
    this.maTestEdge=false;
}

public void setWeight(double weight){
    this.weight=weight;
}

public double getWeight(){
    return this.weight;
}

public Node getN1(){
    return this.n1;
}

public Node getN2(){
    return this.n2;
}

public Node theOtherNode(Node n){
    if(n==this.n1){
        return this.n2;
    }else if(n==this.n2){
        return this.n1;
    }
    return null;
}

public synchronized void visit(){
    this.visit=true;
}

public boolean isVisited(){
    return this.visit;
```

## Edge.java

```
    }

    public int getN1Label(){
        return n1.getLabel();
    }

    public int getN2Label(){
        return n2.getLabel();
    }

    public int getLabel(){
        return this.label;
    }
}
```

## Map.java

```
package emnet.gui;

import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.util.Vector;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import emnet.Frame;
import java.text.DecimalFormat;
import java.awt.Font;

public class Map extends javax.swing.JPanel{
    Frame frame;
    JPanel map=new JPanel();
    Graph graph;
    boolean dataIn;
    boolean supply,demand,neutral;
    DecimalFormat myFormatter=new DecimalFormat("###,###.##");

    public Map(){
        init();
    }

    public void init(){
        this.setLayout(new BorderLayout());
        this.setSize(new Dimension(600,400));
        this.setPreferredSize(new Dimension(600,400));
        this.add(map,BorderLayout.CENTER);
        this.dataIn=false;

        this.addMouseListener(new Map_MouseAdapter(this));
    }

    public void paint(Graphics g){
        if(dataIn){
```

```
        drawNodes(graph,g);
        drawEdges(graph,g);
        frame.setSeperator(graph);
    }else{
        g.drawString("n/a",this.getWidth()/2,this.getHeight()/2);
    }
}

public void setGraph(Graph graph){
    this.graph=graph;
    this.dataIn=true;
    this.repaint();
}

public Graph getGraph(){
    return this.graph;
}

public void sentFrame(Frame frame){
    this.frame=frame;
}

public Map getMap(){
    return this;
}

private void drawNodes(Graph graph,Graphics g){
    double ratio=scaledRatio(graph);
    double newOX=newOX(graph);
    double newOY=newOY(graph);
    int l=12,m=10,s=8;

    g.setFont(new Font(null,Font.PLAIN,12));

    Vector nodeSet=graph.getNodeSet();
    Node node;
    for(int i=0;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);

        if(node.isSupply()){
            g.setColor(Color.RED);
            g.drawOval(new Double((node.getX()-newOX)*ratio).intValue()-l/2,new
Double((node.getY()-newOY)*ratio).intValue()-l/2,l,l);
```

```

        }else if(node.isDemand()){
            g.setColor(Color.BLUE);
            g.drawRect(new Double((node.getX()-newOX)*ratio).intValue()-m/2,new
Double((node.getY()-newOY)*ratio).intValue()-m/2,m,m);
        }else{
            g.setColor(Color.LIGHT_GRAY);
        }
        g.drawString(""+node.getLabel(),new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue());

        if(node.isAccess() && node.isMerge()){
            g.setColor(Color.DARK_GRAY);
            g.drawString("A & M",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
        }else if(node.isAccess()){
            g.setColor(Color.DARK_GRAY);
            g.drawString("A",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
        }else if(node.isMerge()){
            g.setColor(Color.DARK_GRAY);
            g.drawString("M",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
        }

        if(node.isSource()){
            g.setColor(Color.DARK_GRAY);
            g.drawString("$rc",new Double((node.getX()-newOX)*ratio).intValue(),new
Double((node.getY()-newOY)*ratio).intValue()+20);
        }
    }
}

private void drawEdges(Graph graph,Graphics g){
    double ratio=scaledRatio(graph);
    double newOX=newOX(graph);
    double newOY=newOY(graph);

    Vector edgeSet=graph.getEdgeSet();
    Edge edge;

    g.setFont(new Font(null,Font.PLAIN,10));

```

```

    for(int i=0;i<edgeSet.size();i++){
        edge=(Edge)edgeSet.elementAt(i);
        Node n1=edge.getN1(),n2=edge.getN2();
        int n1x=new Double((n1.getX()-newOX)*ratio).intValue();
        int n1y=new Double((n1.getY()-newOY)*ratio).intValue();
        int n2x=new Double((n2.getX()-newOX)*ratio).intValue();
        int n2y=new Double((n2.getY()-newOY)*ratio).intValue();

        if(edge.isFastEdge()){
            g.setColor(Color.BLACK);
        }else if(edge.isDetourEdge()){
            g.setColor(Color.ORANGE);
        }else if(edge.isMAEdge()){
            g.setColor(Color.GREEN);
        }else if(edge.isTestEdge()){
            g.setColor(Color.MAGENTA);
        }else if(edge.isDetourTestEdge()){
            g.setColor(Color.CYAN);
        }else if(edge.isMATestEdge()){
            g.setColor(Color.MAGENTA);
        }else{
            g.setColor(Color.LIGHT_GRAY);
        }

        if(!edge.isDummyEdge()){
            Double weight;
            if(edge.isMAEdge()){
                g.setFont(new Font(null,Font.BOLD,11));
                g.drawLine(n1x,n1y,n2x,n2y);
                weight=new Double(edge.getWeight());
                g.drawString(""+edge.getLabel()+":"+myFormatter.format(weight)+" @B",new
Double((n1x+n2x)/2).intValue(),new Double((n1y+n2y)/2).intValue());
                g.setFont(new Font(null,Font.PLAIN,10));
            }else{
                g.drawLine(n1x,n1y,n2x,n2y);
                weight=new Double(edge.getWeight());
                g.drawString(""+edge.getLabel()+":"+myFormatter.format(weight),new
Double((n1x+n2x)/2).intValue(),new Double((n1y+n2y)/2).intValue());
            }
        }
    }
}

```

```

private double scaledRatio(Graph graph){
    Vector nodeSet=graph.getNodeSet();
    double maxX=0.0,maxY=0.0;
    double newOX=newOX(graph),newOY=newOY(graph);
    Node node;
    for(int i=0;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        if((node.getX()-newOX)>maxX){
            maxX=node.getX()-newOX;
        }else if((node.getY()-newOY)>maxY){
            maxY=node.getY()-newOY;
        }
    }
    return Math.min(map.getWidth()/maxX,map.getHeight()/maxY);
}

```

```

private double newOX(Graph graph){
    Vector nodeSet=graph.getNodeSet();
    Node node=(Node)nodeSet.elementAt(0);
    double minX=node.getX();
    for(int i=1;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        if(node.getX()<minX){
            minX=node.getX();
        }
    }
    return minX;
}

```

```

private double newOY(Graph graph){
    Vector nodeSet=graph.getNodeSet();
    Node node=(Node)nodeSet.elementAt(0);
    double minY=node.getY();
    for(int i=1;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        if(node.getY()<minY){
            minY=node.getY();
        }
    }
    return minY;
}

```



```

public void nodeSetting(boolean supply,boolean demand,boolean neutral){
    this.supply=supply;
    this.demand=demand;
    this.neutral=neutral;
}

void mouse_clicked_actionPerformed(MouseEvent e){
    double ratio=scaledRatio(graph);
    double newOX=newOX(graph),newOY=newOY(graph);
    Vector nodeSet=graph.getNodeSet();
    Node node;
    int err=10;
    for(int i=0;i<nodeSet.size();i++){
        node=(Node)nodeSet.elementAt(i);
        int x=new Double((node.getX()-newOX)*ratio).intValue();
        int y=new Double((node.getY()-newOY)*ratio).intValue();
        if(Math.abs(e.getX()-x)<err && Math.abs(e.getY()-y)<err){
            if(supply){
                node.setNeutral();
                node.setSupply();
            }
            if(demand){
                node.setNeutral();
                node.setDemand();
            }
            if(neutral){
                node.setNeutral();
            }
        }
    }
}

class Map_MouseAdapter extends MouseAdapter{
    Map adaptee;

    Map_MouseAdapter(Map adaptee){
        this.adaptee=adaptee;
    }

    public void mouseClicked(MouseEvent e){

```

**Map.java**

```
        this.adaptee.mouse_clicked_actionPerformed(e);  
    }  
}
```

## **IOGraph.java**

```
package emnet.io;

import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Vector;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;
import java.util.StringTokenizer;

public class IOGraph {

    private Graph g;

    public IOGraph(String dirName,String nodeFile,String edgeFile) throws IOException {
        File inputNodeFile=new File(dirName,nodeFile);
        File inputEdgeFile=new File(dirName,edgeFile);
        Vector nodeSet=new Vector();
        Vector edgeSet=new Vector();
        g=new Graph(nodeSet,edgeSet);

        //read node.txt
        FileReader nodeIn=new FileReader(inputNodeFile);
        BufferedReader buffNodeIn=new BufferedReader(nodeIn);

        String strLine;
        Node node;
        while((strLine=buffNodeIn.readLine())!=null){
            //delimiter of Tab is "\t"
            StringTokenizer strToken=new StringTokenizer(strLine,"\t",false);

            String[] nodeAttr=new String[3];
            for(int i=0;i<3;i++){
                nodeAttr[i]=strToken.nextToken();
            }
            node=new
Node(Integer.parseInt(nodeAttr[0]),Double.parseDouble(nodeAttr[1]),Double.parseDouble(nodeAttr[2]));
            nodeSet.addElement(node);
        }
    }
}
```

## IOGraph.java

```
buffNodeIn.close();
nodeIn.close();

//read edge.txt
FileReader edgeIn=new FileReader(inputEdgeFile);
BufferedReader buffEdgeIn=new BufferedReader(edgeIn);

Edge edge;
while((strLine=buffEdgeIn.readLine())!=null){
    //delimiter of Tab is "\t"
    StringTokenizer strToken=new StringTokenizer(strLine,"\t",false);

    String[] edgeAttr=new String[3];
    for(int i=0;i<3;i++){
        edgeAttr[i]=strToken.nextToken();
    }

    Node n1=(Node)nodeSet.elementAt(Integer.parseInt(edgeAttr[1]));
    Node n2=(Node)nodeSet.elementAt(Integer.parseInt(edgeAttr[2]));
    double
weight=Math.sqrt((n1.getX()-n2.getX())*(n1.getX()-n2.getX())+(n1.getY()-n2.getY())*(n1.getY()-n2.getY()));
    edge=new Edge(Integer.parseInt(edgeAttr[0]),n1,n2,weight);
    edgeSet.addElement(edge);
}
buffEdgeIn.close();
edgeIn.close();
}

public Graph getGraph(){
    return this.g;
}
}
```

## Center.java

```
package emnet.thread;

import java.util.Vector;
import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import emnet.Frame;
import emnet.algorithm.GraphAlgorithm;

public class Center extends Thread{
    //Center
    Graph graph;
    Frame frame;
    boolean finish,available,detourCenterFinish,maCenterFinish;
    Node dummyNode;

    //nodeCenter
    int nodeNum;
    Object[][] nodeCenter;

    //roamerCenter
    int roamerNum;
    Object[][] roamerCenter;

    //fastCenter
    Object[][] fastCenter;

    //detourCenter
    int edgeNum,detourManagerNum;
    Object[][] detourCenter;

    //MutualAssistanceCenter
    Graph emnet;
    int maManagerNum;
    Object[][] maCenter;

    public Center(Graph graph,Frame frame){
        //Center
        this.graph=graph;
        this.frame=frame;
        finish=false;
    }
}
```

```
available=true;

//nodeCenter
//nodeCenter[node][0]: occupy(Boolean)
//nodeCenter[node][1]: distance(Double)
//nodeCenter[node][2]: preNode(Node)
//nodeCenter[node][3]: preEdge(Edge)
//nodeCenter[node][4]: visitorSequence(Vector)
//nodeCenter[node][5]: supply(Node)
//nodeCenter[node][6]: detourDist(Double)???

nodeNum=this.graph.getNodeSet().size();
nodeCenter=new Object[(nodeNum+1)][6];
dummyNode=new Node(nodeNum,0.0,0.0);

for(int i=0;i<(nodeNum+1);i++){
    nodeCenter[i][0]=new Boolean(false);
    nodeCenter[i][1]=new Double(0.0);
    nodeCenter[i][2]=null;
    nodeCenter[i][3]=null;
    nodeCenter[i][4]=new Vector();
    nodeCenter[i][5]=null;
}

//roamerCenter
//roamerCenter[roamer][0]: myNode(Node)
//roamerCenter[roamer][1]: myEdge(Edge)
//roamerCenter[roamer][2]: currNode(Node)
//roamerCenter[roamer][3]: routeSet(Graph)
roamerNum=graph.getSupplyNodeSet().size();
roamerCenter=new Object[roamerNum][4];
for(int i=0;i<roamerNum;i++){
    roamerCenter[i][0]=dummyNode;
    roamerCenter[i][1]=null;
    roamerCenter[i][2]=null;
    roamerCenter[i][3]=null;
}

//fastCenter
//fastCenter[demand][0]: fastPath(Graph)
//fastCenter[demand][1]: supply(Node)
//fastCenter[demand][2]: fastPathLength(Double)
```

## Center.java

```
fastCenter=new Object[nodeNum][3];
for(int i=0;i<nodeNum;i++){
    fastCenter[i][0]=null;
    fastCenter[i][1]=null;
    fastCenter[i][2]=null;
}

//detourCenter
//detourCenter[edge][0]: downstream(Graph)
//detourCenter[edge][1]: upstream(Graph)
//detourCenter[edge][2]: mergeNode(Node)
//detourCenter[edge][3]: accessNode(Node)
//detourCenter[edge][4]: detourPath(Graph)
//detourCenter[edge][5]: systematicDetourCost(Double)
//detourCenter[edge][6]: mergeCost(Vector)
edgeNum=graph.getEdgeSet().size();
detourManagerNum=graph.getSupplyNodeSet().size();
detourCenter=new Object[edgeNum][7];
for(int i=0;i<edgeNum;i++){
    detourCenter[i][0]=null;
    detourCenter[i][1]=null;
    detourCenter[i][2]=null;
    detourCenter[i][3]=null;
    detourCenter[i][4]=null;
    detourCenter[i][5]=new Double(0.0);
    detourCenter[i][6]=new Vector(nodeNum);
}

//mutualAssistantCenter
//maCenter[supply][0]: fastTree(Graph)
//maCenter[supply][1]: territory(Graph) 2ECON
//maCenter[supply][2]: source(Node)
//maCenter[supply][3]: icpSet(Vector)
//maCenter[supply][4]: maPath(Graph)
//maCenter[supply][5]: within territory supply-demand ratio(Double)
//maCenter[supply][6]: mutual assistant supply-demand ratio(Double)
//maCenter[supply][7]: maCost > source to supply
maManagerNum=graph.getSupplyNodeNum();
maCenter=new Object[nodeNum][8];
for(int i=0;i<nodeNum;i++){
    maCenter[i][0]=null;
    maCenter[i][1]=null;
```

```

        maCenter[i][2]=null;
        maCenter[i][3]=new Vector();
        maCenter[i][4]=null;
        maCenter[i][5]=new Vector();
        maCenter[i][6]=new Double(0.0);
        maCenter[i][7]=new Double(0.0);
    }
}

public void run(){
    startRoamerCenter();
    startFastCenter();
    startDetourCenter();
    startMutualAssistanceCenter();
    startOutputReport();
}

//center method
void startRoamerCenter(){
    //roamerCenter
    //send roamers to find fast paths (Shortest Path Forest, SPF)
    sendRoamer();

    Vector nodeSet=graph.getNodeSet();
    int demandNodeNum=graph.getDemandNodeNum();
    do{
        int totalTimes=0;
        watching:
        for(int i=0;i<nodeNum;i++){
            Node tempNode=(Node)nodeSet.elementAt(i);
            //finish condition is focused on demand nodes only
            if(tempNode.isDemand()){
                if(getVisitorNum(tempNode)==roamerNum){
                    totalTimes=totalTimes+roamerNum;
                    if(totalTimes==(demandNodeNum*roamerNum))
                        finish=true;
                }
            }
            else{
                finish=false;
                break watching;
            }
        }
    }
}

```



```

    }
    }while(!finish);
    //roamer center finished!
}

void startFastCenter(){
    //fastCenter
    //fastCenter[demand][0]: fastPath(Graph)
    //fastCenter[demand][1]: supply(Node)
    //fastCenter[demand][2]: fastPathLength(Double)

    //finding fast paths from demand nodes to the dummyNode
    try{
        sleep(1);
    }catch(InterruptedException ex){
        //fast center cannot sleep!
    }

    Vector demandNodeSet=graph.getDemandNodeSet();
    Node tempDemand,tempNode;
    Edge tempPreEdge,tempEdge;
    double length;
    Vector tempFastNodeSet,tempFastEdgeSet;

    for(int i=0;i<demandNodeSet.size();i++){
        tempDemand=(Node)demandNodeSet.elementAt(i);
        tempNode=tempDemand;

        tempFastNodeSet=new Vector();
        tempFastEdgeSet=new Vector();

        find:
        while(tempNode!=dummyNode){
            tempPreEdge=(Edge)getPreEdge(tempNode);
            if(tempNode.isSupply()){
                fastCenter[tempDemand.getLabel()][1]=tempNode;
                if(!tempFastNodeSet.contains(tempNode))
                    tempFastNodeSet.addElement(tempNode);

                //node of fast route belong to the same supply
                Node tempNode2;
                for(int j=0;j<tempFastNodeSet.size();j++){

```

```

        tempNode2=(Node)tempFastNodeSet.elementAt(j);
        nodeCenter[tempNode2.getLabel()][5]=tempNode;
    }
    break find;
}else{
    if(!tempFastNodeSet.contains(tempNode))
        tempFastNodeSet.addElement(tempNode);
    if(!tempFastEdgeSet.contains(tempPreEdge))
        tempFastEdgeSet.addElement(tempPreEdge);
    tempPreEdge.setFastEdge();
}
tempNode=tempPreEdge.theOtherNode(tempNode);
}

fastCenter[tempDemand.getLabel()][0]=new Graph(tempFastNodeSet,tempFastEdgeSet);
length=0.0;
for(int j=0;j<tempFastEdgeSet.size();j++){
    tempEdge=(Edge)tempFastEdgeSet.elementAt(j);
    length=length+tempEdge.getWeight();
}
fastCenter[tempDemand.getLabel()][2]=new Double(length);
}

//maCenter[supply][0]: fastTree(Graph)
Vector fastTreeSupplyNodeSet=graph.getSupplyNodeSet();
Node tempFastTreeSupply;
Vector fastTreeDemandNodeSet=graph.getDemandNodeSet();
Node tempFastTreeDemand;

Graph tempFastPath;
Vector tempFastPathNodeSet;
Node tempFastPathNode;
Vector tempFastPathEdgeSet;
Edge tempFastPathEdge;

Vector tempFastTreeNodeSet;
Vector tempFastTreeEdgeSet;

for(int i=0;i<fastTreeSupplyNodeSet.size();i++){
    tempFastTreeNodeSet=new Vector();
    tempFastTreeEdgeSet=new Vector();

```

```

tempFastTreeSupply=(Node)fastTreeSupplyNodeSet.elementAt(i);
if(!tempFastTreeNodeSet.contains(tempFastTreeSupply))
    tempFastTreeNodeSet.addElement(tempFastTreeSupply);

for(int j=0;j<fastTreeDemandNodeSet.size();j++){
    tempFastTreeDemand=(Node)fastTreeDemandNodeSet.elementAt(j);

    //fast paths with different demands of the same supply
    if(fastCenter[tempFastTreeDemand.getLabel()][1]==tempFastTreeSupply){
        tempFastPath=(Graph)fastCenter[tempFastTreeDemand.getLabel()][0];
        tempFastPathEdgeSet=tempFastPath.getEdgeSet();
        for(int k=0;k<tempFastPathEdgeSet.size();k++){
            tempFastPathEdge=(Edge)tempFastPathEdgeSet.elementAt(k);
            if(!tempFastTreeEdgeSet.contains(tempFastPathEdge)){
                tempFastTreeEdgeSet.addElement(tempFastPathEdge);
            }
        }

        tempFastPathNodeSet=tempFastPath.getNodeSet();
        for(int k=0;k<tempFastPathNodeSet.size();k++){
            tempFastPathNode=(Node)tempFastPathNodeSet.elementAt(k);
            if(!tempFastTreeNodeSet.contains(tempFastPathNode)){
                tempFastTreeNodeSet.addElement(tempFastPathNode);
            }
        }
    }
}

maCenter[tempFastTreeSupply.getLabel()][0]=new
Graph(tempFastTreeNodeSet,tempFastTreeEdgeSet);

//maCenter[supply][5]: within territory supply-demand ratio(Double)
if(getFastTree(tempFastTreeSupply).getDemandNodeNum()!=0){

this.setTerritorySDR(tempFastTreeSupply,1.0/getFastTree(tempFastTreeSupply).getDemandNodeNum());
    }else{
        this.setTerritorySDR(tempFastTreeSupply,1.0);
    }

}

//fast center finished!
//finalization

```

## Center.java

```
    Vector edgeSet=graph.getEdgeSet();
    Edge clearEdge;
    for(int i=0;i<edgeSet.size();i++){
        clearEdge=(Edge)edgeSet.elementAt(i);
        if(!clearEdge.isFastEdge())
            clearEdge.setNeutralEdge();
    }
}

void startDetourCenter(){
    //detourCenter
    try{
        sleep(1);
    }catch(InterruptedException ex){
        //detour center cannot sleep!
    }

    detourCenterFinish=false;
    Vector supplyNodeSet=graph.getSupplyNodeSet();

    Node tempSupply;
    for(int i=0;i<supplyNodeSet.size();i++){
        tempSupply=(Node)supplyNodeSet.elementAt(i);
        new DetourManager(i,this,tempSupply).start();
    }

    while(!detourCenterFinish){
        //detour center waiting for detour managers finish their jobs
    }
    //detour center finished!
}

void startMutualAssistanceCenter(){
    try{
        sleep(1);
    }catch(InterruptedException ex){
        //mutual assistance center cannot sleep!
    }

    //where are supply nodes:
    Vector nodeSet=graph.getNodeSet();
    Node tempNode;
```

```
for(int i=0;i<nodeSet.size();i++){
    tempNode=(Node)nodeSet.elementAt(i);
}

//initialization
Vector edgeSet=graph.getEdgeSet();
Edge tempEdge;
for(int i=0;i<edgeSet.size();i++){
    tempEdge=(Edge)edgeSet.elementAt(i);
    if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge())
        tempEdge.setNeutralEdge();
}

maCenterFinish=false;
Vector supplyNodeSet=graph.getSupplyNodeSet();

Node tempSupply;
for(int i=0;i<supplyNodeSet.size();i++){
    tempSupply=(Node)supplyNodeSet.elementAt(i);
    new MAManager(i,this,tempSupply).start();
}

while(!maCenterFinish){
    //ma center waiting for detour managers finish their jobs
}

//emnet
Vector emnetNodeSet=new Vector();
Vector emnetEdgeSet=new Vector();

Graph tempTerritory;
Vector tempTerritoryNodeSet;
Vector tempTerritoryEdgeSet;

Graph tempMAPath;
Vector tempMAPathNodeSet;
Vector tempMAPathEdgeSet;

for(int i=0;i<supplyNodeSet.size();i++){
    tempSupply=(Node)supplyNodeSet.elementAt(i);

    tempTerritory=this.getTerritory(tempSupply);
```

```

        tempTerritoryNodeSet=tempTerritory.getNodeSet();
        tempTerritoryEdgeSet=tempTerritory.getEdgeSet();

        emnetNodeSet=GraphAlgorithm.union(emnetNodeSet,tempTerritoryNodeSet);
        emnetEdgeSet=GraphAlgorithm.union(emnetEdgeSet,tempTerritoryEdgeSet);

        tempMAPath=this.getMAPath(tempSupply);
        tempMAPathNodeSet=tempMAPath.getNodeSet();
        tempMAPathEdgeSet=tempMAPath.getEdgeSet();

        emnetNodeSet=GraphAlgorithm.union(emnetNodeSet,tempMAPathNodeSet);
        emnetEdgeSet=GraphAlgorithm.union(emnetEdgeSet,tempMAPathEdgeSet);
    }
    emnet=new Graph(emnetNodeSet,emnetEdgeSet);

    //ma center finished!
    //finalization
    for(int i=0;i<edgeSet.size();i++){
        tempEdge=(Edge)edgeSet.elementAt(i);
        if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
            tempEdge.setNeutralEdge();
    }
    //ma center closed!
}

void startOutputReport(){
    //maCenter[supply][6]: mutual assistant supply-demand ratio(Double)
    double ld=0.0;
    Vector supplyNodeSet=graph.getSupplyNodeSet();
    Node tempSupply;
    Vector tempFastTreeEdgeSet;
    Edge tempRuinedEdge;
    for(int i=0;i<supplyNodeSet.size();i++){
        tempSupply=(Node)supplyNodeSet.elementAt(i);
        tempFastTreeEdgeSet=this.getFastTree(tempSupply).getEdgeSet();
        for(int j=0;j<tempFastTreeEdgeSet.size();j++){
            tempRuinedEdge=(Edge)tempFastTreeEdgeSet.elementAt(j);
            if(this.getSystematicDetourCost(tempRuinedEdge)>ld)
                ld=this.getSystematicDetourCost(tempRuinedEdge);
        }
    }
    frame.setLD(ld);
}

```

```

double mac=0.0;
for(int i=0;i<supplyNodeSet.size();i++){
    tempSupply=(Node)supplyNodeSet.elementAt(i);
    mac=mac+this.getMACost(tempSupply);
}
frame.setAMAC(mac/supplyNodeSet.size());
frame.setNC(GraphAlgorithm.networkCost(emnet));

double fastCost=0.0;
double maxFastCost=0.0;
Vector demandNodeSet=graph.getDemandNodeSet();
Node tempDemand;
for(int i=0;i<demandNodeSet.size();i++){
    tempDemand=(Node)demandNodeSet.elementAt(i);

    if(getFastCost(tempDemand)>maxFastCost)
        maxFastCost=getFastCost(tempDemand);

    fastCost=fastCost+this.getFastCost(tempDemand);
}
frame.setATC(fastCost/demandNodeSet.size());
frame.setMTC(maxFastCost);
}

//center field:
public synchronized Graph getGraph(){
    return this.graph;
}

public synchronized boolean isFinished(){
    return this.finish;
}

void sendRoamer(){
    Node supply;
    for(int i=0;i<this.graph.getSupplyNodeSet().size();i++){
        supply=(Node)this.graph.getSupplyNodeSet().elementAt(i);
        roamerCenter[i][0]=supply;
        new Roamer(i,supply,this).start();
    }
}

```

```

public Node getDummyNode(){
    return this.dummyNode;
}

public synchronized void takeKey(Roamer roamer){
    if(!finish){
        while(!available){
            try{
                //roamer is waiting to take!
                wait(1);
            }catch(InterruptedException e){
                //takeKey: cannot wait!
            }
        }
        //roamer took the key!
        available=false;
    }else{
        available=false;
        //roamer took the key, but center is finished!
    }
}

public synchronized void putKey(Roamer roamer){
    if(!finish){
        while(available){
            try{
                //roamer is waiting to put...
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        //roamer put the key!
        available=true;
    }else{
        //roamer put the key & center is already finished!
        available=true;
    }
}

//detour manager take key

```



## Center.java

```
public synchronized void takeKey(DetourManager dmr){
    if(!detourCenterFinish){
        while(!available){
            try{
                //dmr is waiting to take!
                wait(1);
            }catch(InterruptedException e){
                //takeKey: cannot wait!
            }
        }
        //dmr took the key!
        available=false;
    }else{
        available=false;
        //dmr took the key, but detour center is finished!
    }
}

public synchronized void putKey(DetourManager dmr){
    if(!detourCenterFinish){
        while(available){
            try{
                //dmr is waiting to put...
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        //dmr put the key!
        available=true;
    }else{
        //dmr put the key! detour center is already finished!
        available=true;
    }
}

//ma manager take key
public synchronized void takeKey(MAManager maMr){
    if(!maCenterFinish){
        while(!available){
            try{
                //maMr is waiting to take!
```

```

        wait(1);
    }catch(InterruptedException e){
        //takeKey: cannot wait!
    }
}
//maMr took the key!
available=false;
}else{
    available=false;
    //maMr took the key but maCenter is already finished!
}
}

public synchronized void putKey(MAManager maMr){
    if(!maCenterFinish){
        while(available){
            try{
                //maMr is waiting to put...
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        //maMr put the key!
        available=true;
    }else{
        //maMr put the key! and maCenter already finished!
        available=true;
    }
}

public void updateDetourCondition(){
    detourManagerNum--;
    if(detourManagerNum==0)
        detourCenterFinish=true;
}

public void updateMACondition(){
    maManagerNum--;
    if(maManagerNum==0)
        maCenterFinish=true;
}

```

```

//nodeCenter method:
//nodeCenter[node][0]: occupy(Boolean)
public synchronized void setOccupy(Node node,boolean occupy){
    nodeCenter[node.getLabel()][0]=new Boolean(occupy);
    notifyAll();
}

//nodeCenter[node][1]: distance(Double)
public synchronized void setDistance(Node node,double distance){
    nodeCenter[node.getLabel()][1]=new Double(distance);
}

public synchronized double getDistance(Node node){
    Double distance=(Double)nodeCenter[node.getLabel()][1];
    return distance.doubleValue();
}

//nodeCenter[node][2]: preNode(Node)
public synchronized void setPreNode(Node node,Node preNode){
    nodeCenter[node.getLabel()][2]=preNode;
}

public synchronized Node getPreNode(Node node){
    Node preNode=(Node)nodeCenter[node.getLabel()][2];
    return preNode;
}

//nodeCenter[node][3]: preEdge(Edge)
public synchronized void setPreEdge(Node node,Edge preEdge){
    nodeCenter[node.getLabel()][3]=preEdge;
}

public synchronized Edge getPreEdge(Node node){
    Edge preEdge=(Edge)nodeCenter[node.getLabel()][3];
    return preEdge;
}

//nodeCenter[node][4]: visitorSequence(Vector)
public synchronized void addViditor(Node node,Roamer roamer){
    Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
    if(!visitorSequence.contains(roamer))

```

```
        visitorSequence.addElement(roamer);
        nodeCenter[node.getLabel()][4]=visitorSequence;
    }

    public synchronized Vector getVisitorSequence(Node node){
        Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
        return visitorSequence;
    }

    public synchronized int getVisitorNum(Node node){
        Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
        int visitorNum=visitorSequence.size();
        return visitorNum;
    }

    public synchronized Roamer lastVisitor(Node node){
        Vector visitorSequence=(Vector)nodeCenter[node.getLabel()][4];
        Roamer lastVisitor=(Roamer)visitorSequence.lastElement();
        return lastVisitor;
    }

    //nodeCenter[node][5]: supply(Node)
    public Node getSupply(Node node){
        return (Node)nodeCenter[node.getLabel()][5];
    }

    //roamerCenter method:
    //roamerCenter[roamer][0]: myNode(Node)
    public synchronized void setMyNode(Roamer roamer,Node myNode){
        roamerCenter[roamer.getID()][0]=myNode;
    }

    public synchronized Node getMyNode(Roamer roamer){
        Node myNode=(Node)roamerCenter[roamer.getID()][0];
        return myNode;
    }

    //roamerCenter[roamer][1]: myEdge(Edge)
    public synchronized void setMyEdge(Roamer roamer,Edge myEdge){
        roamerCenter[roamer.getID()][1]=myEdge;
    }
}
```

**Center.java**

```
public synchronized Edge getMyEdge(Roamer roamer){
    Edge myEdge=(Edge)roamerCenter[roamer.getID()][1];
    return myEdge;
}

//roamerCenter[roamer][2]: currNode(Node)
public synchronized void setCurrNode(Roamer roamer,Node currNode){
    roamerCenter[roamer.getID()][2]=currNode;
}

public synchronized Node getCurrNode(Roamer roamer){
    Node currNode=(Node)roamerCenter[roamer.getID()][2];
    return currNode;
}

//roamerCenter[roamer][3]: routeSet(Graph)
public synchronized void setRouteSet(Roamer roamer,Graph routeSet){
    roamerCenter[roamer.getID()][3]=routeSet;
}

public synchronized void setRouteSet(Roamer roamer,Vector routeNodeSet,Vector routeEdgeSet){
    roamerCenter[roamer.getID()][3]=new Graph(routeNodeSet,routeEdgeSet);
}

public synchronized Graph getRouteSet(Roamer roamer){
    Graph routeSet=(Graph)roamerCenter[roamer.getID()][3];
    return routeSet;
}

public synchronized Vector getRouteNodeSet(Roamer roamer){
    Graph routeSet=(Graph)roamerCenter[roamer.getID()][3];
    Vector routeNodeSet=routeSet.getNodeSet();
    return routeNodeSet;
}

public synchronized Vector getRouteEdgeSet(Roamer roamer){
    Graph routeSet=(Graph)roamerCenter[roamer.getID()][3];
    Vector routeEdgeSet=routeSet.getEdgeSet();
    return routeEdgeSet;
}

public synchronized void addEdge(Roamer roamer,Edge edge){
```

```
        Graph routeSet=this.getRouteSet(roamer);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeEdgeSet.contains(edge))
            routeEdgeSet.addElement(edge);
        this.setRouteSet(roamer,routeNodeSet,routeEdgeSet);
    }

    public synchronized void addNode(Roamer roamer,Node node){
        Graph routeSet=this.getRouteSet(roamer);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeNodeSet.contains(node))
            routeNodeSet.addElement(node);
        this.setRouteSet(roamer,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeEdge(Roamer roamer,Edge edge){
        Graph routeSet=this.getRouteSet(roamer);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(routeEdgeSet.contains(edge)){
            routeEdgeSet.removeElement(edge);
        }
        this.setRouteSet(roamer,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeSubtree(Roamer roamer,Graph subtree){
        Graph routeSet=(Graph)this.roamerCenter[roamer.getID()][3];
        Vector tempNodeSet1=routeSet.getNodeSet();
        Vector tempEdgeSet1=routeSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();

        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            boolean nodeExist=tempNodeSet1.removeElement(tempNode);
        }

        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
```

```

        tempEdge=(Edge)tempEdgeSet2.elementAt(i);
        boolean edgeExist=tempEdgeSet1.removeElement(tempEdge);
    }
}

public synchronized void addSubtree(Roamer roamer,Graph subtree){
    Graph visitedRouteSet=(Graph)this.roamerCenter[roamer.getID()][3];
    Vector tempNodeSet1=visitedRouteSet.getNodeSet();
    Vector tempEdgeSet1=visitedRouteSet.getEdgeSet();
    Vector tempNodeSet2=subtree.getNodeSet();
    Vector tempEdgeSet2=subtree.getEdgeSet();
    Node tempNode;
    for(int i=0;i<tempNodeSet2.size();i++){
        tempNode=(Node)tempNodeSet2.elementAt(i);
        if(tempNodeSet1.contains(tempNode)){
        }else{
            tempNodeSet1.addElement(tempNode);
        }
    }

    Edge tempEdge;
    for(int i=0;i<tempEdgeSet2.size();i++){
        tempEdge=(Edge)tempEdgeSet2.elementAt(i);
        if(tempEdgeSet1.contains(tempEdge)){
        }else{
            tempEdgeSet1.addElement(tempEdge);
        }
    }
}

//fastCenter
//fastCenter[demand][0]: fastPath(Graph)
//fastCenter[demand][1]: supply(Node)
//fastCenter[demand][2]: fastPathLength(Double)
public double getFastCost(Node demand){
    Double fastCost=(Double)fastCenter[demand.getLabel()][2];
    return fastCost.doubleValue();
}

//detourCenter[edge][0]: downstream(Graph)
public void setDownstream(Edge ruinedEdge,Graph downstream){
    detourCenter[ruinedEdge.getLabel()][0]=downstream;
}

```

```
}

public Graph getDownstream(Edge ruinedEdge){
    return (Graph)detourCenter[ruinedEdge.getLabel()][0];
}

//detourCenter[edge][1]: upstream(Graph)
public void setUpstream(Edge ruinedEdge, Graph upstream){
    detourCenter[ruinedEdge.getLabel()][1]=upstream;
}

public Graph getUpstream(Edge ruinedEdge){
    return (Graph)detourCenter[ruinedEdge.getLabel()][1];
}

//detourCenter[edge][2]: mergeNode(Node)
public void setMergeNode(Edge ruinedEdge, Node mergeNode){
    detourCenter[ruinedEdge.getLabel()][2]=mergeNode;
}

public Node getMergeNode(Edge ruinedEdge){
    return (Node)detourCenter[ruinedEdge.getLabel()][2];
}

//detourCenter[edge][3]: accessNode(Node)
public void setAccessNode(Edge ruinedEdge, Node accessNode){
    detourCenter[ruinedEdge.getLabel()][3]=accessNode;
}

public Node getAccessNode(Edge ruinedEdge){
    return (Node)detourCenter[ruinedEdge.getLabel()][3];
}

//detourCenter[edge][4]: detourPath(Graph)
public void setDetourPath(Edge ruinedEdge, Graph detourPath){
    detourCenter[ruinedEdge.getLabel()][4]=detourPath;
}

public Graph getDetourPath(Edge ruinedEdge){
    return (Graph)detourCenter[ruinedEdge.getLabel()][4];
}
```



## Center.java

```
//detourCenter[edge][5]: systematicDetourCost(Double)
public void setSystematicDetourCost(Edge ruinedEdge,double sdc){
    detourCenter[ruinedEdge.getLabel()][5]=new Double(sdc);
}

public double getSystematicDetourCost(Edge ruinedEdge){
    Double tempDouble=(Double)detourCenter[ruinedEdge.getLabel()][5];
    return tempDouble.doubleValue();
}

//detourCenter[edge][6]: mergeCost(Vector)
public void setMergeCost(Edge ruinedEdge,double cost){
    detourCenter[ruinedEdge.getLabel()][6]=new Double(cost);
}

public double getMergeCost(Edge ruinedEdge){
    Double mergeCost=(Double)detourCenter[ruinedEdge.getLabel()][6];
    return mergeCost.doubleValue();
}

//maCenter
//maCenter[supply][0]: fastTree(Graph)
public Graph getFastTree(Node supply){
    return (Graph)maCenter[supply.getLabel()][0];
}

//maCenter[supply][1]: territory(Graph) 2ECON
public synchronized void setTerritory(Node supply,Graph territory){
    maCenter[supply.getLabel()][1]=territory;
}

public Graph getTerritory(Node supply){
    return (Graph)maCenter[supply.getLabel()][1];
}

//maCenter[supply][2]: source(Node)
public synchronized void setSource(Node supply,Node source){
    maCenter[supply.getLabel()][2]=source;
}

public Node getSource(Node supply){
    return (Node)maCenter[supply.getLabel()][2];
}
```

```
}

//maCenter[supply][3]: icpSet(Vector)
public synchronized void setICPSet(Node supply, Vector icpSet){
    maCenter[supply.getLabel()][3]=icpSet;
}

public Vector getICPSet(Node supply){
    return (Vector)maCenter[supply.getLabel()][3];
}

public int getICPNum(Node supply){
    Vector icpSet=(Vector)maCenter[supply.getLabel()][3];
    return icpSet.size();
}

//maCenter[supply][4]: maPath(Graph)
public synchronized void setMAPath(Node supply, Graph maPath){
    maCenter[supply.getLabel()][4]=maPath;
}

public Graph getMAPath(Node supply){
    return (Graph)maCenter[supply.getLabel()][4];
}

//maCenter[supply][5]: within territory supply-demand ratio(Double)
public synchronized void setTerritorySDR(Node supply, double territorySDR){
    maCenter[supply.getLabel()][5]=new Double(territorySDR);
}

public double getTerritorySDR(Node supply){
    Double territorySDR=(Double)maCenter[supply.getLabel()][5];
    return territorySDR.doubleValue();
}

//maCenter[supply][6]: mutual assistant supply-demand ratio(Double)
public synchronized void setMASdr(Node supply, double maSDR){
    maCenter[supply.getLabel()][6]=new Double(maSDR);
}

public double getMASdr(Node supply){
    Double maSDR=(Double)maCenter[supply.getLabel()][6];
```

**Center.java**

```
        return maSDR.doubleValue();
    }

    //maCenter[supply][7]: maCost > source to supply
    //including maPath cost & merge cost with respect to territory demandNum
    public synchronized void setMACost(Node supply,double maCost){
        maCenter[supply.getLabel()][7]=new Double(maCost);
    }

    public double getMACost(Node supply){
        Double maCost=(Double)maCenter[supply.getLabel()][7];
        return maCost.doubleValue();
    }
}
```

## Detourist.java

```
package emnet.thread;

import emnet.graph.Node;
import emnet.graph.Graph;
import java.util.Vector;
import emnet.graph.Edge;
import emnet.algorithm.GraphAlgorithm;

public class Detourist extends Thread{
    int id;
    Node start,supply,access;
    DetourManager dmr;
    Center center;

    Node currNode,preNode,myNode;
    Edge preEdge,myEdge,currRuinedEdge;
    Graph usableGraph,downstream,upstream,bridge,routeSet,detourPath;
    Vector routeNodeSet,routeEdgeSet,upstreamNodeSet;

    int downstreamDemandNum;
    double mergeCost,systematicDetourCost;

    boolean detouristFinish;

    public Detourist(int id,Node start,DetourManager dmr){
        super(""+id);
        this.id=id;
        this.start=start;
        this.dmr=dmr;
        center=dmr.getCenter();
        supply=dmr.getSupply();

        int edgeNum=dmr.getGraph().getEdgeSet().size();
        Node dummyNode=this.center.getDummyNode();
        dummyNode.setDummy();
        Edge dummyEdge=new Edge(edgeNum,start,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        dmr.getGraph().addNode(dummyNode);
```

```

        dmr.getGraph().addEdge(dummyEdge);

        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);
        dmr.setRouteSet(this,routeNodeSet,routeEdgeSet);

        dmr.setMyNode(this,dummyNode);
        dmr.setMyEdge(this,dummyEdge);
        dmr.setCurrNode(this,start);

        Vector nodeSet=dmr.getGraph().getNodeSet();
        Vector edgeSet=dmr.getGraph().getEdgeSet();
        Vector usableEdgeSet=new Vector();
        Edge tempEdge;
        for(int i=0;i<edgeSet.size();i++){
            tempEdge=(Edge)edgeSet.elementAt(i);
            if(tempEdge!=dmr.getCurrRuinedEdge())
                usableEdgeSet.addElement(tempEdge);
        }
        usableGraph=new Graph(nodeSet,usableEdgeSet);

        downstream=dmr.getCurrDownstream();
        upstream=dmr.getCurrUpstream();
        upstreamNodeSet=upstream.getNodeSet();

        currRuinedEdge=dmr.getCurrRuinedEdge();
        downstreamDemandNum=dmr.getDownstreamDemandNum();
        mergeCost=GraphAlgorithm.getMergeCost(this,start);
        systematicDetourCost=0.0;

        dmr.setCurrDetourCost(this,dmr.getMyNode(this),mergeCost);
        dmr.setPreNode(this,start,dummyNode);
        dmr.setPreEdge(this,start,dummyEdge);

        detouristFinish=false;
    }

    public int getID(){
        return id;
    }

```

```
public Graph getUsableGraph(){
    return usableGraph;
}

public Graph getDownstream(){
    return downstream;
}

public Graph getUpstream(){
    return upstream;
}

public DetourManager getDetourManager(){
    return dmr;
}

public Center getCenter(){
    return center;
}

public Node getMergeNode(){
    return start;
}

public Node getAccessNode(){
    return access;
}

public Edge getRuinedEdge(){
    return currRuinedEdge;
}

public double getMergeCost(){
    return mergeCost;
}

public double getSystematicDetourCost(){
    return systematicDetourCost;
}

public Graph getDetourPath(){
```

```

        return detourPath;
    }

    public void run(){

        dmr.takeKey(this);

        //::map init::
        Vector edgeSet=center.getGraph().getEdgeSet();
        Edge tempEdge;
        for(int i=0;i<edgeSet.size();i++){
            tempEdge=(Edge)edgeSet.elementAt(i);
            if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
                tempEdge.setNeutralEdge();
        }
        //::map init::

        detouring:
        while(!detouristFinish){
            if(dmr.getCurrNode(this)==supply){
                double
sdc=dmr.getCurrDetourCost(this,dmr.getMyNode(this))+dmr.getMyEdge(this).getWeight()*downstreamDemandNum;
                dmr.setCurrDetourCost(this,dmr.getCurrNode(this),sdc);

                dmr.addNode(this,supply);
                dmr.addEdge(this,dmr.getMyEdge(this));

                dmr.setPreNode(this,supply,dmr.getMyNode(this));
                dmr.setPreEdge(this,supply,dmr.getMyEdge(this));

                dmr.updatMinSDC(this,dmr.getCurrDetourCost(this,dmr.getSupply()));

                dmr.getMyEdge(this).setDetourTestEdge(true);

                break detouring;
            }else if(dmr.getMinSDC(this)!=0.0 &&
dmr.getMinSDC(this)<dmr.getCurrDetourCost(this,dmr.getMyNode(this))){
                //some detourist has already found a shorter detour path
                break detouring;
            }else{
                dmr.addNode(this,dmr.getCurrNode(this));
                dmr.addEdge(this,dmr.getMyEdge(this));
            }
        }
    }

```

```

        dmr.setPreNode(this, dmr.getCurrNode(this), dmr.getMyNode(this));
        dmr.setPreEdge(this, dmr.getCurrNode(this), dmr.getMyEdge(this));

        double
sdc=dmr.getCurrDetourCost(this, dmr.getMyNode(this))+dmr.getMyEdge(this).getWeight()*downstreamDemandNum;
        dmr.setCurrDetourCost(this, dmr.getCurrNode(this), sdc);

        dmr.getMyEdge(this).setDetourTestEdge(true);
        dmr.setMyNode(this, dmr.getCurrNode(this));

        //find currEdge & currNode, assign new myNode
        dijkstra(null);

        try{
            sleep(1);
        }catch(InterruptedException ex){
        }

    }

}

//set access node
Vector detourPathNodeSet=new Vector(), detourPathEdgeSet=new Vector();
Node tempCurrNode=dmr.getSupply();
detourPathNodeSet.addElement(tempCurrNode);

Edge tempPreEdge;
Node n1, n2;
do{
    tempPreEdge=dmr.getPreEdge(this, tempCurrNode);

    if(!detourPathEdgeSet.contains(tempPreEdge))
        detourPathEdgeSet.addElement(tempPreEdge);

    n1=tempPreEdge.getN1();
    n2=tempPreEdge.getN2();

    if(!detourPathNodeSet.contains(n1))
        detourPathNodeSet.addElement(n1);
    if(!detourPathNodeSet.contains(n2))

```



```

        detourPathNodeSet.addElement(n2);

        if(upstreamNodeSet.contains(n1) && !upstreamNodeSet.contains(n2)){
            access=n1;
        }else if(upstreamNodeSet.contains(n2) && !upstreamNodeSet.contains(n1)){
            access=n2;
        }else{
            access=null;
        }

        tempCurrNode=dmr.getPreNode(this,tempCurrNode);
    }while(tempCurrNode!=start);

    if(!detourPathNodeSet.contains(start))
        detourPathNodeSet.addElement(start);

    //set detourPath
    detourPath=new Graph(detourPathNodeSet,detourPathEdgeSet);

    //set sdc
    systematicDetourCost=dmr.getCurrDetourCost(this,supply);
    dmr.setSystematicDetourCost(this,systematicDetourCost);

    dmr.updateCurrRuinedEdgeFinish();
    dmr.updateDetourCenter(this);

    dmr.putKey(this);
}

void dijkstra(Edge canceledEdge){

    //find out the best incident edge
    //define new myEdge & currNode
    Vector tempRouteNodeSet=(Vector)dmr.getRouteNodeSet(this);

    Vector incidentEdgeSet=GraphAlgorithm.getIncidentEdges(this,tempRouteNodeSet);
    Vector exclusiveIncidentEdgeSet=new Vector();
    for(int i=0;i<incidentEdgeSet.size();i++){
        if(!dmr.getRouteEdgeSet(this).contains(incidentEdgeSet.elementAt(i))){
            exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
        }
    }
}

```

```

Edge tempEdge;
Node n1,n2;

if(exclusiveIncidentEdgeSet.size(>1){
    tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
    dmr.setMyEdge(this,tempEdge);
    n1=tempEdge.getN1();
    n2=tempEdge.getN2();
    if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
        dmr.setMyNode(this,n1);
        dmr.setCurrNode(this,n2);
    }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
        dmr.setMyNode(this,n2);
        dmr.setCurrNode(this,n1);
    }else{
        //dijkstra error 1: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
    }

    double
min=dmr.getCurrDetourCost(this,dmr.getMyNode(this))+dmr.getMyEdge(this).getWeight()*downstreamDemandNum;

    for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
        tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
        n1=tempEdge.getN1();
        n2=tempEdge.getN2();
        if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
            if((dmr.getCurrDetourCost(this,n1)+tempEdge.getWeight()*downstreamDemandNum)<min){
                min=dmr.getCurrDetourCost(this,n1)+tempEdge.getWeight()*downstreamDemandNum;
                dmr.setMyEdge(this,tempEdge);
                dmr.setMyNode(this,n1);
                dmr.setCurrNode(this,n2);
            }
        }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
            if((dmr.getCurrDetourCost(this,n2)+tempEdge.getWeight()*downstreamDemandNum)<min){
                min=dmr.getCurrDetourCost(this,n2)+tempEdge.getWeight()*downstreamDemandNum;
                dmr.setMyEdge(this,tempEdge);
                dmr.setMyNode(this,n2);
                dmr.setCurrNode(this,n1);
            }
        }else{
            //dijkstra error 2: incident edge error!

```

```

    }
}

}else if(exclusiveIncidentEdgeSet.size()==1){
    tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
    dmr.setMyEdge(this,tempEdge);
    n1=tempEdge.getN1();
    n2=tempEdge.getN2();
    if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
        dmr.setMyNode(this,n1);
        dmr.setCurrNode(this,n2);
    }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
        dmr.setMyNode(this,n2);
        dmr.setCurrNode(this,n1);
    }else{
        //dijkstra error 3: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
    }
}else{
    detouristFinish=true;
    //renders all nodes visited
}
}
}

```

## DetourManager.java

```
package emnet.thread;

import emnet.graph.Node;
import emnet.graph.Graph;
import java.util.Vector;
import emnet.graph.Edge;
import emnet.algorithm.GraphAlgorithm;

public class DetourManager extends Thread{

    int id;
    Center center;
    Node supply;
    Graph graph,fastTree,territory;
    Vector edgeSet,nodeSet,fastTreeEdgeSet,fastTreeNodeSet,detourists,territoryNodeSet,territoryEdgeSet;
    int detouristNum,downstreamDemandNum;

    //nodeDept
    //nodeDept[detourist][node][0]: currDetourDist
    //nodeDept[detourist][node][1]: preNode
    //nodeDept[detourist][node][2]: preEdge
    Object[][][] nodeDept;

    //detourDept
    //detourDept[detourist][0]: myNode(Node)
    //detourDept[detourist][1]: myEdge(Edge)
    //detourDept[detourist][2]: currNode(Node)
    //detourDept[detourist][3]: routeSet(Graph)
    //detourDept[detourist][4]: detourLength(Double)
    //deoutrDept[detourist][5]: bridge(Graph)
    Object[][] detourDept;

    //sdcDept
    //sdcDept[ruinedEdge][0]: minSDC
    Object[][] sdcDept;

    Edge currRuinedEdge;
    Graph currUpstream,currDownstream;
    Vector currDownstreamNodeSet;

    boolean currRuinedEdgeFinish,available;
```

```
//one manager controls one territory
public DetourManager(int id,Center center,Node supply){
    this.id=id;
    this.center=center;
    this.supply=supply;

    graph=center.getGraph();
    edgeSet=graph.getEdgeSet();
    nodeSet=graph.getNodeSet();

    fastTree=center.getFastTree(supply);
    fastTreeEdgeSet=fastTree.getEdgeSet();
    fastTreeNodeSet=fastTree.getNodeSet();

//    detouristNum=fastTreeNodeSet.size();

    nodeDept=new Object[fastTreeNodeSet.size()][nodeSet.size()][3];
    detourDept=new Object[fastTreeNodeSet.size()][6];

    if(fastTreeEdgeSet.size()!=0){
        Edge tempFastTreeEdge=(Edge)fastTreeEdgeSet.elementAt(0);
        int maxEdgeLabel=tempFastTreeEdge.getLabel();
        for(int j=1;j<fastTreeEdgeSet.size();j++){
            tempFastTreeEdge=(Edge)fastTreeEdgeSet.elementAt(j);
            if(tempFastTreeEdge.getLabel()>maxEdgeLabel)
                maxEdgeLabel=tempFastTreeEdge.getLabel();
        }
        sdcDept=new Object[maxEdgeLabel+1][1];
    }
}

public void run(){

    if(fastTreeEdgeSet==null){
        center.updateDetourCondition();
        destroy();
    }
    center.takeKey(this);

    for(int i=0;i<fastTreeEdgeSet.size();i++){
```

## DetourManager.java

```
//initialization
currRuinedEdgeFinish=false;
available=true;

for(int j=0;j<fastTreeNodeSet.size();j++){
    Node tempNode;
    for(int k=0;k<fastTreeNodeSet.size();k++){
        tempNode=(Node)fastTreeNodeSet.elementAt(j);
        nodeDept[j][tempNode.getLabel()][0]=new Double(0.0);
    }
}

for(int j=0;j<detouristNum;j++){
    detourDept[j][0]=null;
    detourDept[j][1]=null;
    detourDept[j][2]=null;
    detourDept[j][3]=null;
    detourDept[j][4]=new Double(0.0);
    detourDept[j][5]=null;
}

Edge tempFastTreeEdge;
for(int j=0;j<fastTreeEdgeSet.size();j++){
    tempFastTreeEdge=(Edge)fastTreeEdgeSet.elementAt(j);
    sdcDept[tempFastTreeEdge.getLabel()][0]=new Double(0.0);
}

currRuinedEdge=(Edge)fastTreeEdgeSet.elementAt(i);
currUpstream=GraphAlgorithm.getSubtreeWithSupply(fastTree,currRuinedEdge);
currDownstream=GraphAlgorithm.getSubtreeWithoutSupply(fastTree,currRuinedEdge);
currDownstreamNodeSet=currDownstream.getNodeSet();
detouristNum=currDownstreamNodeSet.size();

Node tempNode;
downstreamDemandNum=0;
for(int k=0;k<currDownstreamNodeSet.size();k++){
    tempNode=(Node)currDownstreamNodeSet.elementAt(k);
    if(tempNode.isDemand())
        downstreamDemandNum++;
}

//one detourist tests from one node when one currRuinedEdge is simulated
```

**DetourManager.java**

```
    Detourist tempDetourist;
    detourists=new Vector();
    for(int j=0;j<currDownstreamNodeSet.size();j++){
        tempNode=(Node)currDownstreamNodeSet.elementAt(j);
        tempDetourist=new Detourist(j,tempNode,this);
        detourists.addElement(tempDetourist);
        tempDetourist.start();
    }

    //dmr waiting the currRuinedEdge finish
    while(!currRuinedEdgeFinish){

    }

    //currRuinedEdge finish: all situations simulated
    //set detour edges on the shortest detour route
    Detourist bestDetourist=(Detourist)detourists.elementAt(0);
    Detourist tempDetourist1;
    double minSDC=bestDetourist.getSystematicDetourCost();
    for(int j=1;j<detourists.size();j++){
        tempDetourist1=(Detourist)detourists.elementAt(j);
        if(tempDetourist1.getSystematicDetourCost()<minSDC){
            minSDC=tempDetourist1.getSystematicDetourCost();
            bestDetourist=tempDetourist1;
        }
    }

    Vector bestDetourPathEdgeSet=bestDetourist.getDetourPath().getEdgeSet();
    Edge tempEdge1;
    for(int j=0;j<bestDetourPathEdgeSet.size();j++){
        tempEdge1=(Edge)bestDetourPathEdgeSet.elementAt(j);
        tempEdge1.setDetourEdge();
    }
}

//::maCenter[supply][1]: territory(Graph) 2ECON::
Graph territory=fastTree;
Edge tempRuinedEdge;
Graph tempDetourPath;
Vector tempDetourPathNodeSet,tempDetourPathEdgeSet;
Node tempDetourPathNode;
Edge tempDetourPathEdge;
for(int i=0;i<fastTreeEdgeSet.size();i++){
```

**DetourManager.java**

```
tempRuinedEdge=(Edge)fastTreeEdgeSet.elementAt(i);
tempDetourPath=center.getDetourPath(tempRuinedEdge);
if(tempDetourPath!=null){
    tempDetourPathNodeSet=tempDetourPath.getNodeSet();
    tempDetourPathEdgeSet=tempDetourPath.getEdgeSet();
    if(tempDetourPathNodeSet!=null){
        for(int j=0;j<tempDetourPathNodeSet.size();j++){
            tempDetourPathNode=(Node)tempDetourPathNodeSet.elementAt(j);
            if(!territory.hasNode(tempDetourPathNode))
                territory.addNode(tempDetourPathNode);
        }
    }
    if(tempDetourPathEdgeSet!=null){
        for(int j=0;j<tempDetourPathEdgeSet.size();j++){
            tempDetourPathEdge=(Edge)tempDetourPathEdgeSet.elementAt(j);
            if(!territory.hasEdge(tempDetourPathEdge))
                territory.addEdge(tempDetourPathEdge);
        }
    }
}
else{
    //tempDetourPath is null!
}
}

center.setTerritory(supply,territory);
//::maCenter[supply][1]: territory(Graph) 2ECON::

center.updateDetourCondition();
center.putKey(this);
}

//detourDept method:
public synchronized void takeKey(Detourist detourist){
    if(!currRuinedEdgeFinish){
        while(!available){
            try{
                wait(1);
            }catch(InterruptedException e){
                //takeKey: cannot wait!
            }
        }
    }
    available=false;
```



**DetourManager.java**

```
    }else{
        available=false;
    }
}

public synchronized void putKey(Detourist detourist){
    if(!isCurrRuinedEdgeFinish()){
        while(available){
            try{
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        available=true;
    }else{
        available=true;
    }
}

public synchronized boolean isFinished(){
    return currRuinedEdgeFinish;
}

void sendDetourist(Graph downstream){
    Vector fastTreeNodeSet=downstream.getNodeSet();
    Node tempNode;
    for(int i=0;i<fastTreeNodeSet.size();i++){
        tempNode=(Node)fastTreeNodeSet.elementAt(i);
        new Detourist(i,tempNode,this).start();
    }
}

public synchronized void setCurrRuinedEdgeFinish(boolean currRuinedEdgeFinish){
    this.currRuinedEdgeFinish=currRuinedEdgeFinish;
}

public Edge getCurrRuinedEdge(){
    return currRuinedEdge;
}

public Graph getCurrUpstream(){
```

**DetourManager.java**

```
        return currUpstream;
    }

    public Graph getCurrDownstream(){
        return currDownstream;
    }

    public Graph getGraph(){
        return center.getGraph();
    }

    public void setPreNode(Detourist detourist,Node node,Node preNode){
        nodeDept[detourist.getID()][node.getLabel()][1]=preNode;
    }

    public Node getPreNode(Detourist detourist,Node node){
        return (Node)nodeDept[detourist.getID()][node.getLabel()][1];
    }

    public void setPreEdge(Detourist detourist,Node node,Edge preEdge){
        nodeDept[detourist.getID()][node.getLabel()][2]=preEdge;
    }

    public Edge getPreEdge(Detourist detourist,Node node){
        return (Edge)nodeDept[detourist.getID()][node.getLabel()][2];
    }

    public int getDownstreamDemandNum(){
        return downstreamDemandNum;
    }

    public Center getCenter(){
        return center;
    }

    public Node getSupply(){
        return supply;
    }

    public int getID(){
        return id;
    }
}
```

```

public synchronized void updateCurrRuinedEdgeFinish(){
    detouristNum--;
    if(detouristNum==0){
        currRuinedEdgeFinish=true;
    }else{
        currRuinedEdgeFinish=false;
    }
}

public synchronized boolean isCurrRuinedEdgeFinish(){
    return currRuinedEdgeFinish;
}

public synchronized void updateDetourCenter(Detourist detourist){
    Center center=detourist.getCenter();
    DetourManager dmr=detourist.getDetourManager();
    Edge ruinedEdge=detourist.getRuinedEdge();

    //detourCenter[edge][0]: downstream(Graph)
    //detourCenter[edge][1]: upstream(Graph)
    //detourCenter[edge][2]: mergeNode(Node)
    //detourCenter[edge][3]: accessNode(Node)
    //detourCenter[edge][4]: detourPath(Graph)
    //detourCenter[edge][5]: systematicDetourCost(Double)
    //detourCenter[edge][6]: mergeCost(Vector)

    //maCenter[supply][1]: territory(Graph) 2ECON

    double detouristDetourLength=dmr.getSystematicDetourCost(detourist);
    double centerDetourLengthRecord=center.getSystematicDetourCost(detourist.getRuinedEdge());
    if((centerDetourLengthRecord==0.0 || detouristDetourLength<centerDetourLengthRecord){

        center.setDownstream(ruinedEdge,detourist.getDownstream());
        center.setUpstream(ruinedEdge,detourist.getUpstream());
        center.setMergeNode(ruinedEdge,detourist.getMergeNode());
        center.setAccessNode(ruinedEdge,detourist.getAccessNode());
        center.setDetourPath(ruinedEdge,detourist.getDetourPath());
        center.setSystematicDetourCost(ruinedEdge,detouristDetourLength);
        center.setMergeCost(ruinedEdge,detourist.getMergeCost());

        //detouristDetourLength is smaller than center record

```

## DetourManager.java

```
    }else{
        //detouristDetourLength is larger than center record
    }
}

//detourDept method:
//detourDept[detourist][0]: myNode(Node)
public synchronized void setMyNode(Detourist detourist,Node myNode){
    detourDept[detourist.getID()][0]=myNode;
}

public synchronized Node getMyNode(Detourist detourist){
    Node myNode=(Node)detourDept[detourist.getID()][0];
    return myNode;
}

//detourDept[detourist][1]: myEdge(Edge)
public synchronized void setMyEdge(Detourist detourist,Edge myEdge){
    detourDept[detourist.getID()][1]=myEdge;
}

public synchronized Edge getMyEdge(Detourist detourist){
    Edge myEdge=(Edge)detourDept[detourist.getID()][1];
    return myEdge;
}

//detourDept[detourist][2]: currNode(Node)
public synchronized void setCurrNode(Detourist detourist,Node currNode){
    detourDept[detourist.getID()][2]=currNode;
}

public synchronized Node getCurrNode(Detourist detourist){
    Node currNode=(Node)detourDept[detourist.getID()][2];
    return currNode;
}

//detourDept[detourist][3]: routeSet(Graph)
public synchronized void setRouteSet(Detourist detourist,Graph routeSet){
    detourDept[detourist.getID()][3]=routeSet;
}

public synchronized void setRouteSet(Detourist detourist,Vector routeNodeSet,Vector routeEdgeSet){
```

**DetourManager.java**

```
        detourDept[detourist.getID()][3]=new Graph(routeNodeSet,routeEdgeSet);
    }

    public synchronized Graph getRouteSet(Detourist detourist){
        Graph routeSet=(Graph)detourDept[detourist.getID()][3];
        return routeSet;
    }

    public synchronized Vector getRouteNodeSet(Detourist detourist){
        Graph routeSet=(Graph)detourDept[detourist.getID()][3];
        Vector routeNodeSet=routeSet.getNodeSet();
        return routeNodeSet;
    }

    public synchronized Vector getRouteEdgeSet(Detourist detourist){
        Graph routeSet=(Graph)detourDept[detourist.getID()][3];
        Vector routeEdgeSet=routeSet.getEdgeSet();
        return routeEdgeSet;
    }

    public synchronized void addEdge(Detourist detourist,Edge edge){
        Graph routeSet=this.getRouteSet(detourist);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeEdgeSet.contains(edge))
            routeEdgeSet.addElement(edge);
        this.setRouteSet(detourist,routeNodeSet,routeEdgeSet);
    }

    public synchronized void addNode(Detourist detourist,Node node){
        Graph routeSet=this.getRouteSet(detourist);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeNodeSet.contains(node))
            routeNodeSet.addElement(node);
        this.setRouteSet(detourist,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeEdge(Detourist detourist,Edge edge){
        Graph routeSet=this.getRouteSet(detourist);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
```

**DetourManager.java**

```
        if(routeEdgeSet.contains(edge)){
            routeEdgeSet.removeElement(edge);
        }
        this.setRouteSet(detourist,routeNodeSet,routeEdgeSet);
    }

    public synchronized void removeSubtree(Detourist detourist,Graph subtree){
        Graph routeSet=(Graph)this.detourDept[detourist.getID()][3];
        Vector tempNodeSet1=routeSet.getNodeSet();
        Vector tempEdgeSet1=routeSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();

        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            boolean nodeExist=tempNodeSet1.removeElement(tempNode);
        }

        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
            tempEdge=(Edge)tempEdgeSet2.elementAt(i);
            boolean edgeExist=tempEdgeSet1.removeElement(tempEdge);
        }
    }

    public synchronized void addSubtree(Detourist detourist,Graph subtree){

        Graph visitedRouteSet=(Graph)this.detourDept[detourist.getID()][3];
        Vector tempNodeSet1=visitedRouteSet.getNodeSet();
        Vector tempEdgeSet1=visitedRouteSet.getEdgeSet();
        Vector tempNodeSet2=subtree.getNodeSet();
        Vector tempEdgeSet2=subtree.getEdgeSet();

        Node tempNode;
        for(int i=0;i<tempNodeSet2.size();i++){
            tempNode=(Node)tempNodeSet2.elementAt(i);
            if(tempNodeSet1.contains(tempNode)){
            }else{
                tempNodeSet1.addElement(tempNode);
            }
        }
    }
```

```

        Edge tempEdge;
        for(int i=0;i<tempEdgeSet2.size();i++){
            tempEdge=(Edge)tempEdgeSet2.elementAt(i);
            if(tempEdgeSet1.contains(tempEdge)){
            }else{
                tempEdgeSet1.addElement(tempEdge);
            }
        }
    }

    //detourDept[detourist][4]: detourLength(Double)
    public void setSystematicDetourCost(Detourist detourist, double sdc){
        detourDept[detourist.getID()][4]=new Double(sdc);
    }

    public double getSystematicDetourCost(Detourist detourist){
        Double sdc=(Double)detourDept[detourist.getID()][4];
        return sdc.doubleValue();
    }

    public void setCurrDetourCost(Detourist detourist,Node node,double sdc){
        nodeDept[detourist.getID()][node.getLabel()][0]=new Double(sdc);
    }

    public double getCurrDetourCost(Detourist detourist,Node node){
        Double tempDouble=(Double)nodeDept[detourist.getID()][node.getLabel()][0];
        return tempDouble.doubleValue();
    }

    //detourCenter:
    public synchronized void updateMinSDC(double sdc){
        double currSDC=center.getSystematicDetourCost(currRuinedEdge);
        if(currSDC==0.0 || sdc<currSDC)
            center.setSystematicDetourCost(currRuinedEdge,sdc);
    }

    public synchronized double getMinSDC(){
        return center.getSystematicDetourCost(currRuinedEdge);
    }

```

## DetourManager.java

```
//deoutrDept[detourist][5]: bridge(Graph)
//sdcDept
//sdcDept[ruinedEdge][0]: minSDC
public synchronized void updatMinSDC(Detourist detourist,double sdc){
    Edge ruinedEdge=detourist.getRuinedEdge();
    DetourManager dmr=detourist.getDetourManager();
    double currSDC=dmr.getMinSDC(detourist);
    if(sdc<currSDC)
        sdcDept[ruinedEdge.getLabel()][0]=new Double(sdc);
}

public synchronized double getMinSDC(Detourist detourist){
    Edge ruinedEdge=detourist.getRuinedEdge();
    Double minSDC=(Double)sdcDept[ruinedEdge.getLabel()][0];
    return minSDC.doubleValue();
}
}
```



## Helper.java

```
package emnet.thread;

import java.util.Vector;
import emnet.graph.Node;
import emnet.graph.Graph;
import emnet.graph.Edge;

public class Helper extends Thread{
    int id;
    MAManager maMr;
    Node start,source;

    Node currNode,preNode,myNode;
    Edge preEdge,myEdge;

    Graph maPath;

    //maCost = mergeCost + demandNum * bridgeLength(maPath)
    double maCost;

    //for merge cost
    //walkerCenter[node][]:
    //nodeDept
    //nodeDept[node][0]: currCost
    //nodeDept[node][1]: preNode
    //nodeDept[node][2]: preEdge
    Object[][] nodeDept;

    //walkerDept
    //walkerDept[0]: myNode(Node)
    //walkerDept[1]: myEdge(Edge)
    //walkerDept[2]: currNode(Node)
    //walkerDept[3]: routeSet(Graph)
    Object[] walkerDept;

    boolean helperFinish,longer,exclusivelsZero;

    public Helper(int id,Node start,MAManager maMr){

        this.id=id;
        this.start=start;
```

## Helper.java

```
this.maMr=maMr;
this.source=null;

//nodeDept
nodeDept=new Object[maMr.getGraph().getNodeSet().size()][3];

//walkerDept
walkerDept=new Object[4];

Node dummyNode=maMr.getCenter().getDummyNode();
dummyNode.setDummy();
int edgeNum=maMr.getGraph().getEdgeSet().size();
Edge dummyEdge=new Edge(edgeNum,start,dummyNode,0.0);
dummyEdge.setDummyEdge();

maMr.getGraph().addNode(dummyNode);
maMr.getGraph().addEdge(dummyEdge);

//helper init, set to maMr
Vector routeNodeSet=new Vector();
routeNodeSet.addElement(dummyNode);
Vector routeEdgeSet=new Vector();
routeEdgeSet.addElement(dummyEdge);

maMr.setRouteSet(this,routeNodeSet,routeEdgeSet);
maMr.setMyNode(this,dummyNode);
maMr.setMyEdge(this,dummyEdge);
maMr.setCurrNode(this,start);

double mergeCost=getECONMergeCost(start);

maMr.setCurrMACost(this,maMr.getMyNode(this),mergeCost);
maMr.setPreNode(this,start,dummyNode);
maMr.setPreEdge(this,start,dummyEdge);
maCost=0.0;
helperFinish=false;
longer=false;
exclusivelsZero=false;
}

public int getID(){
    return id;
}
```

```

    }

    public Node getStart(){
        return start;
    }

    public MAManager getMAManager(){
        return maMr;
    }

    public double getMACost(){
        return maCost;
    }

    public Graph getMAPath(){
        return maPath;
    }

    public Node getSource(){
        return source;
    }

    public void run(){
        maMr.takeKey(this);

        //::map init::
        Vector edgeSet=maMr.getCenter().getGraph().getEdgeSet();
        Edge tempEdge;
        for(int i=0;i<edgeSet.size();i++){
            tempEdge=(Edge)edgeSet.elementAt(i);
            if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
                tempEdge.setNeutralEdge();
        }
        //::map init::

        maHelping:
        while(!helperFinish){

            if(maMr.getCurrNode(this).isSupply() && maMr.getCurrNode(this)!=maMr.getSupply()){

                source=maMr.getCurrNode(this);
            }
        }
    }

```

```

        double
currMACost=maMr.getCurrMACost(this,maMr.getMyNode(this))+maMr.getMyEdge(this).getWeight()*maMr.getDemamNum();
        maMr.setCurrMACost(this,maMr.getCurrNode(this),currMACost);

        maMr.addNode(this,maMr.getCurrNode(this));
        maMr.addEdge(this,maMr.getMyEdge(this));

        maMr.setPreNode(this,maMr.getCurrNode(this),maMr.getMyNode(this));
        maMr.setPreEdge(this,maMr.getCurrNode(this),maMr.getMyEdge(this));

        maMr.updateMinMACost(maMr.getCurrMACost(this,maMr.getCurrNode(this)));

        maMr.getMyEdge(this).setMATestEdge(true);

        break maHelping;
    }else if(maMr.getMinMACost()!=0.0 &&
maMr.getMinMACost(<maMr.getCurrMACost(this,maMr.getMyNode(this)))
        //some detourist has already found a shorter detour path
        longer=true;
        break maHelping;
    }else{
        maMr.addNode(this,maMr.getCurrNode(this));
        maMr.addEdge(this,maMr.getMyEdge(this));

        maMr.setPreNode(this,maMr.getCurrNode(this),maMr.getMyNode(this));
        maMr.setPreEdge(this,maMr.getCurrNode(this),maMr.getMyEdge(this));

        double
currMACost=maMr.getCurrMACost(this,maMr.getMyNode(this))+maMr.getMyEdge(this).getWeight()*maMr.getDemamNum();
        maMr.setCurrMACost(this,maMr.getCurrNode(this),currMACost);

        maMr.getMyEdge(this).setMATestEdge(true);
        maMr.setMyNode(this,maMr.getCurrNode(this));

        //find currEdge & currNode, assign new myNode
        dijkstra();

        try{
            sleep(1);
        }catch(InterruptedException ex){
            System.out.println("maMr "+maMr.getID()+", helper "+getID()+" cannot sleep: "+ex);
        }
    }

```

```

    }
}

//helper finished!
//maPath setting:
if(!longer && !exclusivelsZero){
    Vector maPathNodeSet=new Vector();
    Vector maPathEdgeSet=new Vector();

    Node tempCurrNode=maMr.getCurrNode(this);
    Edge tempPreEdge;

    maPathSetting:
    do{
        if(!maPathNodeSet.contains(tempCurrNode))
            maPathNodeSet.addElement(tempCurrNode);

        tempPreEdge=maMr.getPreEdge(this,tempCurrNode);

        //tempPreEdge!=null
        if(!tempPreEdge.isDummyEdge()){
            if(!maPathEdgeSet.contains(tempPreEdge))
                maPathEdgeSet.addElement(tempPreEdge);

            tempCurrNode=maMr.getPreNode(this,tempCurrNode);
        }else{
            break maPathSetting;
        }
    }while(tempCurrNode!=start);

    if(!maPathNodeSet.contains(start))
        maPathNodeSet.addElement(start);

    maPath=new Graph(maPathNodeSet,maPathEdgeSet);

    //update mutual assistant path to center
    //maCost setting:
    maCost=maMr.getCurrMACost(this,maMr.getCurrNode(this));
}
maMr.updateMAFinish();
maMr.putKey(this);

```

```

    }

    void dijkstra(){
        //find out the best incident edge
        //define new myEdge & currNode
        Graph tempRouteSet=(Graph)maMr.getRouteSet(this);
        Vector tempRouteNodeSet=(Vector)tempRouteSet.getNodeSet();

        Vector incidentEdgeSet=getIncidentEdgeSet(maMr.getUsableGraph(),tempRouteNodeSet);

        Vector exclusiveIncidentEdgeSet=new Vector();
        for(int i=0;i<incidentEdgeSet.size();i++){
            if(!maMr.getRouteEdgeSet(this).contains(incidentEdgeSet.elementAt(i))){
                exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
            }
        }

        Edge tempEdge;
        Node n1,n2;

        if(exclusiveIncidentEdgeSet.size()>1){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
            maMr.setMyEdge(this,tempEdge);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                maMr.setMyNode(this,n1);
                maMr.setCurrNode(this,n2);
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                maMr.setMyNode(this,n2);
                maMr.setCurrNode(this,n1);
            }else{
                //dijkstra error 1: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
            }
            double
            min=maMr.getCurrMACost(this,maMr.getMyNode(this))+maMr.getMyEdge(this).getWeight()*maMr.getDemandNum();
            for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
                tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
                n1=tempEdge.getN1();
                n2=tempEdge.getN2();
                if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                    if(maMr.getCurrMACost(this,n1)+tempEdge.getWeight()*maMr.getDemandNum()<min){

```

```

        min=maMr.getCurrMACost(this,n1)+tempEdge.getWeight()*maMr.getDemamNum();
        maMr.setMyEdge(this,tempEdge);
        maMr.setMyNode(this,n1);
        maMr.setCurrNode(this,n2);
    }
}
else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
    if(maMr.getCurrMACost(this,n2)+tempEdge.getWeight()*maMr.getDemamNum()<min){
        min=maMr.getCurrMACost(this,n2)+tempEdge.getWeight()*maMr.getDemamNum();
        maMr.setMyEdge(this,tempEdge);
        maMr.setMyNode(this,n2);
        maMr.setCurrNode(this,n1);
    }
}
else{
    //dijkstra error 2: incident edge error!"
}
}
}
else if(exclusiveIncidentEdgeSet.size()==1){
    tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
    maMr.setMyEdge(this,tempEdge);
    n1=tempEdge.getN1();
    n2=tempEdge.getN2();
    if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
        maMr.setMyNode(this,n1);
        maMr.setCurrNode(this,n2);
    }
    else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
        maMr.setMyNode(this,n2);
        maMr.setCurrNode(this,n1);
    }
    else{
        //dijkstra error 3: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
    }
}
else{
    exclusivesZero=true;
    helperFinish=true;
    //renders all nodes visited
}
}

double getECONMergeCost(Node node){
    Walker walker=new Walker(this,node);
    walker.start();
}

```

## Helper.java

```
while(!walker.isFinish()){
    //wait for walker to calculate merge cost
}
return walker.getTerritoryMergeCost();
}

Vector getIncidentEdgeSet(Graph usableGraph,Vector routeNodeSet){
    Edge myEdge=maMr.getMyEdge(this);

    Vector incidentEdges=new Vector();
    Edge tempEdge;
    Node n1,n2;

    for(int i=0;i<routeNodeSet.size();i++){
        Vector tempEdgeSet=usableGraph.incidentEdgeSet((Node)routeNodeSet.elementAt(i));
        for(int j=0;j<tempEdgeSet.size();j++){
            tempEdge=(Edge)tempEdgeSet.elementAt(j);
            if(!incidentEdges.contains(tempEdge))
                incidentEdges.addElement(tempEdge);

            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(routeNodeSet.contains(n1) && routeNodeSet.contains(n2))
                incidentEdges.removeElement(tempEdge);
            if(n1==maMr.getCenter().getDummyNode() || n2==maMr.getCenter().getDummyNode())
                incidentEdges.removeElement(tempEdge);
        }
    }
    if(incidentEdges.contains(myEdge))
        incidentEdges.removeElement(myEdge);

    return incidentEdges;
}

//walker methods:
//nodeDept[node][0]: currCost
public void setCurrCost(Node node,double cost){
    nodeDept[node.getLabel()][0]=new Double(cost);
}

public double getCurrCost(Node node){
    Double tempDouble=(Double)nodeDept[node.getLabel()][0];
```



```
        return tempDouble.doubleValue();
    }

    //nodeDept[node][1]: preNode
    public void setPreNode(Node node, Node preNode){
        nodeDept[node.getLabel()][1]=preNode;
    }

    public Node getPreNode(Node node){
        return (Node)nodeDept[node.getLabel()][1];
    }

    //nodeDept[node][2]: preEdge
    public void setPreEdge(Node node, Edge preEdge){
        nodeDept[node.getLabel()][2]=preEdge;
    }

    public Edge getPreEdge(Node node){
        return (Edge)nodeDept[node.getLabel()][2];
    }

    //walkerDept[0]: myNode(Node)
    public void setMyNode(Node node){
        walkerDept[0]=node;
    }

    public Node getMyNode(){
        return (Node)walkerDept[0];
    }

    //walkerDept[1]: myEdge(Edge)
    public void setMyEdge(Edge edge){
        walkerDept[1]=edge;
    }

    public Edge getMyEdge(){
        return (Edge)walkerDept[1];
    }

    //walkerDept[2]: currNode(Node)
    public void setCurrNode(Node node){
        walkerDept[2]=node;
    }
```

```
}

public Node getCurrNode(){
    return (Node)walkerDept[2];
}

//walkerDept[3]: routeSet(Graph)
public void setRouteSet(Vector nodeSet,Vector edgeSet){
    walkerDept[3]=new Graph(nodeSet,edgeSet);
}

public Graph getRouteSet(){
    return (Graph)walkerDept[3];
}

public void addNode(Node node){
    Graph routeSet=(Graph)walkerDept[3];
    Vector nodeSet=routeSet.getNodeSet();
    if(!nodeSet.contains(node))
        nodeSet.addElement(node);
}

public void addEdge(Edge edge){
    Graph routeSet=(Graph)walkerDept[3];
    Vector edgeSet=routeSet.getEdgeSet();
    if(!edgeSet.contains(edge))
        edgeSet.addElement(edge);
}
}
```

## MAManager.java

```
package emnet.thread;

import java.util.Vector;
import emnet.graph.Node;
import emnet.graph.Graph;
import emnet.graph.Edge;
import emnet.algorithm.GraphAlgorithm;

public class MAManager extends Thread{

    int id;
    Center center;

    Node supply;
    Graph territory,usableGraph;
    Vector icp,helpers,usableGraphNodeSet,usableGraphEdgeSet,interfaceNodes;
    int demandNum,helperNum;

    //nodeDept
    //nodeDept[helper][node][0]: currMACost
    //nodeDept[helper][node][1]: preNode
    //nodeDept[helper][node][2]: preEdge
    Object[][][] nodeDept;

    //maDept
    //maDept[helper][0]: myNode(Node)
    //maDept[helper][1]: myEdge(Edge)
    //maDept[helper][2]: currNode(Node)
    //maDept[helper][3]: routeSet(Graph)
    Object[][] maDept;

    //maCost
    double minMACost;

    boolean available,maFinish;

    public MAManager(int id,Center center,Node supply){
        this.id=id;
        this.center=center;
        this.supply=supply;
    }
}
```

```
territory=center.getTerritory(supply);

if(territory.getDemandNodeSet().size()!=0){
    demandNum=territory.getDemandNodeSet().size();

    Vector territoryNodeSet=territory.getNodeSet();

    //icp init, usableGraph init
    Graph graph=center.getGraph();
    Vector graphEdgeSet=graph.getEdgeSet();

    usableGraphNodeSet=new Vector();
    usableGraphEdgeSet=new Vector();
    icp=new Vector();

    Edge tempEdge;
    Node n1,n2;
    for(int i=0;i<graphEdgeSet.size();i++){
        tempEdge=(Edge)graphEdgeSet.elementAt(i);
        n1=tempEdge.getN1();
        n2=tempEdge.getN2();

        if(!territoryNodeSet.contains(n1) && !territoryNodeSet.contains(n2)){
            if(!usableGraphEdgeSet.contains(tempEdge))
                usableGraphEdgeSet.addElement(tempEdge);
            if(!usableGraphNodeSet.contains(n1))
                usableGraphNodeSet.addElement(n1);
            if(!usableGraphNodeSet.contains(n2))
                usableGraphNodeSet.addElement(n2);
        }

        if(territoryNodeSet.contains(n1) && !territoryNodeSet.contains(n2)){
            if(!usableGraphEdgeSet.contains(tempEdge))
                usableGraphEdgeSet.addElement(tempEdge);
            if(!usableGraphNodeSet.contains(n1))
                usableGraphNodeSet.addElement(n1);
            if(!usableGraphNodeSet.contains(n2))
                usableGraphNodeSet.addElement(n2);
            if(!tempEdge.isDummyEdge()){
                if(!icp.contains(n1))
                    icp.addElement(n1);
            }
        }
    }
}
```

```

    }
}

if(!territoryNodeSet.contains(n1) && territoryNodeSet.contains(n2)){
    if(!usableGraphEdgeSet.contains(tempEdge))
        usableGraphEdgeSet.addElement(tempEdge);
    if(!usableGraphNodeSet.contains(n1))
        usableGraphNodeSet.addElement(n1);
    if(!usableGraphNodeSet.contains(n2))
        usableGraphNodeSet.addElement(n2);
    if(!tempEdge.isDummyEdge()){
        if(!icp.contains(n2))
            icp.addElement(n2);
    }
}

//while territory is not convex
if(!territory.hasEdge(tempEdge) && !usableGraphEdgeSet.contains(tempEdge)){
    usableGraphEdgeSet.addElement(tempEdge);
    if(!usableGraphNodeSet.contains(n1))
        usableGraphNodeSet.addElement(n1);
    if(!usableGraphNodeSet.contains(n2))
        usableGraphNodeSet.addElement(n2);
    if(!tempEdge.isDummyEdge()){
        if(!icp.contains(n1))
            icp.addElement(n1);
        if(!icp.contains(n2))
            icp.addElement(n2);
    }
}
}

usableGraph=new Graph(usableGraphNodeSet,usableGraphEdgeSet);
interfaceNodes=GraphAlgorithm.getInterfaceNodes(territory,graph);
helperNum=icp.size();
nodeDept=new Object[icp.size()][graph.getNodeSet().size()][3];
for(int i=0;i<icp.size();i++){
    for(int j=0;j<graph.getNodeSet().size();j++){
        nodeDept[i][j][0]=new Double(0.0);
    }
}

maDept=new Object[icp.size()][4];
minMACost=0.0;

```

```
        available=true;
        maFinish=false;
    }else{
        maFinish=true;
    }

}

public void run(){
    center.takeKey(this);

    //map init
    Vector edgeSet=center.getGraph().getEdgeSet();
    Edge tempEdge;
    for(int i=0;i<edgeSet.size();i++){
        tempEdge=(Edge)edgeSet.elementAt(i);
        if(!tempEdge.isFastEdge() && !tempEdge.isDetourEdge() && !tempEdge.isMAEdge())
            tempEdge.setNeutralEdge();
    }

    if(territory.getDemandNodeSet().size()!=0){
        helpers=new Vector();
        Node tempNode;
        Helper tempHelper;
        for(int i=0;i<icp.size();i++){
            tempNode=(Node)icp.elementAt(i);
            tempHelper=new Helper(i,tempNode,this);
            helpers.addElement(tempHelper);
            tempHelper.start();
        }

        while(!maFinish){
            //watching
        }

        //finding supply source finished: all situations simulated
        //set ma edges on the shortest ma route
        //find a subject helper
        Helper bestHelper=(Helper)helpers.elementAt(0);
        for(int j=1;j<helpers.size();j++){
            if(bestHelper.getMACost()==0.0)
                bestHelper=(Helper)helpers.elementAt(j);
        }
    }
}
```

```

    }

    //find a competitor helper
    Helper tempHelper1;
    double minMACost=bestHelper.getMACost();
    for(int j=1;j<helpers.size();j++){
        tempHelper1=(Helper)helpers.elementAt(j);
        if(tempHelper1.getMACost()!=0.0 && tempHelper1.getMACost(<minMACost){
            minMACost=tempHelper1.getMACost();
            bestHelper=tempHelper1;
        }
    }

    updateMACenter(bestHelper);

    Vector bestMAPathEdgeSet=bestHelper.getMapPath().getEdgeSet();
    if(bestMAPathEdgeSet.size(>0){
        Edge tempEdge1;
        Node n1,n2;
        Node source=bestHelper.getSource();
        Graph sourceTerritory=center.getTerritory(source);
        for(int j=0;j<bestMAPathEdgeSet.size();j++){
            tempEdge1=(Edge)bestMAPathEdgeSet.elementAt(j);
            tempEdge1.setMAEdge();

            n1=tempEdge1.getN1();
            n2=tempEdge1.getN2();
            if(territory.hasNode(n1) && !territory.hasNode(n2))
                n1.setMerge();
            if(territory.hasNode(n2) && !territory.hasNode(n1))
                n2.setMerge();
            if(sourceTerritory.hasNode(n1) && !sourceTerritory.hasNode(n2))
                n1.setAccess();
            if(sourceTerritory.hasNode(n2) && !sourceTerritory.hasNode(n1))
                n2.setAccess();

            if(j==(bestMAPathEdgeSet.size()-1)){
                if(territory.hasNode(n1) && sourceTerritory.hasNode(n1)){
                    n1.setAccess();
                    n1.setMerge();
                }
                if(territory.hasNode(n2) && sourceTerritory.hasNode(n2)){

```

```

        n2.setAccess();
        n2.setMerge();
    }
}
}
}else{
    bestHelper.getSource().setSource();
}
}else{
    //demandNum=0

    //mutualAssistantCenter
    //maCenter[supply][2]: source(Node)
    //maCenter[supply][3]: icpSet(Vector)
    //maCenter[supply][4]: maPath(Graph)
    //maCenter[supply][7]: maCost > source to supply
    center.setSource(supply,supply);
    center.setICPSet(supply,territory.getNodeSet());
    center.setMAPath(supply,territory);
    center.setMACost(supply,0.0);
}

//ma manager finish its job!
center.updateMACondition();
center.putKey(this);
}

public int getID(){
    return id;
}

public Center getCenter(){
    return center;
}

public Node getSupply(){
    return supply;
}

public Graph getGraph(){
    return center.getGraph();
}
}

```



```
public Graph getUsableGraph(){
    return usableGraph;
}

public Graph getTerritory(){
    return territory;
}

public int getDemanNum(){
    return demandNum;
}

public int getlCP(){
    return interfaceNodes.size();
}

public synchronized void updateMinMACost(double currMACost){
    if(minMACost==0.0 || currMACost<minMACost)
        minMACost=currMACost;
}

public double getMinMACost(){
    return minMACost;
}

public void updateMAFinish(){
    helperNum--;
    if(helperNum==0){
        maFinish=true;
    }
}

public synchronized void takeKey(Helper helper){
    if(!maFinish){
        while(!available){
            try{
                wait(1);
            }catch(InterruptedException e){
                //takeKey: cannot wait!
            }
        }
    }
}
```

```

        available=false;
    }else{
        available=false;
        //helper took the key, but center is finished!
    }
}

public synchronized void putKey(Helper helper){
    if(!maFinish){
        while(available){
            try{
                //helper is waiting to put...
                wait(1);
            }catch(InterruptedException e){
                //putKey: cannot wait!
            }
        }
        //helper put the key!
        available=true;
    }else{
        //helper put the key, maFinish=true!
        available=true;
    }
}

//nodeDept
//nodeDept[helper][node][0]: currMACost
public void setCurrMACost(Helper helper,Node node,double maCost){
    nodeDept[helper.getID()][node.getLabel()][0]=new Double(maCost);
}

public double getCurrMACost(Helper helper,Node node){
    Double maCost=(Double)nodeDept[helper.getID()][node.getLabel()][0];
    return maCost.doubleValue();
}

//nodeDept[helper][node][1]: preNode
public void setPreNode(Helper helper,Node node,Node preNode){
    nodeDept[helper.getID()][node.getLabel()][1]=preNode;
}

public Node getPreNode(Helper helper,Node node){

```

```
        return (Node)nodeDept[helper.getID()][node.getLabel()][1];
    }

    //nodeDept[helper][node][2]: preEdge
    public void setPreEdge(Helper helper,Node node,Edge preEdge){
        nodeDept[helper.getID()][node.getLabel()][2]=preEdge;
    }

    public Edge getPreEdge(Helper helper,Node node){
        return (Edge)nodeDept[helper.getID()][node.getLabel()][2];
    }

    //maDept
    //maDept[helper][0]: myNode(Node)
    public void setMyNode(Helper helper,Node myNode){
        maDept[helper.getID()][0]=myNode;
    }

    public Node getMyNode(Helper helper){
        return (Node)maDept[helper.getID()][0];
    }

    //maDept[helper][1]: myEdge(Edge)
    public void setMyEdge(Helper helper,Edge myEdge){
        maDept[helper.getID()][1]=myEdge;
    }

    public Edge getMyEdge(Helper helper){
        return (Edge)maDept[helper.getID()][1];
    }

    //maDept[helper][2]: currNode(Node)
    public void setCurrNode(Helper helper,Node currNode){
        maDept[helper.getID()][2]=currNode;
    }

    public Node getCurrNode(Helper helper){
        return (Node)maDept[helper.getID()][2];
    }

    //maDept[helper][3]: routeSet(Graph)
    public void setRouteSet(Helper helper,Vector routeNodeSet,Vector routeEdgeSet){
```

```
        maDept[helper.getID()][3]=new Graph(routeNodeSet,routeEdgeSet);
    }

    public Graph getRouteSet(Helper helper){
        return (Graph)maDept[helper.getID()][3];
    }

    public Vector getRouteEdgeSet(Helper helper){
        Graph routeSet=getRouteSet(helper);
        return routeSet.getEdgeSet();
    }

    public synchronized void addEdge(Helper helper,Edge edge){
        Graph routeSet=this.getRouteSet(helper);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeEdgeSet.contains(edge))
            routeEdgeSet.addElement(edge);
        this.setRouteSet(helper,routeNodeSet,routeEdgeSet);
    }

    public synchronized void addNode(Helper helper,Node node){
        Graph routeSet=this.getRouteSet(helper);
        Vector routeNodeSet=routeSet.getNodeSet();
        Vector routeEdgeSet=routeSet.getEdgeSet();
        if(!routeNodeSet.contains(node))
            routeNodeSet.addElement(node);
        this.setRouteSet(helper,routeNodeSet,routeEdgeSet);
    }

    public synchronized void updateMACenter(Helper helper){
        //mutualAssistantCenter
        //maCenter[supply][2]: source(Node)
        //maCenter[supply][3]: icpSet(Vector)
        //maCenter[supply][4]: maPath(Graph)
        //maCenter[supply][7]: maCost > source to supply

        MAManager maMr=helper.getMAManger();
        Center center=maMr.getCenter();
        Node supply=maMr.getSupply();

        center.setSource(supply,helper.getSource());
    }
```

# **MAManager.java**

```
        center.setICPSet(supply,interfaceNodes);
        center.setMAPath(supply,helper.getMAPath());
        center.setMACost(supply,helper.getMACost());
    }
}
```

## Roamer.java

```
package emnet.thread;

import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Vector;
import emnet.algorithm.GraphAlgorithm;
import emnet.graph.Graph;

public class Roamer extends Thread{
    Node supply;
    int id;
    Center center;
    boolean roamerFinish;

    Node currNode,preNode,myNode;
    Edge preEdge,myEdge;
    Graph graph,routeSet;
    Vector routeNodeSet,routeEdgeSet;

    public Roamer(int id,Node supply,Center center){
        super(""+id);
        this.id=id;
        this.supply=supply;
        this.center=center;
        roamerFinish=false;
        graph=center.getGraph();
        int nodeNum=graph.getNodeSet().size();
        int edgeNum=graph.getEdgeSet().size();

        //roamerCenter init
        //roamerCenter[roamer][0]: myNode(Node)
        //roamerCenter[roamer][1]: myEdge(Edge)
        //roamerCenter[roamer][2]: currNode(Node)
        //roamerCenter[roamer][3]: routeSet(Graph)

        Node dummyNode=this.center.getDummyNode();
        dummyNode.setDummy();
        Edge dummyEdge=new Edge(edgeNum,supply,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        this.graph.addNode(dummyNode);
```

```

        this.graph.addEdge(dummyEdge);

        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);
        this.center.setRouteSet(this,routeNodeSet,routeEdgeSet);

        this.center.setMyNode(this,dummyNode);
        this.center.setMyEdge(this,dummyEdge);
        this.center.setCurrNode(this,supply);

        //nodeCenter init
        //nodeCenter[node][0]: occupy(Boolean)
        //nodeCenter[node][1]: distance(Double)
        //nodeCenter[node][2]: preNode(Node)
        //nodeCenter[node][3]: preEdge(Edge)
        //nodeCenter[node][4]: visitorSequence(Vector)

        this.center.setDistance(supply,0.0);
        this.center.setPreNode(supply,dummyNode);
        this.center.setPreEdge(supply,dummyEdge);
        this.center.addViditor(supply,this);
    }

    public int getID(){
        return this.id;
    }

    public Node getSupply(){
        return this.supply;
    }

    public Center getCenter(){
        return this.center;
    }

    public void run(){

        roaming:
        while(!center.isFinished() && !roamerFinish){
            //take the key to have the right to run

```

```

center.takeKey(this);

if(center.isFinished()){
    center.putKey(this);
    break roaming;
}

//test currNode is visited or not
if(!center.getCurrNode(this).isVisited()){
    //currNode is not visited
    center.getCurrNode(this).visit();
    updateVisitorSequence(0,this,center.getCurrNode(this));

    //update myEdge, myNode, currNode, preNode
    //update currNode distance by myNode & myEdge
    //update routeSet (myNode, myEdge)
    //set fast edge
    center.addNode(this,center.getCurrNode(this));
    center.addEdge(this,center.getMyEdge(this));

    center.setPreNode(center.getCurrNode(this),center.getMyNode(this));
    center.setPreEdge(center.getCurrNode(this),center.getMyEdge(this));
    double distance=center.getDistance(center.getMyNode(this))+center.getMyEdge(this).getWeight();
    center.setDistance(center.getCurrNode(this),distance);
    center.getMyEdge(this).setTestEdge();
    center.setMyNode(this,center.getCurrNode(this));

    dijkstra();
}else{
    //currNode is visited
    //Comparison: change to new location to find myEdge to find currNode, and update to myNode
    Vector visitorSequence=center.getVisitorSequence(center.getCurrNode(this));
    if(!visitorSequence.contains(this)){
        //comparison
        double currDistance=center.getDistance(center.getCurrNode(this));
        double myDistance=center.getDistance(center.getMyNode(this));
        if(myDistance+center.getMyEdge(this).getWeight()<currDistance){
            //closer! subtree finding!
            //1. find out who the last roamer is
            Roamer lastRoamer=center.lastVisitor(center.getCurrNode(this));

            //2. update visitor sequence [finish condition]

```



```

        updateVisitorSequence(1,this,center.getCurrNode(this));

        //3. find out where the last roamer is now
        Node lastRoamerLocation=center.getCurrNode(lastRoamer);

        //4. remove last roamer's subtree
        Graph lastRoamerRouteSet=center.getRouteSet(lastRoamer);
        Graph
subtree=GraphAlgorithm.getSubtreeWithCertainNode(lastRoamerRouteSet,center.getCurrNode(this),center.getPreEdge(center.getCurrNode(this)));

        center.removeSubtree(lastRoamer,subtree);
        center.removeEdge(lastRoamer,center.getPreEdge(center.getCurrNode(this)));
        center.getPreEdge(center.getCurrNode(this)).setNeutralEdge();

        //5. add myEdge and the subtree to current roamer's RouteSet
        center.setPreEdge(center.getCurrNode(this),center.getMyEdge(this));
        center.setPreNode(center.getCurrNode(this),center.getMyNode(this));
        center.addSubtree(this,subtree);
        center.addEdge(this,center.getMyEdge(this));
        center.getMyEdge(this).setTestEdge();

        //6. relocate last roamer's place to preNode to prevent missing, relocate current roamer
using dijkstra
        center.setCurrNode(lastRoamer,center.getPreNode(center.getCurrNode(this)));

        //7. update the distance in the subtree
        Vector subtreeNodeSet=subtree.getNodeSet();
        Node tempNode;
        double saving;
        for(int i=0;i<subtreeNodeSet.size();i++){
            tempNode=(Node)subtreeNodeSet.elementAt(i);

            saving=center.getDistance(center.getCurrNode(this))-(center.getDistance(center.getMyNode(this))+center.getMyEdge(this).getWeight());

            center.setDistance(tempNode,center.getDistance(tempNode)-saving);
        }

        dijkstra();
    }else{
        //not closer, remove myEdge from routeEdgeSet, find new myEdge
        //update visitor sequence: downstream subtree

```

```

        updateVisitorSequence(1,this,center.getCurrNode(this));
        center.removeEdge(this,center.getMyEdge(this));

        dijkstra();
    }
    }else{
        //I visited this node before
        dijkstra();
    }
}
center.putKey(this);

try{
    sleep(1);
}catch(InterruptedException ex){
    //roamer cannot sleep
}

}
//roamer finished his job!
}

void dijkstra(){
    //find out the best incident edge
    //define new myEdge & currNode
    Graph tempRouteSet=(Graph)center.getRouteSet(this);
    Vector tempRouteNodeSet=(Vector)tempRouteSet.getNodeSet();

    Vector incidentEdgeSet=GraphAlgorithm.getIncidentEdgeSet(this,tempRouteNodeSet);
    Vector exclusiveIncidentEdgeSet=new Vector();
    for(int i=0;i<incidentEdgeSet.size();i++){
        if(!center.getRouteEdgeSet(this).contains(incidentEdgeSet.elementAt(i))){
            exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
        }
    }

    Edge tempEdge;
    Node n1,n2;
    if(exclusiveIncidentEdgeSet.size(>)1){
        tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
        center.setMyEdge(this,tempEdge);
        n1=tempEdge.getN1();
    }
}

```

```

        n2=tempEdge.getN2();
        if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
            center.setMyNode(this,n1);
            center.setCurrNode(this,n2);
        }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
            center.setMyNode(this,n2);
            center.setCurrNode(this,n1);
        }else{
            //dijkstra error 1: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
        }

        double min=center.getDistance(center.getMyNode(this))+center.getMyEdge(this).getWeight();
        for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                if(center.getDistance(n1)+tempEdge.getWeight()<min){
                    min=center.getDistance(n1)+tempEdge.getWeight();
                    center.setMyEdge(this,tempEdge);
                    center.setMyNode(this,n1);
                    center.setCurrNode(this,n2);
                }
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                if(center.getDistance(n2)+tempEdge.getWeight()<min){
                    min=center.getDistance(n2)+tempEdge.getWeight();
                    center.setMyEdge(this,tempEdge);
                    center.setMyNode(this,n2);
                    center.setCurrNode(this,n1);
                }
            }else{
                //dijkstra error 2: incident edge error!
            }
        }
    }else if(exclusiveIncidentEdgeSet.size()==1){
        tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
        center.setMyEdge(this,tempEdge);
        n1=tempEdge.getN1();
        n2=tempEdge.getN2();
        if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
            center.setMyNode(this,n1);
            center.setCurrNode(this,n2);
        }
    }
}

```

```

        }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
            center.setMyNode(this,n2);
            center.setCurrNode(this,n1);
        }else{
            //dijkstra error 3: not incident edge! check GraphAlgorithm.getIncidentEdgeSet()
        }
    }else{
        roamerFinish=true;
        //renders all nodes visited
        Vector nodeSet=graph.getNodeSet();
        Node tempNode;
        for(int i=0;i<nodeSet.size();i++){
            tempNode=(Node)nodeSet.elementAt(i);
            center.addViditor(tempNode,this);
        }
    }
}

```

```

void updateVisitorSequence(int sort,Roamer roamer,Node currNode){
    //roamer to be updated....
    //selection choice:
    //case 0: currNode is never visited
    //case 1: currNode is visited

    Vector myVisitorSequence=center.getVisitorSequence(center.getMyNode(this));
    Graph routeSet,subtree;
    Edge preEdge;
    Vector subtreeNodeSet,tempSubtreeNodeVisitorSequence;
    Node tempSubtreeNode,tempPreNode;
    Roamer lastRoamer,tempRoamer;

    switch(sort){
        case 0:
            //case 0: currNode is never visited
            center.addViditor(currNode,roamer);
            break;

        case 1:
            //case 1: currNode is visited
            //add visiotrs to downstream: subtree
            //subtree of currNode
            lastRoamer=center.lastVisitor(currNode);

```

```

        routeSet=center.getRouteSet(lastRoamer);
        preEdge=center.getPreEdge(currNode);
        tempPreNode=center.getPreNode(currNode);
        subtree=GraphAlgorithm.getSubtreeWithCertainNode(routeSet,currNode,preEdge);
        subtreeNodeSet=subtree.getNodeSet();

        for(int i=0;i<myVisitorSequence.size();i++){
            tempRoamer=(Roamer)myVisitorSequence.elementAt(i);
            for(int j=0;j<subtreeNodeSet.size();j++){
                tempSubtreeNode=(Node)subtreeNodeSet.elementAt(j);
                tempSubtreeNodeVisitorSequence=center.getVisitorSequence(tempSubtreeNode);
                if(!tempSubtreeNodeVisitorSequence.contains(tempRoamer))
                    center.addViditor(tempSubtreeNode,tempRoamer);
            }
        }
        break;
    }
    //update visitorSequence finish!
}
}

```

## Walker.java

```
package emnet.thread;

import emnet.graph.Graph;
import emnet.graph.Node;
import emnet.graph.Edge;
import java.util.Vector;

public class Walker extends Thread{
    double territoryMergeCost;

    Helper helper;
    MAManager maMr;
    Graph territory;
    int demandNum;

    boolean finish;

    public Walker(Helper helper,Node start){
        territoryMergeCost=0.0;

        this.helper=helper;
        maMr=helper.getMAManger();
        territory=maMr.getTerritory();
        demandNum=territory.getDemandNodeNum();

        int edgeNum=maMr.getCenter().getGraph().getEdgeSet().size();
        Node dummyNode=maMr.getCenter().getDummyNode();
        dummyNode.setDummy();
        Edge dummyEdge=new Edge(edgeNum,start,dummyNode,0.0);
        dummyEdge.setDummyEdge();

        maMr.getGraph().addNode(dummyNode);
        maMr.getGraph().addEdge(dummyEdge);

        //helper init, set to maMr
        Vector routeNodeSet=new Vector();
        routeNodeSet.addElement(dummyNode);
        Vector routeEdgeSet=new Vector();
        routeEdgeSet.addElement(dummyEdge);

        helper.setRouteSet(routeNodeSet,routeEdgeSet);
    }
}
```

## Walker.java

```
        helper.setMyNode(dummyNode);
        helper.setMyEdge(dummyEdge);
        helper.setCurrNode(start);

        helper.setCurrCost(helper.getMyNode(),0.0);

        helper.setPreNode(start,dummyNode);
        helper.setPreEdge(start,dummyEdge);

        finish=false;
    }

    public double getTerritoryMergeCost(){
        return territoryMergeCost;
    }

    public void run(){

        walking:
        while(!finish){
            if(helper.getCurrNode().isDemand()){

                updateFinish();

                helper.addNode(helper.getCurrNode());
                helper.addEdge(helper.getMyEdge());

                helper.setPreNode(helper.getCurrNode(),helper.getMyNode());
                helper.setPreEdge(helper.getCurrNode(),helper.getMyEdge());

                double currCost=helper.getCurrCost(helper.getMyNode())+helper.getMyEdge().getWeight();
                helper.setCurrCost(helper.getCurrNode(),currCost);

                territoryMergeCost=territoryMergeCost+currCost;

                if(!finish){
                    helper.setMyNode(helper.getCurrNode());
                    dijkstra();
                }else{
                    break walking;
                }
            }else{

```

```

        helper.addNode(helper.getCurrNode());
        helper.addEdge(helper.getMyEdge());

        helper.setPreNode(helper.getCurrNode(),helper.getMyNode());
        helper.setPreEdge(helper.getCurrNode(),helper.getMyEdge());

        double currCost=helper.getCurrCost(helper.getMyNode())+helper.getMyEdge().getWeight();
        helper.setCurrCost(helper.getCurrNode(),currCost);

        helper.setMyNode(helper.getCurrNode());
        dijkstra();
    }
}

void updateFinish(){
    demandNum--;

    if(demandNum==0)
        finish=true;
}

public boolean isFinish(){
    return finish;
}

void dijkstra(){
    Graph tempRouteSet=(Graph)helper.getRouteSet();
    Vector tempRouteNodeSet=(Vector)tempRouteSet.getNodeSet();
    Vector incidentEdgeSet=getIncidentEdgeSet(territory,tempRouteNodeSet);
    Vector exclusiveIncidentEdgeSet=new Vector();
    for(int i=0;i<incidentEdgeSet.size();i++){
        if(!helper.getRouteSet().hasEdge((Edge)incidentEdgeSet.elementAt(i))){
            exclusiveIncidentEdgeSet.addElement(incidentEdgeSet.elementAt(i));
        }
    }

    Edge tempEdge;
    Node n1,n2;
    if(exclusiveIncidentEdgeSet.size(>1){
        tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
        helper.setMyEdge(tempEdge);
    }
}

```



```

        n1=tempEdge.getN1();
        n2=tempEdge.getN2();
        if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
            helper.setMyNode(n1);
            helper.setCurrNode(n2);
        }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
            helper.setMyNode(n2);
            helper.setCurrNode(n1);
        }
        double min=helper.getCurrCost(helper.getMyNode()+helper.getMyEdge().getWeight());
        for(int i=1;i<exclusiveIncidentEdgeSet.size();i++){
            tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(i);
            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
                if(helper.getCurrCost(n1)+tempEdge.getWeight()<min){
                    min=helper.getCurrCost(n1)+tempEdge.getWeight();
                    helper.setMyEdge(tempEdge);
                    helper.setMyNode(n1);
                    helper.setCurrNode(n2);
                }
            }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
                if(helper.getCurrCost(n2)+tempEdge.getWeight()<min){
                    min=helper.getCurrCost(n2)+tempEdge.getWeight();
                    helper.setMyEdge(tempEdge);
                    helper.setMyNode(n2);
                    helper.setCurrNode(n1);
                }
            }
        }
    }
    }else if(exclusiveIncidentEdgeSet.size()==1){
        tempEdge=(Edge)exclusiveIncidentEdgeSet.elementAt(0);
        helper.setMyEdge(tempEdge);
        n1=tempEdge.getN1();
        n2=tempEdge.getN2();
        if(tempRouteNodeSet.contains(n1) && !tempRouteNodeSet.contains(n2)){
            helper.setMyNode(n1);
            helper.setCurrNode(n2);
        }else if(tempRouteNodeSet.contains(n2) && !tempRouteNodeSet.contains(n1)){
            helper.setMyNode(n2);
            helper.setCurrNode(n1);
        }
    }
}

```

```

    }else{
        finish=true;
        //renders all nodes visited
    }
}

Vector getIncidentEdgeSet(Graph usableGraph,Vector routeNodeSet){
    Edge myEdge=helper.getMyEdge();

    Vector incidentEdges=new Vector();
    Edge tempEdge;
    Node n1,n2;

    for(int i=0;i<routeNodeSet.size();i++){
        Vector tempEdgeSet=usableGraph.incidentEdgeSet((Node)routeNodeSet.elementAt(i));
        for(int j=0;j<tempEdgeSet.size();j++){
            tempEdge=(Edge)tempEdgeSet.elementAt(j);
            if(!incidentEdges.contains(tempEdge))
                incidentEdges.addElement(tempEdge);

            n1=tempEdge.getN1();
            n2=tempEdge.getN2();
            if(routeNodeSet.contains(n1) && routeNodeSet.contains(n2))
                incidentEdges.removeElement(tempEdge);
            if(n1==maMr.getCenter().getDummyNode() || n2==maMr.getCenter().getDummyNode())
                incidentEdges.removeElement(tempEdge);
        }
    }
    if(incidentEdges.contains(myEdge))
        incidentEdges.removeElement(myEdge);

    return incidentEdges;
}
}

```

## App.java

```
package emnet;

import java.awt.Toolkit;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Dimension;

public class App {
    boolean packFrame = false;

    /**
     * Construct and show the application.
     */
    public App() {
        Frame frame = new Frame();
        // Validate frames that have preset sizes
        // Pack frames that have useful preferred size info, e.g. from their layout
        if (packFrame) {
            frame.pack();
        }
        else {
            frame.validate();
        }

        // Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation( (screenSize.width - frameSize.width) / 2,
                           (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }

    /**
     * Application entry point.
     */
}
```

## App.java

```
* @param args String[]
*/
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            try {
                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            }
            catch (Exception exception) {
                exception.printStackTrace();
            }

            //read in node & edge data
            //add to table
            //picture the map
            //frame.add table & map
            //set supply & demand nodes
            //response in map
            //after press "start" button, new ant & antCommunicationCenter
            //show fast routes, detour routes, mutual assistant routes
            //show evaluation indices
            //input expert acceptable time
            //illustrate radar digram and standardized evaluation value

            new App();
        }
    });
}
```

## Frame\_AboutBox.java

```
package emnet;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Frame_AboutBox
    extends JDialog implements ActionListener {
    JPanel panel1 = new JPanel();
    JPanel panel2 = new JPanel();
    JPanel insetsPanel1 = new JPanel();
    JPanel insetsPanel2 = new JPanel();
    JPanel insetsPanel3 = new JPanel();
    JButton button1 = new JButton();
    JLabel imageLabel = new JLabel();
    JLabel label1 = new JLabel();
    JLabel label2 = new JLabel();
    JLabel label3 = new JLabel();
    JLabel label4 = new JLabel();
    ImageIcon image1 = new ImageIcon();
    BorderLayout borderLayout1 = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    FlowLayout flowLayout1 = new FlowLayout();
    GridLayout gridLayout1 = new GridLayout();
    String product = "Earthquake Mitigation Network Design";
    String version = "version 1.0, percy.itt.nctu.tw";
    String copyright = "Copyright (c) 2006";
    String comments = "Decision Making Tool for Network Design";

    public Frame_AboutBox(Frame parent) {
        super(parent);
        try {
            setDefaultCloseOperation(DISPOSE_ON_CLOSE);
            jbInit();
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    public Frame_AboutBox() {
```

## Frame\_AboutBox.java

```
this(null);
}

/**
 * Component initialization.
 *
 * @throws java.lang.Exception
 */
private void jbInit() throws Exception {
    image1 = new ImageIcon(emnet.Frame.class.getResource("about.png"));
    imageLabel.setIcon(image1);
    setTitle("About");
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText(version);
    label3.setText(copyright);
    label4.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
    button1.setText("OK");
    button1.addActionListener(this);
    insetsPanel2.add(imageLabel, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
    setResizable(true);
}

/**
```

# Frame\_AboutBox.java

```
* Close the dialog on a button event.  
*  
* @param actionEvent(ActionEvent)  
*/  
public void actionPerformed(ActionEvent actionEvent) {  
    if (actionEvent.getSource() == button1) {  
        dispose();  
    }  
}  
}
```

## Frame\_TermBox.java

```
package emnet;

import javax.swing.JDialog;
import java.awt.GridLayout;
import javax.swing.JLabel;
import java.awt.Dimension;

public class Frame_TermBox extends JDialog{

    JLabel
        label_eva=new JLabel(" E: Overall Evaluation"),
        label_ld=new JLabel(" LD: Longest Detour Cost"),
        label_amac=new JLabel(" AMAC: Average Mutal Assistance Cost"),
        label_nc=new JLabel(" NC: Network Cost"),
        label_atc=new JLabel(" ATC: Average Travel Cost"),
        label_mtc=new JLabel(" MTC: Maximum Travel Cost");

    public Frame_TermBox(Frame parent){
        super(parent);
        try {
            setDefaultCloseOperation(DISPOSE_ON_CLOSE);
            jbInit();
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    void jbInit(){
        Dimension d=new Dimension(250,25);
        this.setTitle("Terminology");
        this.getContentPane().setLayout(new GridLayout(6,1,5,5));
        label_eva.setPreferredSize(d);
        label_ld.setPreferredSize(d);
        label_amac.setPreferredSize(d);
        label_nc.setPreferredSize(d);
        label_atc.setPreferredSize(d);
        label_mtc.setPreferredSize(d);

        this.getContentPane().add(label_atc);
        this.getContentPane().add(label_mtc);
    }
}
```



```
        this.getContentPane().add(label_ld);
        this.getContentPane().add(label_amac);
        this.getContentPane().add(label_nc);
        this.getContentPane().add(label_eva);

        this.setResizable(false);
    }
}
```

## Frame.java

```
package emnet;

import java.awt.*;
import java.awt.event.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JLabel;

import emnet.gui.Map;
import emnet.graph.Graph;
import java.util.Vector;
import java.io.IOException;
import emnet.io.IOGraph;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.SwingConstants;
import javax.swing.JTabbedPane;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JSplitPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.JTable;
import javax.swing.JScrollPane;
import emnet.graph.Node;
import emnet.graph.Edge;
import emnet.thread.Center;
import java.text.DecimalFormat;

public class Frame
    extends JFrame {
    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
    JMenuBar jMenuBar1 = new JMenuBar();
    JMenu jMenuFile = new JMenu();
    JMenuItem jMenuItemFileExit = new JMenuItem();
    JMenu jMenuHelp = new JMenu();
    JMenuItem jMenuItemHelpAbout = new JMenuItem();
    JMenuItem jMenuItemHelpTerm=new JMenuItem();
```

```

JPanel panel_gridbag=new JPanel(new GridLayout(4,1,2,2));
JPanel panel_setting=new JPanel(new BorderLayout());

JLabel label_dir=new JLabel("common dir: ");
JTextField txt_dir=new JTextField();
JLabel label_node=new JLabel("node file: ");
JTextField txt_node=new JTextField();
JLabel label_edge=new JLabel("edge file: ");
JTextField txt_edge=new JTextField();
JLabel label_dataIn=new JLabel("2econ?");

DefaultTableModel nodeTableModel,edgeTableModel;
JTable
    nodeTable=new JTable(nodeTableModel),
    edgeTable=new JTable(edgeTableModel);

JLabel seperator=new JLabel("[ no graph ]",SwingConstants.CENTER);

JRadioButton radio_supply=new JRadioButton("supply");
JRadioButton radio_demand=new JRadioButton("demand");
JRadioButton radio_neutral=new JRadioButton("neutral");
ButtonGroup radioGroup=new ButtonGroup();
JButton button_run=new JButton("run");

JTextField
    txt_atc_low=new JTextField(),txt_mtc_low=new JTextField(),
    txt_ld_low=new JTextField(),txt_amac_low=new JTextField(),
    txt_nc_low=new JTextField(),

    txt_atc_up=new JTextField(),txt_mtc_up=new JTextField(),
    txt_ld_up=new JTextField(),txt_amac_up=new JTextField(),
    txt_nc_up=new JTextField();

JLabel
    atc_output=new JLabel("0.0      "),mtc_output=new JLabel("0.0      "),
    ld_output=new JLabel("0.0      "),amac_output=new JLabel("0.0      "),
    nc_output=new JLabel("0.0      "),e_output=new JLabel("[ pls fill \"expert\" tab ] ");

JLabel save_dir=new JLabel();

Map map=new Map();

```

## Frame.java

```
Graph graph;
Center center;
DecimalFormat myFormatter=new DecimalFormat("###,###.##");

public Frame() {
    try {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        jblInit();
    }
    catch (Exception exception) {
        exception.printStackTrace();
    }
}

//Component initialization
private void jblInit() throws Exception {
    contentPane = (JPanel) getContentPane();
    contentPane.setLayout(borderLayout1);
    setSize(new Dimension(600, 700));
    setTitle(":: EMNet 2006 :: ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new Frame_jMenuFileExit_ActionAdapter(this));
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new Frame_jMenuHelpAbout_ActionAdapter(this));
    jMenuHelpTerm.setText("Term");
    jMenuHelpTerm.addActionListener(new Frame_jMenuHelpTerm_ActionAdapter(this));
    jMenuBar1.add(jMenuFile);
    jMenuFile.add(jMenuFileExit);
    jMenuBar1.add(jMenuHelp);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuHelp.add(jMenuHelpTerm);
    setJMenuBar(jMenuBar1);

    Dimension big=new Dimension(146,20),big2=new Dimension(185,20),med=new
    Dimension(100,20),small=new Dimension(65,20),xs=new Dimension(30,20);

    JPanel panel_bottom=new JPanel(new BorderLayout());
    JPanel panel_left=new JPanel(new BorderLayout());

    panel_gridbag.setPreferredSize(new Dimension(275,120));
```

```

label_dir.setHorizontalAlignment(SwingConstants.RIGHT);
label_dir.setPreferredSize(new Dimension(100, 20));
txt_dir.setPreferredSize(new Dimension(200, 20));
txt_dir.setText("/Users/percyhou/Desktop/graphFiles");
JPanel gridBag1=new JPanel(new GridBagLayout());
gridBag1.add(label_dir);
gridBag1.add(txt_dir);

label_node.setHorizontalAlignment(SwingConstants.RIGHT);
label_node.setPreferredSize(new Dimension(100, 20));
txt_node.setPreferredSize(new Dimension(200, 20));
txt_node.setText("node_grid.txt");
JPanel gridBag2=new JPanel(new GridBagLayout());
gridBag2.add(label_node);
gridBag2.add(txt_node);

label_edge.setHorizontalAlignment(SwingConstants.RIGHT);
label_edge.setPreferredSize(new Dimension(100, 20));
txt_edge.setPreferredSize(new Dimension(200, 20));
txt_edge.setText("edge_grid.txt");
JPanel gridBag3=new JPanel(new GridBagLayout());
gridBag3.add(label_edge);
gridBag3.add(txt_edge);

label_dataIn.setHorizontalAlignment(SwingConstants.RIGHT);
label_dataIn.setPreferredSize(new Dimension(100, 20));
JButton button_import=new JButton("import");
button_import.addActionListener(new Frame_button_import_ActionAdapter(this));

JPanel panel_import=new JPanel(new BorderLayout());
panel_import.add(label_dataIn,BorderLayout.WEST);
panel_import.add(button_import,BorderLayout.EAST);

panel_gridbag.add(gridBag1);
panel_gridbag.add(gridBag2);
panel_gridbag.add(gridBag3);
panel_gridbag.add(panel_import);

panel_left.add(panel_gridbag,BorderLayout.NORTH);

JTabbedPane jtabbedPane=new JTabbedPane();

```

## Frame.java

```
jtabbedPane.setPreferredSize(new Dimension(300,300));
JScrollPane tab_node=new JScrollPane(nodeTable);
JScrollPane tab_edge=new JScrollPane(edgeTable);
jtabbedPane.add(tab_node,"node");
jtabbedPane.add(tab_edge,"edge");

panel_left.add(jtabbedPane,BorderLayout.CENTER);
panel_bottom.add(panel_left,BorderLayout.WEST);

JPanel panel_right=new JPanel(new BorderLayout());

//radio buttons: supply, demand, neutral
panel_setting.setPreferredSize(new Dimension(300,120));

radio_supply.addActionListener(new Frame_radio_ActionAdapter(this));
radio_demand.addActionListener(new Frame_radio_ActionAdapter(this));
radio_neutral.addActionListener(new Frame_radio_ActionAdapter(this));
button_run.addActionListener(new Frame_button_run_ActionAdapter(this));

JPanel panel_radio=new JPanel(new GridBagLayout());
radioGroup.add(radio_supply);
radioGroup.add(radio_demand);
radioGroup.add(radio_neutral);
panel_radio.add(radio_supply);
panel_radio.add(radio_demand);
panel_radio.add(radio_neutral);
panel_radio.add(button_run);

panel_setting.add(panel_radio,BorderLayout.NORTH);

//expert bounds
JTabbedPane jtabbedPane_right=new JTabbedPane();
jtabbedPane_right.setPreferredSize(new Dimension(300,200));
JPanel tab_index=new JPanel(new BorderLayout());
JPanel panel_index=new JPanel(new GridLayout(9,1,2,2));
JPanel tab_result=new JPanel(new GridLayout(9,1,2,2));
tab_index.add(panel_index,BorderLayout.NORTH);
jtabbedPane_right.add(tab_result,"result");
jtabbedPane_right.add(tab_index,"expert");

JPanel panel_expert0=new JPanel(new BorderLayout());
JPanel panel_expert1=new JPanel(new GridBagLayout());
```

## Frame.java

```
JPanel panel_expert2=new JPanel(new GridBagLayout());
JPanel panel_expert3=new JPanel(new GridBagLayout());
JPanel panel_expert4=new JPanel(new GridBagLayout());
JPanel panel_expert5=new JPanel(new GridBagLayout());
JPanel panel_expert6=new JPanel(new BorderLayout());

JLabel
    label_mtc=new JLabel(" MTC: ",SwingConstants.RIGHT),
    label_atc=new JLabel(" ATC: ",SwingConstants.RIGHT),
    label_ld=new JLabel(" LD: ",SwingConstants.RIGHT),
    label_amac=new JLabel(" AMAC: ",SwingConstants.RIGHT),
    label_nc=new JLabel(" NC: ",SwingConstants.RIGHT);

JLabel
    label_atc_unit=new JLabel(" m",SwingConstants.LEFT),
    label_mtc_unit=new JLabel(" m",SwingConstants.LEFT),
    label_ld_unit=new JLabel(" m",SwingConstants.LEFT),
    label_amac_unit=new JLabel(" m",SwingConstants.LEFT),
    label_nc_unit=new JLabel(" m",SwingConstants.LEFT);

label_atc.setPreferredSize(small);
label_mtc.setPreferredSize(small);
label_ld.setPreferredSize(small);
label_amac.setPreferredSize(small);
label_nc.setPreferredSize(small);

txt_atc_low.setPreferredSize(small);
txt_mtc_low.setPreferredSize(small);
txt_ld_low.setPreferredSize(small);
txt_amac_low.setPreferredSize(small);
txt_nc_low.setPreferredSize(small);

txt_atc_up.setPreferredSize(small);
txt_mtc_up.setPreferredSize(small);
txt_ld_up.setPreferredSize(small);
txt_amac_up.setPreferredSize(small);
txt_nc_up.setPreferredSize(small);

label_atc_unit.setPreferredSize(xs);
label_mtc_unit.setPreferredSize(xs);
label_ld_unit.setPreferredSize(xs);
label_amac_unit.setPreferredSize(xs);
```

## Frame.java

```
label_nc_unit.setPreferredSize(xs);

JButton button_set=new JButton("set");
button_set.addActionListener(new Frame_button_set_ActionAdapter(this));

panel_expert0.add(new JLabel(" acceptable range:"),BorderLayout.WEST);

panel_expert1.add(label_atc);
panel_expert1.add(txt_atc_low);
panel_expert1.add(new JLabel(" ~ "));
panel_expert1.add(txt_atc_up);

panel_expert2.add(label_mtc);
panel_expert2.add(txt_mtc_low);
panel_expert2.add(new JLabel(" ~ "));
panel_expert2.add(txt_mtc_up);

panel_expert3.add(label_ld);
panel_expert3.add(txt_ld_low);
panel_expert3.add(new JLabel(" ~ "));
panel_expert3.add(txt_ld_up);

panel_expert4.add(label_amac);
panel_expert4.add(txt_amac_low);
panel_expert4.add(new JLabel(" ~ "));
panel_expert4.add(txt_amac_up);

panel_expert5.add(label_nc);
panel_expert5.add(txt_nc_low);
panel_expert5.add(new JLabel(" ~ "));
panel_expert5.add(txt_nc_up);

panel_expert6.add(button_set,BorderLayout.EAST);

panel_index.add(panel_expert0);
panel_index.add(panel_expert1);
panel_index.add(panel_expert2);
panel_index.add(panel_expert3);
panel_index.add(panel_expert4);
panel_index.add(panel_expert5);
panel_index.add(panel_expert6);
```



## Frame.java

```
JTextField txtField=new JTextField("[indices]",15);
txtField.setBackground(Color.WHITE);
txtField.setEditable(false);

JPanel panel_output0=new JPanel(new GridBagLayout());
JPanel panel_output3=new JPanel(new GridBagLayout());
//   JPanel panel_output29=new JPanel(new GridBagLayout());
JPanel panel_output4=new JPanel(new GridBagLayout());
//   JPanel panel_output49=new JPanel(new GridBagLayout());
JPanel panel_output5=new JPanel(new GridBagLayout());
JPanel panel_output1=new JPanel(new GridBagLayout());
JPanel panel_output2=new JPanel(new GridBagLayout());
JPanel panel_output6=new JPanel(new BorderLayout());

JLabel
    label_atc1=new JLabel(" ATC: ",SwingConstants.RIGHT),
    label_mtc1=new JLabel(" MTC: ",SwingConstants.RIGHT),
    label_ld1=new JLabel(" LD: ",SwingConstants.RIGHT),
    label_amac1=new JLabel(" AMAC: ",SwingConstants.RIGHT),
    label_nc1=new JLabel(" NC: ",SwingConstants.RIGHT),
    label_e=new JLabel(" E: ",SwingConstants.RIGHT);

JLabel
    label_atc_unit1=new JLabel(" ",SwingConstants.LEFT),
    label_mtc_unit1=new JLabel(" ",SwingConstants.LEFT),
    label_ld_unit1=new JLabel(" ",SwingConstants.LEFT),
    label_amac_unit1=new JLabel(" ",SwingConstants.LEFT),
    label_nc_unit1=new JLabel(" ",SwingConstants.LEFT),
    label_e_unit1=new JLabel();

label_atc1.setPreferredSize(small);
label_mtc1.setPreferredSize(small);
label_ld1.setPreferredSize(small);
label_amac1.setPreferredSize(small);
label_nc1.setPreferredSize(small);
label_e.setPreferredSize(small);

atc_output.setPreferredSize(big);
mtc_output.setPreferredSize(big);
ld_output.setPreferredSize(big);
amac_output.setPreferredSize(big);
nc_output.setPreferredSize(big);
```

```
e_output.setPreferredSize(big);

atc_output.setHorizontalAlignment(SwingConstants.RIGHT);
mtc_output.setHorizontalAlignment(SwingConstants.RIGHT);
ld_output.setHorizontalAlignment(SwingConstants.RIGHT);
amac_output.setHorizontalAlignment(SwingConstants.RIGHT);
nc_output.setHorizontalAlignment(SwingConstants.RIGHT);
e_output.setHorizontalAlignment(SwingConstants.RIGHT);

label_atc_unit1.setPreferredSize(xs);
label_mtc_unit1.setPreferredSize(xs);
label_ld_unit1.setPreferredSize(xs);
label_amac_unit1.setPreferredSize(xs);
label_nc_unit1.setPreferredSize(xs);
label_e_unit1.setPreferredSize(xs);

panel_output0.add(label_e);
panel_output0.add(e_output);
panel_output0.add(label_e_unit1);

panel_output1.add(label_atc1);
panel_output1.add(atc_output);
panel_output1.add(label_atc_unit1);

panel_output2.add(label_mtc1);
panel_output2.add(mtc_output);
panel_output2.add(label_mtc_unit1);

panel_output3.add(label_ld1);
panel_output3.add(ld_output);
panel_output3.add(label_ld_unit1);

panel_output4.add(label_amac1);
panel_output4.add(amac_output);
panel_output4.add(label_amac_unit1);

panel_output5.add(label_nc1);
panel_output5.add(nc_output);
panel_output5.add(label_nc_unit1);

save_dir.setPreferredSize(new Dimension(200,20));
JButton button_save=new JButton("save");
```

## Frame.java

```
//      button_save.addActionListener(this);

panel_output6.add(save_dir,BorderLayout.WEST);
panel_output6.add(button_save,BorderLayout.EAST);

tab_result.add(panel_output1);
tab_result.add(panel_output2);
tab_result.add(panel_output3);
tab_result.add(panel_output4);
tab_result.add(panel_output5);
tab_result.add(panel_output0);
tab_result.add(panel_output6);

panel_setting.add(jtabbedPane_right,BorderLayout.CENTER);
panel_right.add(panel_setting,BorderLayout.EAST);
panel_bottom.add(panel_right,BorderLayout.CENTER);
panel_bottom.add(seperator,BorderLayout.NORTH);

map.sendFrame(this);

JSplitPane sp=new JSplitPane(JSplitPane.VERTICAL_SPLIT,map,panel_bottom);
sp.setDividerSize(8);
sp.setDividerLocation(270);
sp.setResizeWeight(0.5);
sp.setContinuousLayout(true);
sp.setOneTouchExpandable(true);
contentPane.add(sp,BorderLayout.CENTER);
}

public void setATC(double atc){
    atc_output.setText(myFormatter.format(atc)+"      ");
    txt_atc_low.setText(myFormatter.format(atc*0.8));
    txt_atc_up.setText(myFormatter.format(atc*1.2));
}

public void setMTC(double mtc){
    mtc_output.setText(myFormatter.format(mtc)+"      ");
    txt_mtc_low.setText(myFormatter.format(mtc*0.8));
    txt_mtc_up.setText(myFormatter.format(mtc*1.2));
}

public void setLD(double ld){
```

## Frame.java

```
        Id_output.setText(myFormatter.format(Id)+"      ");
        txt_Id_low.setText(myFormatter.format(Id*0.8));
        txt_Id_up.setText(myFormatter.format(Id*1.2));
    }

    public void setAMAC(double amac){
        amac_output.setText(myFormatter.format(amac)+"      ");
        txt_amac_low.setText(myFormatter.format(amac*0.8));
        txt_amac_up.setText(myFormatter.format(amac*1.2));
    }

    public void setNC(double nc){
        nc_output.setText(myFormatter.format(nc)+"      ");
        txt_nc_low.setText(myFormatter.format(nc*0.8));
        txt_nc_up.setText(myFormatter.format(nc*1.2));
    }

    public void setE(double e){
        e_output.setText(myFormatter.format(e)+"      ");
    }

    void jMenuFileExit_actionPerformed(ActionEvent actionEvent) {
        System.exit(0);
    }

    void jMenuHelpAbout_actionPerformed(ActionEvent actionEvent) {
        Frame_AboutBox dlg = new Frame_AboutBox(this);
        Dimension dlgSize = dlg.getPreferredSize();
        Dimension frmSize = getSize();
        Point loc = getLocation();
        dlg.setLocation( (frmSize.width - dlgSize.width) / 2 + loc.x,
                        (frmSize.height - dlgSize.height) / 2 + loc.y);
        dlg.setModal(true);
        dlg.pack();
        dlg.show();
    }

    void jMenuHelpTerm_actionPerformed(ActionEvent actionEvent) {
        Frame_TermBox dlg = new Frame_TermBox(this);
        Dimension dlgSize = dlg.getPreferredSize();
        Dimension frmSize = getSize();
        Point loc = getLocation();
```

## Frame.java

```
        dlg.setLocation( (frmSize.width - dlgSize.width) / 2 + loc.x,
                        (frmSize.height - dlgSize.height) / 2 + loc.y);
        dlg.setModal(true);
        dlg.pack();
        dlg.show();
    }

    void button_import_actionPerformed(ActionEvent actionEvent){

        setATC(0.0);
        setMTC(0.0);
        setLD(0.0);
        setAMAC(0.0);
        setNC(0.0);
        setE(0.0);

        //read in files: node.txt & edge.txt, assign to field argument
        try {
            IOGraph ioGraph=new IOGraph(txt_dir.getText(),txt_node.getText(),txt_edge.getText());
            this.graph=ioGraph.getGraph();
            Vector nodeSet=graph.getNodeSet();
            Vector edgeSet=graph.getEdgeSet();

            Node tempNode;
            boolean econ=true;
            for(int i=0;i<nodeSet.size();i++){
                tempNode=(Node)nodeSet.elementAt(i);
                if(graph.incidentEdgeSet(tempNode).size()<2){
                    econ=false;
                }
            }

            if(econ){
                label_dataIn.setText("2econ!");
            }else{
                label_dataIn.setText("not 2econ!");
            }
        }
        catch (IOException ex) {
            System.out.println("Data Read-In Problem!");
            ex.printStackTrace();
        }
    }
}
```

```

        save_dir.setText("dir: "+txt_dir.getText());

        map.setGraph(graph);
        this.seperator.setText("[ graph components: "+graph.getNodeSet().size()+" nodes,
"+graph.getEdgeSet().size()+" edges ]");

        this.fleshTable(graph);
        this.repaint();
    }

    void radio_actionPerformed(ActionEvent actionEvent){
        map.nodeSetting(radio_supply.isSelected(),radio_demand.isSelected(),radio_neutral.isSelected());
        this.repaint();
    }

    void button_run_actionPerformed(ActionEvent actionEvent){
        center=new Center(graph,this);
        center.start();

        this.repaint();
    }

    void button_set_actionPerformed(ActionEvent actionEvent){
        double e=1.0;

        double atc=new Double(atc_output.getText()).doubleValue();
        double low_atc=new Double(txt_atc_low.getText()).doubleValue();
        double up_atc=new Double(txt_atc_up.getText()).doubleValue();
        double std_atc=standardN(atc,low_atc,up_atc);

        double mtc=new Double(mtc_output.getText()).doubleValue();
        double low_mtc=new Double(txt_mtc_low.getText()).doubleValue();
        double up_mtc=new Double(txt_mtc_up.getText()).doubleValue();
        double std_mtc=standardN(mtc,low_mtc,up_mtc);

        double amac=new Double(amac_output.getText()).doubleValue();
        double low_amac=new Double(txt_amac_low.getText()).doubleValue();
        double up_amac=new Double(txt_amac_up.getText()).doubleValue();
        double std_amac=standardN(amac,low_amac,up_amac);

        double ld=new Double(ld_output.getText()).doubleValue();

```

## Frame.java

```
double low_Id=new Double(txt_Id_low.getText()).doubleValue();
double up_Id=new Double(txt_Id_up.getText()).doubleValue();
double std_Id=standardN(l_d,low_Id,up_Id);

double nc=new Double(nc_output.getText()).doubleValue();
double low_nc=new Double(txt_nc_low.getText()).doubleValue();
double up_nc=new Double(txt_nc_up.getText()).doubleValue();
double std_nc=standardP(nc,low_nc,up_nc);

e=(1.0/3.0)*((std_atc + std_mtc)/2.0 + (std_Id + std_amac)/2.0 + std_nc);

this.setE(e);
}

//positive
double standardP(double x,double low,double up){
    if(up<=x){
        return 1.0;
    }else if(low<=x && x<up){
        return (x-low)/(up-low);
    }else{
        return 0.0;
    }
}

//negative
double standardN(double x,double low,double up){
    if(x<low){
        return 1.0;
    }else if(low<=x && x<up){
        return (low-x)/(low-up);
    }else{
        return 0.0;
    }
}

public void setSeperator(Graph graph){

    int nodeNum=0,edgeNum=0;
    Vector nodeSet=graph.getNodeSet();
    Vector edgeSet=graph.getEdgeSet();
```

## Frame.java

```
Node tempNode;
for(int i=0;i<nodeSet.size();i++){
    tempNode=(Node)nodeSet.elementAt(i);
    if(!tempNode.isDummy())
        nodeNum++;
}

Edge tempEdge;
for(int i=0;i<edgeSet.size();i++){
    tempEdge=(Edge)edgeSet.elementAt(i);
    if(!tempEdge.isDummyEdge())
        edgeNum++;
}

seperator.setText("[ graph components: "+nodeNum+" nodes, "+edgeNum+" edges | supply
"+graph.getSupplyNodeNum()+" , demand "+graph.getDemandNodeNum()+" ]");
this.repaint();
}

public Map getMap(){
    return this.map;
}

void fleshTable(Graph graph){
    Vector nodeSet=graph.getNodeSet(),edgeSet=graph.getEdgeSet();
    Object nodeData[]=new Object[3];
    DefaultTableModel nodeTableModel=new DefaultTableModel();
    nodeTableModel.addColumn("label");
    nodeTableModel.addColumn("x");
    nodeTableModel.addColumn("y");
    for(int i=0;i<nodeSet.size();i++){
        Node node=(Node)nodeSet.elementAt(i);
        nodeData[0]=""+node.getLabel();
        nodeData[1]=""+node.getX();
        nodeData[2]=""+node.getY();
        nodeTableModel.addRow(nodeData);
    }
    this.nodeTable.setModel(nodeTableModel);

    Object edgeData[]=new Object[3];
    DefaultTableModel edgeTableModel=new DefaultTableModel();
    edgeTableModel.addColumn("label");
```



## Frame.java

```
        edgeTableModel.addColumn("n1");
        edgeTableModel.addColumn("n2");
        for(int i=0;i<edgeSet.size();i++){
            Edge edge=(Edge)edgeSet.elementAt(i);
            edgeData[0]="" + edge.getLabel();
            edgeData[1]="" + edge.getN1().getLabel();
            edgeData[2]="" + edge.getN2().getLabel();
            edgeTableModel.addRow(edgeData);
        }
        this.edgeTable.setModel(edgeTableModel);
    }

}

class Frame_jMenuFileExit_ActionAdapter
    implements ActionListener {
    Frame adaptee;

    Frame_jMenuFileExit_ActionAdapter(Frame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuFileExit_actionPerformed(actionEvent);
    }
}

class Frame_jMenuHelpAbout_ActionAdapter
    implements ActionListener {
    Frame adaptee;

    Frame_jMenuHelpAbout_ActionAdapter(Frame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuHelpAbout_actionPerformed(actionEvent);
    }
}

class Frame_jMenuHelpTerm_ActionAdapter
```

## Frame.java

```
    implements ActionListener {
    Frame adaptee;

    Frame_jMenuHelpTerm_ActionAdapter(Frame adaptee) {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent) {
        adaptee.jMenuHelpTerm_actionPerformed(actionEvent);
    }
}

class Frame_button_import_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_button_import_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent){
        adaptee.button_import_actionPerformed(actionEvent);
    }
}

class Frame_radio_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_radio_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent){
        adaptee.radio_actionPerformed(actionEvent);
    }
}

class Frame_button_run_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_button_run_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }
}
```

## Frame.java

```
    public void actionPerformed(ActionEvent actionEvent){
        adaptee.button_run_actionPerformed(actionEvent);
    }
}

class Frame_button_set_ActionAdapter implements ActionListener{
    Frame adaptee;
    Frame_button_set_ActionAdapter(Frame adaptee){
        this.adaptee=adaptee;
    }

    public void actionPerformed(ActionEvent actionEvent){
        adaptee.button_set_actionPerformed(actionEvent);
    }
}
```



## 個人簡歷

侯鵬曦

交通大學交通運輸研究所 博士 (90.09~95.12)  
交通大學交通運輸研究所 碩士 (88.09~90.06)  
台灣大學土木工程學系 學士 (84.09~88.06)

### A. 期刊論文

1. 侯鵬曦、徐淵靜，「Survivable network design model for earthquake disaster」，中國土木水利工程學刊（已接受）。
2. Hsu, Y.C. and Hou, P.H. "Finding dominant links of emergency network with respect to earthquake disaster", Journal of the Eastern Asia Society for Transportation Studies, 6, pp. 77-90, 2005.
3. 徐淵靜、侯鵬曦，「高雄市防災路網研擬之研究—地理資訊系統之應用」，中華道路，第四十二卷，第二三四期，頁10-20，民國92年。

### B. 研討會論文

4. Hou, P.H. "Network planning for earthquake disaster in Kaohsiung City", 2006 International Conference on Deep Ocean Water and Industrial Development, Kaohsiung, Taiwan, 2006.
5. Hou, P.H., Lien, Barry C.S. and Chen, Johnny J.Y. "ITS-based road network application: Finding shortest time path under traffic signal system", The 13th World Congress & Exhibition on Intelligent Transport Systems and Services, London, England, 2006.
6. Lien, Barry C.S., Chen, Johnny J.Y. and Hou, P.H. "Advanced timetable query system in Taiwan railway transportation", The 13th World Congress & Exhibition on Intelligent Transport Systems and Services, London, England, 2006.
7. Lien, Barry C.S., Chen, Johnny J.Y. and Hou, P.H. "Implementation of dynamic navigation system with real-time traffic events using digital audio broadcasting (DAB) as the communication system", The 13th World Congress & Exhibition on Intelligent Transport Systems and Services, London, England, 2006.
8. 侯鵬曦，「以虛擬身分構築線上購物之安全環境」，2006電子商務與數位生活研討會，台北大學三峽校區，民國95年。
9. 侯鵬曦，「數位生活之險境—數位落差形成原因之初探」，2006電子商務與數位生活研討會，台北大學三峽校區，民國95年。
10. Hou, H.S. and Hou, P.H. "Establishment of Taiwan Hub Port", Proceedings of the 9th HKSTS Conference, Hong Kong, China, 2004.
11. Hou, H.S. and Hou, P.H. "Integrated Development Project of Ports in Taiwan", Proceedings of

the 9th HKSTS Conference, Hong Kong, China, 2004.

12. 徐淵靜、侯鵬曦，「路網配置之防震災品質評估」，第十九屆中華民國運輸研討會，台南長榮大學，民國93年。
13. Hsu, Y.C. and Hou, P.H. "Rescue network design with respect to earthquake", Proceedings of International Symposium on City Planning 2004, Sapporo, Japan, 2004.
14. Hsu, Y.C., Hsia, L.M., Hsu, M.C. and Hou, P.H. "Urban location characteristics concerned by enterprises: A case in Taipei", Proceedings of International Symposium on City Planning 2004, Sapporo, Japan, 2004.
15. 徐淵靜、侯鵬曦，「以消費者導向構建電子商店之付費系統」，2004海峽兩岸智慧型運輸系統論文研討會論文集，中國哈爾濱，民國93年。
16. Hou, H.S. and Hou, P.H. "Integrated Planning of Kaohsiung Port", Proceedings of PACON 2004, Hawaii, USA, 2004.
17. 徐淵靜、侯鵬曦，「防災路網模式構建」，第十八屆中華民國運輸研討會論文集，新竹交通大學，民國92年。
18. Hou, H.S. and Hou, P.H. "Environmental impact of redevelopment of port of Keelung", Proceedings of PACON 2003, Kaohsiung, Taiwan, 2003.
19. Hou, H.S. and Hou, P.H. "Transportation policy: Case study of Kaohsiung city, Proceedings of the 5th EAST, Fukuoka, Japan, 2003.
20. Hou, H.S. and Hou, P.H. "Offshore zone reclamation: Kaohsiung South Star Plan, Proceedings of PACON 2002, Chiba, Japan, 2002.

### C. 研究報告及其他

21. 高齡社會的來臨：為2025年的台灣社會規劃之整合研究「高齡社會之交通與運輸」，國科會專題研究，民國95年。
22. 市區道路防災服務水準之評估II，國科會專題研究，民國94年。
23. 防災路網服務水準之評估模式I，國科會專題研究，民國93年。
24. 都市防災道路系統之研究，國科會專題研究，民國91年。
25. 台北縣交通空氣污染改善減量成效評估計畫，北縣環保局，民國91年。
26. 機場噪音輔助執行項目分析評估，空軍總部，民國91年。