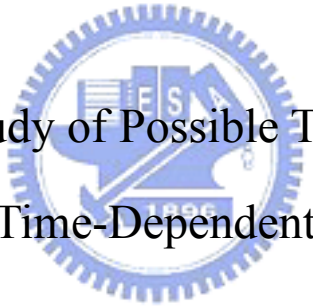
國 立 交 通 大 學

運 輸 科 技 與 管 理 學 系

碩 士 論 文

探討時間相依之可能旅行路徑問題



A Study of Possible Travel Routes in Time-Dependent Networks

研 究 生： 簡 嘉 英

指導教授： 王 晉 元

中 華 民 國 九 十 五 年 六 月

探討時間相依之可能旅行路徑問題

A Study of Possible Travel Routes in Time-Dependent Networks

研究生：簡嘉英　　Student：Chia-Ying Chien

指導教授：王晉元　　Advisor：Jin-Yuan Wang

國 立 交 通 大 學

運 輸 科 技 與 管 理 學 系

碩 士 論 文

A Thesis

Submitted to Department of Transportation Technology and Management

College of Management

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Transportation Technology and Management

July 2005

Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

探討時間相依之可能旅行路徑問題

學生：簡嘉英　　指導教授：王晉元

國立交通大學運輸科技與管理學系碩士班

摘要

探討時間相依之旅行路徑為先進旅行者資訊系統（ATIS）發展重點之一。然而過去研究多著重在求解時間相依之最短路徑問題，忽略其發生機率可能極低，因此本研究之目的在於考量具有時間相依性之節線成本情況下，探討不同時空下對於到達時間和旅行路徑之影響，以作為旅行者旅行路徑規劃之參考依據。

在網路資訊的設計上，本研究加入不同時段對於旅行時間影響之概念，並且以時間範圍和相對應之機率方式表示。首先修正 Miller-Hooks and Mahmassani (1998 提出的演算法，找出時間相依之最短路徑；接著，依據標記修正法，並融入旅行路徑機率之概念，求解單一路徑在相同出發時間下，不同到達時間範圍以及機率問題；最後以虛擬路網測試，驗證其可行性，並探討在給定不同相關旅行資訊之前提下，所能提供旅行者資訊之限制，確定本研究所提出之演算法適用性。

關鍵字：時間相依、旅行時間、先進旅行者資訊

Possible Travel Routes in Time-Dependent Networks

Student： Chia-Ying Chien　　Advisor：Jin-Yuan Wang

Department of Transportation Technology and Management

National Chiao Tung University

Abstract

Time dependent route guidance and travel estimation are important elements in a Advanced traveler information system（APTS）. Most previous researches focus on providing shortest paths; but neglect the occurrence probability. This shortage results in the extreme low probability of the provided route.

We propose three procedures to determine possible travel routes and their associated probabilities in a time-dependent network. The first and the second algorithm generate possible travel time range of the suggested path based on the algorithm proposed by Miller-Hooks and Mahmassani (1998). The third algorithm calculates the occurrence probabilities of a given route. The computation complexity of these three algorithms is also analyzed.

We use simulated data to evaluate performance of these three algorithms. The testing results shows that these algorithms are efficient and can provide useful information to travelers.

Keywords： Time Dependent Shortest Path、ATIS、Traveler Information

誌謝

本論文得以順利完成，最感謝的莫過於　王晉元老師了。在研究所兩年來，從一開始唸 paper 老是抓不到重點，到後來慢慢學會分析一篇 paper 的優缺點，以及構思演算法以及撰寫論文的過程中，恩師總是不厭其煩地給予指導和幫助。除了研究之外，生活的待人處事方面，學生也受益良多。

感謝中華大學運輸科技與物流管理學系蘇昭銘老師以及成功大學交通管理科學系胡大瀛老師在論文口試過程中惠賜卓見及不吝指正，使本論文更臻完備。

在 ITSLAB 三年來的日子轉眼即逝，在這段時間內非常高興可以認識一群好朋友。感謝大師兄、小松等學長在我遇到疑惑時，總是能給予適切的解答與幫助，還有去年論文誌謝的電子檔拿來借我直接改，真是太感謝了！感謝 hoho 跟 how 幫我解決電腦的疑難雜症，感謝認真魔人小翔總是提醒我什麼時候該做什麼事情、該交哪些東西，不然我的論文也不會那麼早完成；另外也感謝彥佑、瑞豐、dozo、思文、文誠、LC 能在研究之餘帶給我歡樂！

最後要感謝一直在背後默默陪伴我的家人，每當我遇到挫折時，他們總是能鼓勵、支持我，並且能讓我在求學過程中無後顧之憂。

這篇論文可以順利完成，要感謝的人實在是太多了，最後再次感謝這些年來各位的支持和陪伴，希望能在此與你們共同分享這份喜悅，謝謝大家!

簡嘉英

新竹交大

2006/07/07

目　　錄

圖　　目　　錄

表　　目　　錄

第一章　緒論

1.1　研究動機

近年來，隨著資訊科技和通訊系統發展迅速，智慧型運輸系統（Intelligent Transportation Systems, ITS）逐漸受到重視，其中先進旅行者資訊系統(Advanced Traveler Information System, ATIS)之目標在於提供旅行者必要的旅行資訊，使其順利地達到目的地；因此如何提出有用而且準確率高之訊息是許多人關心之課題。

在世界各國已有不少發展先進旅行者資訊系統應用於私人運具和大眾運輸系統上之案例[2]，例如私人運具上所安裝的定位導航系統，可提供駕駛者查詢從起點至迄點的路線導引功能，在大眾運輸部份則以架設查詢網站形式，並加入運具轉換之考量和步行導引說明[14]，都將有助於初次到達或不熟悉該地之旅客順利到達目的地，對於交通觀光之發展極為有利。此外，在現有的路線導引應用系統部分，為符合使旅客最快完成其旅次之目的，多半以最短路徑方式呈現結果。

儘管路線導引系統之目的在於提供最快到達之路線資訊，然而受限於資料蒐集不易、相關技術之發展等因素，使得過去系統多採用靜態資料，即網路節線成本以定性的距離或時間等方式表示，並未考慮時間相依之網路節線成本問題，即網路節線成本應隨著到達該節線時間不同而變動，而非以從起點出發時之節線成本來看；實際應用上，時間相依之成本卻是影響旅行時間的最大因素，因此為提供旅行者更有效的行前資訊，估計時間相依之最短路徑才能更符合需求。

雖然達到「在最短時間內到達目的地」是最理想化的目標，然而其發生的機率值可能極低，甚至趨近於零，故可行性不高；此外，路線導引系統為達到分散車流之目的，大部分採用呈現多種資訊之方式供使用者選擇多元化。因此，行前資訊之路線規劃不僅侷限於最短路徑、最快到達之單一結果，尚須提供更多有效的建議方案，並加入相對應的發生機率風險之考量較為適當，例如

在給定預計到達迄點之時間下，提供最短路徑的旅行時間誤差範圍，或者，在某一時間點出發可準時於預計時間到達的機率風險等資訊，甚至若能具體呈現出各種到達時間範圍以及相對應的發生機率，都能提供給使用者作為判斷選擇旅行路徑和決定出發時間之參考依據，比如說假設某甲希望早上八點半出門，十點能到迄點，而旅行資訊計算結果為九點半到十點到達的機率為 0.3，10 點到 10 點半到達的機率為 0.7，則某甲即可就此資訊決定是否須提早出門，以確保準時到達迄點。

1.2 研究目的

本研究之目的為在考量具有時間相依性之節線成本情況下，探討不同時空下對於到達時間和旅行路徑之影響，即依照不同的起迄點、預定到達時間等條件，產生各種不同的結果，包括最短路徑之最早應出發時間、最晚可出發時間以及在某一出發時間對於準時到達目的地之風險，讓旅行者可視即時路況或根據自我需求改變其決策，免除因為不確定性所造成不必要的旅行時間和等候時間之浪費。

1.3 研究範圍

本研究將有下列五項限制和假設前提：

1. 網路節線旅行成本具有時間相依性，因此適用於非先進先出（non-FIFO）之網路，意即較早從起點出發未必會先到達迄點。
2. 相關的節線旅行時間成本以及其機率分配資訊已知。
3. 本研究將一天區分為不同的時段，如尖峰和離峰，以便呈現不同時段的旅行時間之差異。
4. 網路節線之旅行時間以時間間距表示，例如從節點 a 至節點 b 所需花費之旅行時間為 10 至 15 分鐘，將更符合現實生活之網路模式。

1.4 研究流程

本研究的研究流程如圖 1-1 所示，茲將流程圖中各步驟詳細說明如下：

1. 描述與界定問題

依據目前實際路網，提出適用之路徑問題，並根據研究動機與目的將問題作一完整的描述與界定。

2. 蒐集與回顧文獻

蒐集國內外探討時間相依之最短路徑問題之相關文獻，並回顧這些文獻之解決方法與演算法，比較分析不同解法的限制條件、優缺點與適用性，再從中挑出一個較符合本研究之解法，針對其不合理性，修正為較符合現況之設計。

3. 建構時間相依之網路模式

根據目前實際路網可提供之相關資料形式，建構一時間相依之網路模式來描述求解旅行路徑的問題特性與相關的限制條件。

4. 設計求解模式之演算法

利用所設計之演算法求解不同時空下旅行路徑之問題。

5. 撰寫程式

6. 進行測試範例求解

設計一測試範例以驗證本研究所提出之演算法之可行性。

7. 分析與評估測試結果

分析求解結果，評估本研究提出之演算法之正確率，並和文獻回顧之演算法比較其效率性。

8. 修正演算法

依據測試結果的分析，若演算法正確且能夠在可容忍的時間內求解

則不需要修正，反之則重新設計或修正求解模式之演算法。

9. 結論與建議

對本研究之過程與結果提出結論與建議。

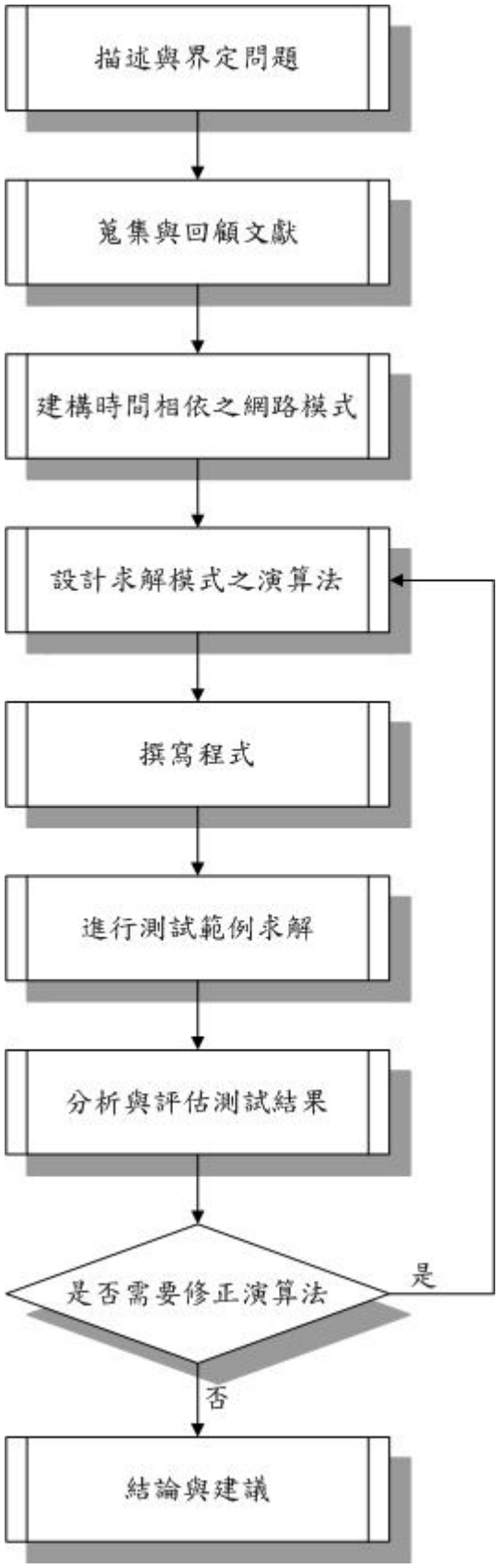

描述與界定問題
蒐集與回顧文獻
建構時間相依之網路模式
設計求解模式之演算法
撰寫程式
進行測試範例求解
分析與評估測試結果
是否需要修正演算法
是
否
結論與建議

圖 1-1　研究流程圖

第二章　文獻回顧

本研究主要目的在於提供具有時間相依性之相關路徑資訊，以便旅行者選擇，故需針對不同旅行時間提出適合之演算法。因此，本章首先先定義最短路徑問題，以及時間相依最短路徑問題，並彙整過去相關的文獻，探討各種演算法對於旅行時間之計算和處理方式，加以分類與討論，從中找出最符合本研究之問題特性之方法，以作為本研究演算法設計之基礎。

2.1　最短路徑問題

探討最短路徑問題，首先須對其定義：令一網路圖形為G（V , E），其中V為由n個節點（node）所組成的集合，表示為V={ V_1, V_2、、、V_n }；E是由m條節線（link）所構成之集合，即E={ E_1, E_2、、、E_m }，通常m≧n。令C（u ,v）為兩相鄰節點u與節點v間的節線（u ,v）之旅行成本，其中（u ,v）∈E，u、v∈V，C（u ,v）≧0。最短路徑問題即在於給定起點s和迄點t之條件下，s、t∈V，求解從起點到迄點的總旅行成本最短之路徑。

最短路徑問題根據起迄點之不同型態，大致可分為三類，依序為（1）由特定起點至特定迄點（一對一）之最短路徑問題（one-to-one shortest path）；（2）由一特定點至網路中其他各點（一對多）之最短路徑問題（one-to-all shortest path）；以及（3）網路中各點間（多對多）之最短路徑問題（all-to-all shortest path）。

基本上，一對一之最短路徑問題和一對多之最短路徑問題之求解方式一致，主要差異在於後者是將前者之終點延伸至網路中的所有點。依照過去文獻關於一對一之最短路徑問題之解法，最為眾人熟知的主要有兩種，分別為Dijkstra [8]標記設定法（Label Setting Algorithm），以及Ford [13]標記修正法（Label Correcting Algorithm）。Dijkstra 標記設定法僅適用於網路中無負成本之節線存在，其求解理論為給予每個節點一個暫時性標記（Label），代表由起點至此節點的旅行成本，隨著演算步驟的循環，這些標記亦隨著更新，且每次

循環均會產生一由暫時性標記轉換而成的永久性標記，此標記即為由起點至此節點之最小旅行成本，其計算之時間複雜度為O（n^2）。Ford標記修正法則不限定網路中節線成本為正或負，但若有負節線成本存在，仍須有偵測負迴圈存在之機制；其概念在於反覆使用k階的最短路徑，改善成k+1 階的最短路徑，其計算之時間複雜度為O（n^3）。

過去在求解多對多之最短路徑問題主要有 Floyd[15]多對多最短路徑演算法，其不侷限於非負節線成本之網路，若有負值節線成本存在仍適用；基本概念為上述一對一最短路徑解法之延伸，將每一節點均視為終點，做一對一最短路徑解法以求解。

此外，最短路徑問題可依照時間相依之特性，區分為以下四種類型：

1. 一條最短路徑（Shortest Path）：此類問題為最基本的最短路徑問題。在一網路中給予特定的起迄點，根據最短路徑演算法即可求解。
2. K 條最短路徑（K-Shortest Path）：此種問題就是所謂的多條最短路徑問題，主要針對不同使用者有不同的需求所致。在一網路中給予特定的起迄點，經由演算法計算後，可提供 K 條不同的路徑資訊供旅行者選擇。
3. 時間相依的一條最短路徑（Time-Dependent Shortest Path）：前述的兩種問題均只考慮靜態網路之情況，但就現實面來看，實際道路交通之情況中，網路之路線成本應會隨著時間而變化；因此，加入時間對於成本之影響考量，將可更準確的提供最短路徑資訊。
4. 時間相依的 K 條最短路徑（Time-Dependent K-Shortest Path）：如同上述，在考慮時間性之影響下，提供 K 條最短路徑資訊。

2.2 時間相依最短路徑問題

一般而言，探討最短路徑問題，其所謂的網路節線旅行成本可區分為三類，依序為：旅行距離、直接成本以及旅行時間[12]。本研究主要針對時間相依路徑問題著手，故將節線旅行時間視為網路節線成本，其與 2.1 所提到的最

短路徑問題之差異僅在於網路節線成本之性質不同，基本求解概念仍一致。

基本的最短路徑問題中，網路中任兩相鄰之節點i、j之旅行時間t_{ij}恆為常數；時間相依最短路徑問題(time-dependent shortest path problem)，則將網路中任兩點相鄰之節點i、j之旅行時間視為與出發時間(t_i)相依之變數，令為$t_{ij}(t_i)$，其值會隨著從節點i出發之時間不同而改變。

本研究對於時間相依最短路徑問題之定義如下：令一網路圖形為G（V，E），其中V為由n個節點所組成的集合，表示為V={V_1, V_2、、、V_n}；E是由m條節線所構成之集合，即E={E_1, E_2、、、E_m}，一般情況下m≧n。設t（u ,v）為兩相鄰節點u與節點v間的節線（u ,v）之旅行時間成本，其中（u ,v）∈E，u、v∈V，t（u ,v）≧0，且節線旅行時間t（u ,v）為一隨機變數，其數值與到達該節線之時間有關，可視為一連續型隨機過程(continuous-time stochastic process)；時間相依最短路徑問題乃在給定起點s和迄點t之下，s、t∈V，求解從起點到迄點的總旅行時間最短之路徑。

2.3 時間相依最短路徑之文獻回顧

本節主要針對過去文獻中所提出之時間相依最短路徑演算法，以及其對於節線旅行時間之計算方式加以說明。

陳慧琪[1]以考慮是否延後出發時間與繞路行進之情況下，發展所適用之時間相依最短路徑演算法。乃將旅行時間分為三種型態：1.旅行時間為一時間相依之隨機變數，且使用單一期望值函數滿足時間相依和尖離峰之特性；2.旅行時間為一時間相依之隨機變數，且期望值函數隨出發時間所屬時段而變動；3.將旅行時間視為定性，將一天分為數個時段，各時段有其相對應的旅行時間。當旅行時間視為滿足一統計分配之隨機變數時，利用模擬方法來求解，最後再以標註設定法為基礎加以修正。經由測試結果得知，不論旅行時間為隨機變數或定性，考慮延後出發與繞路行進之演算法所求得之總旅行時間均較不考慮之總旅行時間小，因此能提供使用者一條有效的行前路線建議。

Azaron and Kianfar [3]使用隨機的動態規劃找到從起點至迄點的動態最短

路徑，其節線長度為呈指數分配的獨立隨機變數。每個節點均有一隨連續時間馬可夫鏈（continuous time Markovprocess）變化的環境變數，每個節線過渡時期（transition time）的指數分配參數也是起始節點環境變數的狀態函數。假設到達每個節點，即可得知其環境變數的狀態以及鄰近節點的環境變數狀態，因此可決定朝最佳的節線前進或等待。在航線規劃問題，儘管已知所有節點的環境變數，然而在測試範例的時間複雜度卻呈現指數成長，故不適用。

Ziliaskopoulos and Mahmassani [4]主要求解網路中任一節點至一特定終點（多對一）之定性且時間相依之最小可能時間路徑問題，且適用於非先進先出(non-FIFO, Non First-In-First-Out)情況。此研究以最佳解的Bellman最適化原則為基礎，將所考量的時間範圍分割成M個時間間隔(time interval)，且每個節點及時間間隔均有一個相對應的路段旅行成本。其提出從所有點到終點之最短旅行成本演算法，乃修正自一對多的標記修正法，採用反推之方式（backward），意即將特定終點視為起點，網路其他節點視為迄點加以計算；演算法首先由終點出發，求得終點的所有上游點到達終點的最短旅行成本，將有更新最短旅行時間值之上游點，加入到Scan Eligible List（SE List）當中；再由SE List當中挑選第一順位節點出發，求得該點的上游點當中之最短旅行成本值有變化者，加入到SE List當中，直到SE List當中為空集合時為止。此演算法所需要的演算時間複雜度為$O(V^3M^3)$，其中V為節點總數。

Miller-Hooks and Mahmassani [6]採用 Ziliaskopoulos and Mahmassani [4]之架構，不同之處在於此篇研究將定性且時間相依之網路改為動態且隨時間變化之網路，求取尖峰時段不同起點出發（多對一）之時間相依最短路徑。假設隨機網路節線權重相互獨立，且可適用於非先進先出之網路。第一個演算法針對每個出發的時間間格，求出從網路中的任一點至迄點之最小可能時間路徑、最小可能旅行時間以及其相對應的機率下界值；第二個演算法則提升至找出 K 條最小可能時間之路徑和其相對應的機率下界值。此兩種方法均能有效找出最短路徑，且 SE List 架構提供更多資訊時，計算的時間複雜度仍不會增加太多。

Miller-Hooks [7]主要延續 Miller-Hooks and Mahmassani [6]之研究，由原先提供可能最小旅行時間路徑之機率值，修正為計算最小旅行時間之期望值；此資訊較符合現況，有助於使用者參考。

Fu and Rilett[9,10]之目的在於估計每天固定發生的網路狀態下之動態起迄點之旅行時間。首先將網路依照都市型態分為三個區域，假設區域中的每條節線旅行時間曲線一致，並將一天時間依照尖離峰分為三個時段，以離峰時段之旅行時間為基礎，尖峰時段旅行時間則設定為常態分配變數。此研究提出一兩階之 feed forward 類神經網路（Artificial Neural Network, ANN）將不同時段之旅行時間行為模式化，結果驗證類神經網路可預測在動態網路下兩地間的旅行時間，亦可追蹤兩地間的動態旅行時間型態，且在非循環式的壅塞網路下，此法預測之旅行時間可能亦較佳。

Davies and Lingras[5]放鬆區別（distinct）所有節點的限制，求解動態網路最短路徑（shortest walk）問題。類似於 Fu and Rilett[9,10]之作法，並假設節線權重（weight）為已知的時間函數。主要採用基因演算法（Genetic Algorithms, GAs），延伸至重新規劃最短路徑問題（GAs for Rerouting Shortest Paths, GARSP），其犧牲結果的準確率以取得更迅速的執行時間；然而，GARSP 僅適用於較單純之網路，於高度動態，節線成本呈非線性之網路下，所得結果較修正後的 Dijkstra 演算法差。

Fu and Rilett[9,11]主要求解在給定網路的出發時間下，求得從起點至迄點的最小路徑。首先將網路節線的旅行時間設定為連續的隨機變數，且其機率分配和一天的時間是相依的，並假設可獲得不同時間點的平均值和變異數，以及二次微分。此研究根據 K 條最短路徑演算法，在網路上沒有交通事故發生之前提下，提出一套啟發式演算法，其測試結果找出較佳解，且僅小幅增加 K 值和計算時間，然而其節線旅行時間資料為虛擬值，故無法得知是否適用於實際路網。

Fu and Rilett[12]探討於資訊可從智慧型運輸系統獲得之情況下，求解動態與隨機最短路徑問題（Dynamic and Stochastic Shortest Path Problem, DSSPP）。動態與隨機網路表示每條路徑之旅行時間均為一機率變數，且其機率分配與該節線出發的時間有關，意即此類網路中的節線旅行時間為連續時間型的隨機過程（continuous-time stochastic process），且假設在不同時間點，每個節線的旅行時間相互獨立。首先利用統計方法了解動態與隨機網路中的路徑旅行時間平均值與變異數的關係，證明隨機動態最短路徑問題在計算上相當棘手複雜，故

此研究根據 K 條最短路徑演算法概念，將時間相依最短路徑求解總旅行時間的問題，轉換為求解到達時間的問題，並利用泰勒展開式將旅行時間期望值展開，利用取其二階近似值的方式，求解到達時間的期望值。其測試結果發現 K 值越大結果越好，且增加的計算時間均在可容忍範圍，但是節線旅行時間資料卻為虛擬值，故無法得知是否適用於實際路網。

Waller and Ziliaskopoulos[16]主要探討在考慮時間和空間相依之旅行成本下的隨機最短路徑問題，適用於當使用者已進入網路後，仍可依實際狀況隨時調整其路線規劃。空間相依的最短路徑問題（Spatially Dependent Online Shortest Path, SD-OSP）即假設節線的成本相依性只受鄰近節線影響，此研究設定每一節線只和其一上游點相依（one-step spatial dependence），且網路節線成本必須大於零，亦不允許在節點上等待；空間相依最短路徑問題（Temporally Dependent Online Shortest Path, TD-OSP）假設到達任一節點時，即可得知連結此節點之節線成本，且未來短期的節線成本可根據目前的觀察值產生；時空相依的最短路徑問題（Temporally Spatially Dependent Online Shortest Path, TSD-OSP）有兩項假設前提，為須有即時監控系統收集下游點的資訊，且經過一節線所需的旅行時間很小，不至於在經過期間其旅行時間成本會有所改變。此兩種問題均修正自多對一的標註修正法，求解定性且隨時間而變的最短路徑，且即使增加網路複雜度，求解品質和計算時間均不會改變太大。

2.4 小結

上述文獻中，對於時間相依最短路徑問題之節線旅行時間之表示，多半採用定性、離散型分配的歷史統計資料；其主因在於現實生活中，旅行時間之資料取得困難，相關的旅行時間函數之建立也會相當複雜，故為了方便求解，而犧牲隨時間變化之旅行成本考量。

關於時間相依最短路徑問題之求解方法部分，除了少數利用馬可夫鏈、類神經網路和基因演算法以外，大多演算法之發展仍採用標記修正法之概念作為研究基礎，再根據不同的研究範圍與目的加以修正。

綜合以上所述，過去文獻在解決時間相依最短路徑問題上，仍未完整表示時間相依之節線旅行時間特性，且極少探討相關的風險問題，因此依然無法滿足本研究之發展目的；惟 Miller-Hooks and Mahmassani[6]第一個演算法所提出之機率表示概念較為相近，故本研究將針對此演算法配合本研究目的加以修正。

第三章　研究架構

在過去文獻中，採用各種不同方法求解最短路徑問題，但在表達時間相依之節線成本上仍有些不足之處，且大多未考慮到發生機率風險層面的問題；在少數論及最短路徑發生可能性之文獻中，以 Miller-Hooks and Mahmassani [6]所提出的第一個演算法提出機率之概念與本研究較為相近，因此本章首先針對第一個演算法作一介紹說明，再分析其優缺點，最後依據本研究目的加以修改，提出適用之演算法，並做一整體說明。

3.1　Miller-Hooks and Mahmassani 之演算法架構

本節將說明 Miller-Hooks and Mahmassani [6]中第一個演算法架構，首先定義欲求解之問題，再說明求解方法和演算法架構，最後以此方法求解一簡單範例。

3.1.1　問題定義

Miller-Hooks and Mahmassani [6]之研究為在一個具有方向性之網路，針對尖峰時段（peak period），求解時間相依之最小可能旅行路徑問題。該研究假設網路之隨機節線成本互相獨立，且為一非先進先出之網路，節線旅行時間則以時間相依之非負且不連續隨機變數的分配函數（distribution function）表示，在超過尖峰時段之後的旅行時間不隨時間改變，因此僅觀察尖峰時段。並假設在實際狀況中，旅行時間集合和其相對應的機率集合均為已知。

首先介紹相關變數之定義，分述如下：

G：一具有方向性之網路；$G = (V, A, I, T, P)$。

V：節點的集合，$|V| = n$。

A：節線的集合，$|A| = m$。

I：所有節點的出發時間集合。

T(t)：在每一個出發時間$t \in I$，每個節線$(i, j) \in A$可能的非負節線旅行時間$\tau_{i,j}^{k}(t)$之集合；其中，k=1,…,$K_{i,j}(t)$，$K_{i,j}(t)$為在時間 t 節線(i,j)的旅行時間編號。

P：旅行時間$\tau_{i,j}^{k}(t)$的發生機率$\rho_{i,j}^{k}(t)$之集合，故$\sum_{k=1}^{K_{i,j(t)}} \rho_{i,j}^{k}(t) = 1$，$\forall t \in I$。

3.1.2 求解方法

此演算法主要修正自標記修正法，以求解任一節點至迄點 N 之最小可能旅行時間以及其相對應之路線和機率下界值。首先設定在網路中的各節點於不同時間出發所需紀錄的標籤（label）包含兩項資訊，分別為從該節點至迄點的最小可能旅行時間上界值以及相對應的發生機率下界值。

此演算法採用反推之方式，即從迄點作為起始點，其他節點視為迄點。演算法中的每個步驟均需計算目前節點之每一出發時間間隔的暫時標籤，並且和此節點現在的標籤比較，若暫時標籤有較小值或機率較高的相同值，則取代目前的標籤，否則不需進行更新；如此反覆計算將可求解。

各項變數設定如下所述：

$[\lambda_i^m(t)]_{m\in\{1.2\}}$：節點 i 在時間 t 出發時的標籤值。當演算法結束，$\lambda_i^1(t)$為在時間 t 節點 i 的最小旅行時間上界值，$\lambda_i^2(t)$為其相對應的發生機率下界值。

$\pi_i^1(t)$：在時間 t 於節點 i 的下游點。

$\pi_i^2(t)$：在時間 t 從節點 i 出發到下游點所相對應的出發時間。

$\Gamma^{-1}(j)$：節點 j 的上游點集合。

$[\eta_i^m(t)]_{m\in\{1,2\}}$：在時間 t 從節點 i 出發的暫時標籤向量。其中 m=1 時，代表在時間 t 從節點 i 出發至迄點的總旅行時間，於演算法的任一步驟，令 $\lambda_i^1(t) = \min\{\lambda_i^1(t), \eta_i^1(t)\}$；m=2 時，代表 $\eta_i^1(t)$ 所相對應的可能機率值。

3.1.3 演算法之步驟

Miller-Hooks and Mahmassani [6]第一個演算法的步驟如下：

步驟 0：初始化並建立 Scan Eligible List

1. 節點標籤初始化

初始化標籤和路徑 pointers

$\lambda_i^1(t) = \infty \qquad \forall i \in V \setminus N, t \in I$

$\lambda_N^1(t) = 0 \qquad \forall t \in I$

$\lambda_i^2(t) = 1 \qquad \forall i \in V, t \in I$

$\pi_i^c = \infty \qquad \forall i \in V \setminus N, t \in I, p \in \{1,2\}$

$\pi_N^1 = N$ 且 $\pi_N^2 = 1 \qquad \forall t \in I$

flag = 0

2. Scan Eligible List 初始化

在 SE list 插入節點 N。

步驟 1：選擇目前節點

如果 SE List 為空集合，則跳至步驟 3；如果 SE list 不為空集合，則從 list 集合中選取第一個節點，為節點 j。

步驟 2：更新節點標籤

1. 建立暫時的標籤

利用式(1)決定在每個 $i \in \Gamma^{-1}(j)$ 、 $\forall t \in I$ 以及每個從 i 出發的可能旅行時間 $\eta_i^1(t)$

$$\eta_i^1(t) = \min_p \{\tau_{i,j}^p(t) + \lambda_j^1(t + \tau_{i,j}^p(t))\} \quad \text{.....} \quad (1)$$

其中，p 為在時間 t 節線(i,j)可能旅行時間的索引集合；

$q = \arg\min_p \{\tau_{i,j}^p(t) + \lambda_j^1(t + \tau_{i,j}^p(t))\}$ ，

即回傳 p 值使得 $\{\tau_{i,j}^p(t) + \lambda_j^1(t + \tau_{i,j}^p(t))\}$ 最小；

$s = \max_q \{\rho_{i,j}^q(t) \times \lambda_j^2(t + \tau_{i,j}^q(t))\}$ ，

即回傳使得 q 值 $\{\rho_{i,j}^q(t) \times \lambda_j^2(t + \tau_{i,j}^q(t))\}$ 最大；

$\eta_i^2(t) = \rho_{i,j}^s(t) \times \lambda_j^2(t + \tau_{i,j}^s(t))$ 。

需特別注意的是，可能會有一個以上的 p 值滿足式(1)，則當這種狀態出現時，為有效紀錄路線，需加入機率之考量，即選擇具有較高發生機率之路線。

2. 標籤比較

若 $\{\eta_i^1(t) < \lambda_i^1(t)$ 或者 $(\eta_i^1(t) = \lambda_i^1(t)$ 且 $(\eta_i^2(t) > \lambda_i^2(t))\}$ ，則 $\lambda_i^m(t) = \eta_i^m(t)$

$\forall m \in \{1,2\}$ ； $\pi_i^1(t) = j$ ， $\pi_i^2(t) = \min\{t + \pi_{i,j}^s(t), t_0 + I\delta\}$ 。

flag = 1 代表節點 i 應加入至 SE List；若 $i \notin SE$ 且 flag = 1，則將節點 i 加入至 SE List。若所有 $i \in \Gamma^{-1}(j)$ 都已計算完畢，則回到步驟 1。

步驟 3：停止。

演算法結束會得到最小可能時間路線，以及相對應的時間值和發生機率的下界值。

3.1.4 演算法之範例

在此以 Miller-Hooks and Mahmassani [6]附錄 B 之第一個演算法的測試範例作為說明：

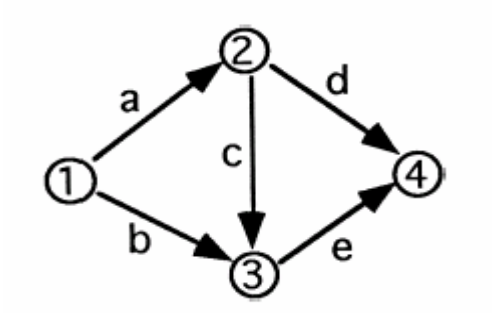


圖 3-1　Miller-Hooks and Mahmassani 演算法範例說明之網路圖[6]

表 3-1　範例網路旅行分佈函數

資料來源：Miller-Hooks and Mahmassani [6]

Arc a		Arc b		Arc c		Arc d		Arc e	
t=0	t=0	t=2	t=3	t=2	t=3	t=4	t=5	t=6	t=7
2（0.5）	5（0.4）	4（0.2）	1（0.4）	3（0.8）	6（0.4）	4（0.2）	5（0.3）	2（0.1）	3（0.3）
3（0.5）	7（0.6）	5（0.8）	3（0.6）	7（0.2）	7（0.6）	6（0.8）	8（0.7）	9（0.9）	4（0.7）

本範例為求解從圖 3-1 網路上的各個節點在不同時間出發之情況下，到達節點 4 的最短路徑，以及所相對應的機率值，亦即為分別將節點 1、節點 2、節點 3 視為起點，節點 4 視為迄點。詳細演算步驟如下：

Iteration1

從以上資訊可知各項初始化值：

節點 1 至節點 4 所有可能發生的出發時間集合 I = {0,1,2,3,4,5,6,7}

V = {1,2,3,4}

A = {a,b,c,d,e}

迄點 N = 4

步驟 0

$\lambda_i^1(t) = \infty \qquad \forall i \in \{1,2,3\}, t \in I$

$\lambda_4^1(t) = 0 \qquad \forall t \in I$

$\lambda_i^2(t) = 1 \qquad \forall i \in V, t \in I$

$\pi_i^p(t) = \infty \qquad \forall i \in \{1,2,3\}, t \in I, p \in \{1,2\}$

$\pi_4^1(t) = 4 \qquad \forall t \in I$ （此非必要條件，除非 $\lambda_N^1(t) > 0$ ）

$\pi_4^2(t) = 4 \qquad \forall t \in I$

將迄點加入 SE list 中，SE = {4}

步驟 1： 從 SE = {4} 中取出第一個節點設為目前節點 j = 4, SE = {}

步驟 2

對節點 4 而言，其上游點 $i \in \{2,3\}$，依序計算，故先選擇節點 2。

$i = 2$，計算各個時間點從節點 2 出發所需花費的旅行時間和相對應的機率值，對節點 2 而言，需經過節線 d 才能到達節點 4，可能出發時間有 t=2 以及 t=3 兩種，依序討論：

$t = 2$，

節點 2 於時間 t=2 出發至節點 4 有兩種可能旅行時間，取較小值

$$\eta_2^1 = \min_{k=1,2}\{\tau_{2,4}^1(2) + \lambda_4^1(2 + \tau_{2,4}^1(2)), \tau_{2,4}^2(2) + \lambda_4^1(2 + \tau_{2,4}^2(2))\}$$

$$= \min_{k=1,2}\{3 + \lambda_4^1(2+3), 7 + \lambda_4^1(2+7)\} = \min_{k=1,2}\{3+0, 7+0\} = 3$$

紀錄使 η_2^1 最大的 k 值，$q = \arg\min_{k=1,2}\{\tau_{2,4}^k(2) + \lambda_4^1(2 + \tau_{2,4}^k(2))\} = 1$

紀錄使η_2^2最小的 k 值，$s = \max_{q=1}\{\rho_{2,4}^1(2) \times \lambda_4^2(2 + \tau_{2,4}^1(2))\} = 1$

節點 2 於時間 t=2 出發至節點 4 最小旅行時間所可能發生的機率

$\eta_2^2 = \rho_{2,4}^1(2) \times \lambda_4^2(2 + \tau_{2,4}^1(2)) = 0.8 \times \lambda_4^2(2+3) = 0.8$

將暫時標籤和永久標籤比較，若暫時標籤較好則需更新永久標籤

$\eta_2^1 = 3 < \lambda_2^1(2) = \infty$

$\lambda_2^1(2) = 3$ 且 $\lambda_2^2(2) = 0.8$

$\pi_2^1(2) = 4$ 且 $\pi_2^2(2) = \min\{2+3,7\} = 5$

t = 3

$\eta_2^1 = \min_{k=1,2}\{6 + \lambda_4^1(3+6), 7 + \lambda_4^1(3+7)\} = \min_{k=1,2}\{6+0, 7+0\} = 6$

q = 1, s = 1

$\eta_2^2 = 0.4 \times 1 = 0.4$

$\eta_2^1 = 6 < \lambda_2^1(3) = \infty$

$\lambda_2^1(3) = 6, \lambda_2^2(3) = 0.4, \pi_2^1(3) = 4, \pi_2^2(3) = \min\{6+3,7\} = 7$

永久標籤被更新，故需將節點 2 加入至 SE List，因此 SE = {2}。

i = 3

$\lambda_3^1(4) = 4, \lambda_3^2(4) = 0.2, \pi_3^1(4) = 4, \pi_3^2(4) = 7$

$\lambda_3^1(5) = 5, \lambda_3^2(5) = 0.3, \pi_3^1(5) = 4, \pi_3^2(5) = 7$

$\lambda_3^1(6) = 2, \lambda_3^2(6) = 0.1, \pi_3^1(6) = 4, \pi_3^2(6) = 7$

$\lambda_3^1(7) = 3, \lambda_3^2(7) = 0.3, \pi_3^1(7) = 4, \pi_3^2(7) = 7$

SE = {2,3}

Iteration2

步驟 1： j = 2，SE = {3}

步驟 2

i = 1，t = 0， $\lambda_1^1(0) = 5, \lambda_1^2(0) = 0.4, \pi_1^1(0) = 2, \pi_1^2(0) = 2$

SE = {3,1}

Iteration3

步驟 1： j = 3, SE = {1}

步驟 2： i = {1,2}

i = 1，$\eta_1^1 = 10 < \lambda_1^1(0) = 5$ ，因此不更新

i = 2

t = 2，沒有更新

t = 3， $\lambda_2^1(3) = 5$，$\lambda_2^2(3) = 0.08$，$\pi_2^1(3) = 3$，$\pi_2^2(3) = 4$

SE = {1,2}

Iteration4

步驟 1： j = 1, SE = {2}

步驟 2： i = {}

Iteration 5

步驟 1： j = 2, SE = { }

步驟 2： i = {1}, i = 1 沒有更新

Iteration 6

步驟 1： SE = { }

步驟 3： SE List 為空集合，故結束計算，最後結果如表 3-2 所示。

表 3-2　Miller-Hooks and Mahmassani [6]第一個演算法的範例結果

資料來源：Miller-Hooks and Mahmassani [6]

Origin node	Departure time	Travel time	Probability	Path
1	0	5	0.4	1－2－4
2	2	3	0.8	2－4
2	3	5	0.08	2－3－4
3	4	4	0.2	3－4
3	5	5	0.3	3－4
3	6	2	0.1	3－4
3	7	3	0.3	3－4

3.1.5　演算法之分析

Miller-Hooks and Mahmassani [6]提出在隨機且動態網路中決定可能最短時間路徑有效之方法；然而，其演算法僅針對最短路徑之問題作探討，卻忽略風險層面之考量，亦即為最短路徑所相對應之發生機率可能極小，以表 3-2 之範例結果為例，在時間等於 3 時從節點 2 出發，經由節點 3 到節點 4，其旅行時間雖然最小，但相對應的發生機率只有 0.08，且此範例僅有四個節點，若在大型路網中，最短路徑之發生機率甚至可能趨近於零，因此即使提供最小可能期望旅行時間之資訊，對於使用者而言仍為無效之資訊，於實際應用上也不一定最令人滿意，可見資訊之提供仍不夠充足。

此外，其演算法設定旅行時間為一時間相依之離散隨機變數，對於每個節點、每個時間間隔設定擁有特定的旅行成本，例如在 3.1.4 之範例中，節線 c 在時間 2 之旅行時間為 4 或 5 兩種；隨著網路增大時，各個節點所可能包含的出發時間點也會不斷增加，造成計算過於複雜。再者，其各節點的出發時間和各節線的旅行時間之呈現均以單一時間點表示，儘管其資料來源來自於歷史統計資料，旅行時間會受出發時間影響，相較於現實生活之情況仍有些差距。

因此，本研究根據上述之缺點加以修正網路和演算法架構，提出三種演算法。除了在給予預計到達迄點之時間情況下，提供最晚可出發時間和相對應之最短路徑，以及其路徑所相對應之最早應出發時間之外，並在給予預計出發時間且選擇最短路徑之情況下，探討可落在預計到達時段之機率，以呈現更多元化的資訊供使用者參考。

3.2 新演算法之網路架構和相關變數

根據 3.1 演算法之概念，可得知 Miller-Hooks and Mahmassani [6]網路架構之設計在表達時間相依之動態網路上仍有不足之處，如同 3.1.5 所述，各節線的出發時間和各節線的旅行時間之呈現應由單一時間點改為某時間範圍，因此本研究將針對網路架構作一修正。

首先，如同 Miller-Hooks and Mahmassani [6]之假設，網路之隨機節線權重互相獨立，且為一非先進先出之網路，並假設在實際狀況中，各節線之旅行時間範圍集合和其相對應的機率集合均為已知，此外儘管不允許節點間有停等之情況，但為非必要之假設。

一般探討時間相依之路網，大多將節線旅行時間分為尖峰和離峰時段，其中尖峰時段之旅行時間為動態，離峰時段之旅行時間為定性，故多僅討論尖峰時段[9]；然而即使是離峰時段，旅行時間仍會有所變動，只是變動性不如尖峰時段來得大，因此本研究根據 Fu and Rilett[9,10]之時段區分方法，先將各節線的旅行時間分為尖離峰時段表示，由於節線有方向性，即使同一節線上午和下午尖峰時段的旅行時間仍會不同，所以仍需分開討論，故一節線的旅行時間在三種不同時段均不同，依序為上午尖峰時段、下午尖峰時段以及離峰時段。

一節線在每個時段的旅行時間為呈現動態網路之形式，則以某時間範圍表示，以分鐘為單位，並假設在此時間範圍內的每個時間點發生機率為均質分配，且延續 Miller-Hooks and Mahmassani [6]之概念，同一節線同一時段之旅行時間在不同時間範圍有不同的發生機率，故亦需考慮機率問題。

綜合以上所述，整體網路架構之呈現將使得旅行時間隨著不同出發時間而

不同，且更能貼近實際狀況。以某節線（i,j）作為一範例說明（如表 3.3），於下午尖峰時段，經過此節線的旅行時間可能有 0.3 的機率需耗時 10 至 15 分鐘，也有 0.7 的機率需耗時 15 至 20 分鐘，依此類推，故在討論旅行路徑時，這些可能性均需加入考量。

表 3.3　網路呈現方式（以某節線為例）

	旅行時間 1（可能機率）	旅行時間 2（可能機率）
上午尖峰	15～20（0.3）	20~25（0.7）
下午尖峰	10～15（0.4）	15～20（0.6）
離峰時段	5～10（0.8）	10～15（0.2）

註：旅行時間之單位為分鐘。

此外，關於網路的各個相關變數之定義，分述如下：

G：一具有方向性之網路；$G=(V,A,T,P)$。

V：節點的集合，$|V|=$ n。

A：節線的集合，$|A|=m$。

T：節線（i,j）可能之非負節線旅行時間範圍集合。

P：節線（i,j）之旅行時間範圍之發生機率 $p_{i,j}^{s}$ 集合，故 $\sum p_{i,j}^{s}=1$。

Γ (i)：節點 i 之下游點集合。

Γ^{-1}(j)：節點 j 之上游點集合。

本研究所提出之演算法中，關於各項變數設定如下所述：

$[nt_i^1, nt_i^2]$：從節點 i 出發之時間範圍標籤。

ns_i^r：其中 r =1,2。當 r=1 時代表在時間範圍$[nt_i^1, nt_i^2]$從節點 i 出發之上游點標籤， 當 r=2 時代表在時間範圍$[nt_i^1, nt_i^2]$從節點 i 出發之下游點標籤。

p_i ：到達節點 i 在時間範圍$[nt_i^1, nt_i^2]$內所相對應之可能發生機率值。

$[t_{i,j}^k, t_{i,j}^{k+1}]$：節線（i,j）於到達節點 j 時之旅行時間範圍，其中，k=1,3,5,...,(2n-1)。

$[\bar{t}_{i,j}^k, \bar{t}_{i,j}^{k+1}]$：當經過節線（i,j）橫跨兩個不同時段時，節線（i,j）於跨時段後之旅行時間範圍，其中，k=1,3,5,..,(2n-1)。

$p_{i,j}^s$ ：$[t_{i,j}^k, t_{i,j}^{k+1}]$所相對應之可能發生機率值，其中 s=1,2,…,n。

$Q_{i,i}$：當經過節線（i,j）橫跨兩個不同時段之時間分界點。

ϑ_i^m：從節點 i 出發的暫時標籤向量，其中 m=1,2。當 m=1 時，代表從節點 i 出發時間；當 m=2 時，代表從節點 i 出發到迄點的路線中的下游點。

$[\gamma_i^1, \gamma_i^2]$：節點 i 的暫時標籤向量之一，代表從節點 i 出發的時間範圍。

ρ_i ：節點 i 的暫時標籤向量之一，代表從節點 i 在時間範圍$[\gamma_i^1, \gamma_i^2]$出發準時到達迄點的發生機率。

3.3 新演算法之架構

雖然從 3.1 節可得知最短路徑之可能機率值，在實際路網應用上，卻忽略此條路徑之可能發生機率極低，故本研究為加強過去研究僅探討最短路徑不足之處，提出三個演算法。首先，前兩個演算法主要為求解最短路徑可能的旅行時間範圍，分別為「可準時抵達目的地之最晚出發時間之分析」和「可準時抵達目的地之最早出發時間之分析」；第三個演算法則提供單一路徑的各種可能到達時間範圍和機率，稱為「各種到達時間之風險演算法」。

本研究為簡化演算法之說明，所有節線（i,j）之旅行時間範圍$[t_{i,j}^k, t_{i,j}^{k+1}]$僅取兩種，分別為$[t_{i,j}^1, t_{i,j}^2]$以及$[t_{i,j}^3, t_{i,j}^4]$，因此網路之各項變數設定可以表 3.4 為

例；然而，三種演算法均仍適用於多種旅行時間範圍，不失其一般性。

表 3.4　網路之變數表示（以某節線為例）

	旅行時間範圍 1（機率）	旅行時間範圍 2（機率）
原本時段	$t_{i,j}^{1} \sim t_{i,j}^{2}$（p1）	$t_{i,j}^{3} \sim t_{i,j}^{4}$（p2）
跨越後時段	$\bar{t}_{i,j}^{1} \sim \bar{t}_{i,j}^{2}$（$\bar{p}1$）	$\bar{t}_{i,j}^{3} \sim \bar{t}_{i,j}^{4}$（$\bar{p}2$）

前兩個演算法將同於 Miller-Hooks and Mahmassani [6]之概念，所有計算採用反推之方式，即從迄點作為起始點，其他節點視為迄點，便可從迄點之到達時間回推至起點之出發時間，故到達節點 i 之時間範圍計算如下，式（2）為最快之時間，式（3）為最慢之時間。

$$nt_i = nt_j - t_{i,j}^{1} \quad \text{.....} \quad (2)$$

$$nt_i = nt_j - t_{i,j}^{4} \quad \text{.....} \quad (3)$$

此外，在計算過程中，極有可能在經過某節線時會跨越兩個不同時段；以表 3.3 為例，假設上午尖峰時段為 AM 7：30 至 AM 9：00，到達節點 j 之時間為 AM 9：05，欲回推到達節點 i 之時間範圍，當節點 j 屬於離峰時段需要旅行時間 10 至 15 分鐘時，會跨越 AM 9：00，但在這以前屬於上午尖峰時段，因此節線（i,j）之旅行時間需加以修正；由於假設在同一路段之旅行時間範圍為均值分配，故修正後的旅行時間範圍為 $\bar{\tau}_{i,j}^{1} \sim \bar{\tau}_{i,j}^{2}$，如式（4），而其相對應之機率值不變。

$$\bar{\tau}_{i,j}^{1} = nt_j - Q_{\mathrm{i,j}} + \bar{t}_{i,j}^{1} \times \frac{t_{i,j}^{1} - (nt_j - Q_{\mathrm{i,j}})}{t_{i,j}^{1}}$$

到 $\bar{\tau}_{i,j}^{2} = nt_j - Q_{\mathrm{i,j}} + \bar{t}_{i,j}^{4} \times \dfrac{t_{i,j}^{1} - (nt_j - Q_{\mathrm{i,j}})}{t_{i,j}^{1}}$ …… （4）

以上述情況為例，$\bar{\tau}_{i,j}^{1} \sim \bar{\tau}_{i,j}^{2}$ 則如式（5）所示，即原本的旅行時間需 10 至 15 分鐘，修正為 12.5 至 21.67 分鐘。

$$\bar{\tau}_{i,j}^{1} = (9:05 - 9:00) + 15 \times \frac{10 - (9:05 - 9:00)}{10} = 5 + 15 \times \frac{10-5}{10} = 12.5$$

$$\bar{\tau}_{i,j}^{2} = (9:05 - 9:00) + 25 \times \frac{15 - (9:05 - 9:00)}{15} = 5 + 25 \times \frac{15-5}{15} = 21.67 \text{ ..（5）}$$

本研究的第三個演算法採用正推之計算方式，其旅行時間之推算概念仍類似於前兩個演算法，惟計算式需做些修正。由於需考慮所有可能發生的旅行時間範圍，因此從節點 i 出發到達節點 j 之時間範圍和機率會有兩種情形，計算如式（6）。

$nt_{\mathrm{j}}^{1} = nt_{\mathrm{i}}^{1} + t_{i,j}^{1}$ 且 $nt_{\mathrm{j}}^{2} = nt_{\mathrm{i}}^{2} + t_{i,j}^{2}$，$p_{\mathrm{j}} = p_{\mathrm{i}} \times p_{i,j}^{1}$

或者 $nt_{\mathrm{j}}^{1} = nt_{\mathrm{i}}^{1} + t_{i,j}^{3}$ 且 $nt_{\mathrm{j}}^{2} = nt_{\mathrm{i}}^{2} + t_{i,j}^{4}$，$p_{\mathrm{j}} = p_{\mathrm{i}} \times p_{i,j}^{2}$ …… （6）

此外，跨時段之旅行時間修正式如式（7），同樣的，其相對應之機率值不變。

$$\bar{\tau}_{i,j}^{1} = Q_{\mathrm{i,j}} + \bar{t}_{i,j}^{1} \times \frac{t_{i,j}^{1} - (Q_{\mathrm{i,j}} - nt_{\mathrm{i}}^{1})}{t_{i,j}^{1}}$$

$$\bar{\tau}_{i,j}^{2} = Q_{\mathrm{i,j}} + \bar{t}_{i,j}^{4} \times \frac{t_{i,j}^{2} - (Q_{\mathrm{i,j}} - nt_{j}^{2})}{t_{i,j}^{2}} \quad \text{……} \quad （7）$$

以表 3.3 為例，假設上午尖峰時段為 AM 7：30 至 AM 9：00，從節點 i 出發時間為 AM 8：55，欲計算到達節點 j 之時間範圍。從節點 i 出發之時間屬於上午尖峰時段需要旅行時間 15 至 20 分鐘時，會跨越 AM 9：00，但在這以後屬於離峰時段，因此節線（i,j）之旅行時間需加以修正；由於假設在同一路段之旅行時間範圍為均值分配，故修正後的旅行時間範圍為 $\bar{\tau}_{i,j}^{1} \sim \bar{\tau}_{i,j}^{2}$，即原本的旅行時間需 10 至 15 分鐘，修正為 8.33 至 16.25 分鐘，旅行時間之機率仍

如式（8）之 p_j 維持不變。

$$\bar{\tau}_{i,j}^{1} = (9:00-8:55)+5\times\frac{15-(9:00\text{-}8:55)}{15}=5+5\times\frac{15-5}{15}=8.33$$

$$\bar{\tau}_{i,j}^{2} = (9:00-8:55)+15\times\frac{20-(9:00-8:55)}{20}=5+15\times\frac{20-5}{20}=16.25\ldots.\text{（8）}$$

3.4 可準時抵達目的地之最晚出發時間之分析

對於旅行者而言，通常會希望在預計的時間點準時到達目的地，卻往往因為無法掌握旅行時間和行走路線，而不知何時出發較適合。因此，旅行者所需要的資訊除了旅行路線之外，尚須得知旅行時間，且其最為關切的即是如何在最小旅行時間內到達目的地，故本研究首先依據 3.2 節網路架構發展一個新的最小旅行時間演算法；首先定義欲求解之問題，再說明求解方法和演算法架構，最後以一簡單範例說明。

3.4.1 問題定義

此演算法主要在一個具有方向性之網路，且已知欲到達迄點 N 之時間，求解任一節點可準時到達迄點 N 之最晚出發時間和其相對應之路徑，以下簡稱為「A1 演算法」。

3.4.2 求解方法

本演算法和 Miller-Hooks and Mahmassani [6] 第一個演算法之求解方法極為相似，主要差異在於網路架構之不同，故旅行時間之計算方式亦有所不同，此部分在 3.3 節已說明，在此不再重複。求解問題前須先得知欲到達迄點時間；對每一節點而言，均具有兩個永久標籤，一個提供從該節點可能準時到達迄點的最晚出發時間，預設值為一極小值，另一個則紀錄路線資訊，即為從該節點至迄點的最小旅行時間路線之下游點。此外，需設定 SE list 用來記錄永久標

籤被更新的節點集合，由於本演算法採用反推之計算方式，因此一開始需先將迄點加入至 SE list。

在演算過程中，每個步驟需先從 SE list 取出第一個節點設為目前節點 j，並對其上游點 i 集合計算出發時間，設為暫時標籤，並且與節點 i 的永久標籤比較，若暫時標籤的出發時間較晚，則取代至永久標籤，或者若出發時間相同則同時記錄，否則不需更新永久標籤；如此反覆計算至 SE list 為空集合即可求解。

3.4.3 求解步驟

步驟一：初始化並建立 Scan Eligible List。

設定上午尖峰、下午尖峰以及離峰時段之範圍，令欲到達迄點 N 之時間為 nt_N，其他節點出發時間 nt_i = -1，其中 i ={1,2,…,N-1}，且 $ns_N^2 = N$，。並將迄點 N 加入至 SE list。

步驟二：選擇目前節點。

判斷 SE list 是否為空集合，若是則跳至步驟五；若否，則從 list 集合中選取第一個節點，令為節點 j。

步驟三：建立暫時的標籤 ϑ_i^1 及 ϑ_i^2 。

1. 搜尋節點 j 的所有上游點 $i \in \Gamma^{-1}(j)$（即 $\forall i \left| (i,j) \in A \right.$）

2. 記錄目前節點 j 為節點 i 的下游點標籤 ϑ_i^2 。

3. 判斷目前節點 j 所屬時段，並根據節線（i,j）在此時段所相對應的最小旅行時間 $t_{i,j}^1$，決定上游點 i 之出發時間 ϑ_i^1，如式（9）所示。

$$\vartheta_i^1 = nt_j - t_{i,j}^1 \quad \text{.....(9)}$$

4. 判斷 ϑ_i^1 和 nt_j 是否屬於同一時段，若是則跳至步驟四；若否則往下一

步驟。

5. 重新計算節線（i,j）之最小旅行時間，則如式（10）所示。

$$\bar{\tau}_{i,j} = nt_j - Q_{\mathrm{i,j}} + \bar{t}^{1}_{i,j} \times \frac{Q_{\mathrm{i,j}} - \vartheta^{1}_{i}}{t^{1}_{i,j}} \quad \text{.....} \quad (10)$$

6. 根據式（11）重新計算上游點 i 之出發時間。

$$\vartheta^{1}_{i} = nt_j - \bar{\tau}_{i,j} \quad \text{.....} \quad (11)$$

步驟四：更新節點標籤。

將目前節點 i 出發時間的永久標籤 nt_{i} 和步驟三節點 i 的暫時標籤 ϑ^{1}_{i} 比較，若暫時標籤 ϑ^{1}_{i} 較永久標籤值 nt_{i} 小，則需更新節點 i 標籤 nt_{i} 及 ns^{2}_{i}；若 ϑ^{1}_{i} 和 nt_{i} 值相同則兩個下游點均需記錄，亦即為同記錄兩條路徑。此外，若 $i \notin SE$，則將節點 i 加入至 SE List。若所有 $i \in \Gamma^{-1}(j)$ 都已計算完畢，則回到步驟二。

步驟五：停止。

演算法結束後，可根據每個節點的永久標籤 nt_{i} 得到最晚可出發時間，再以 ns_{i} 依序尋找下游點，則可得知其相對應之路徑。

3.4.4 演算法之分析

論點 1：對任一節線 $(\mathrm{i_x, j}) \in \mathrm{A}$，$\mathrm{i_x} \in \Gamma^{-1}(j)$ 而言，若最晚到達節點 j 之時間為 nt_j，從節點 $\mathrm{i_x}$ 到節點 j 的可能最短旅行時間為 $t^{1}_{\mathrm{i_x,j}}$，則最晚從節點 $\mathrm{i_x}$ 出發之時間必定為 $nt^{x}_{\mathrm{i_x}} = nt_{\mathrm{j}} - t^{1}_{\mathrm{i_x,j}}$。

證明：

根據 3.2 節之各參數定義，對任一節線 $(\mathrm{i_x, j}) \in \mathrm{A}$，$\mathrm{i_x} \in \Gamma^{-1}(j)$ 而言，其所有可能的旅行時間必定符合式（12）。

$t^{1}_{i_x,j} < t^{2}_{i_x,j} \le t^{3}_{i_x,j} < t^{4}_{i_x,j} \le ... \le t^{k}_{i_x,j} < t^{k+1}_{i_x,j}$，其中 k=1,3,…,(2n-1)…..…….. （12）

由於本演算法主要在於求解在已知到達迄點之時間前提下，最快之最短路徑問題，亦即為利用反推之方式求得最晚可從各節點出發之時間。假設最晚到達節點 j 之時間為 nt_j，若選擇經過節線 $(i_x, j) \in A$，則根據式（13）可知，最晚從節點 i_x 出發之時間 $nt^{x}_{i_x}$ 如式（9）所示，否則必須再更早的時間從節點 i_x 出發才可準時到達節點 j，因此論點 1 成立。

$$\left(nt_j - t^{1}_{i_x,j}\right) > \left(nt_j - t^{s}_{i_x,j}\right) \quad \text{for } s = 2,3,4 \text{ ……..…….. （13）}$$

$$nt^{x}_{i_x} = nt_j - t^{1}_{i_x,j} \text{ ……. （9）}$$

▌

論點 2：對任一節點 i，其下游點集合為 $\Gamma(i) = \{j_1, j_2, ..., j_z\}$ *，必定存在一條節線* $(i, j_x) \in A$ ，$j_x \in \Gamma(i)$ *，使得從節點 i 出發時間最晚為* nt^{x}_{i} 。

證明：

對任一節點 i，其下游點集合 $\Gamma(i) = \{j_1, j_2, ..., j_z\}$。假設存在一條節線 $(i, j_x) \in A$，$1 \le x \le z$，使得從節點 i 出發時間最晚為 nt^{x}_{i}，如式（14）

$$nt^{x}_{i} = nt_{j_x} - t^{1}_{i,j_x} \text{ ……..... （14）}$$

如果假設不成立，則必定存在一條節線 $(i, j_y) \in A$，$1 \le y \le z$，使得從節點 i 出發時間最晚為 nt^{y}_{i}，如式（15）。

$$nt^{y}_{i} = nt_{j_y} - t^{1}_{i,j_y} \ge nt^{x}_{i} \text{ ...……..……..... （15）}$$

則 nt^{y}_{i} 才是最晚從節點 i 出發之時間，和一開始的假設互相矛盾，因此節線 (i, j_y) 不存在，故在時間 nt^{x}_{i} 從節點 i 出發，經過節線 (i, j_x)，才是最快的最短路徑，因此論點 2 成立。但需特別注意的是，當 nt^{y}_{i} 等於 nt^{x}_{i} 情況下，必須同時紀錄兩條路徑。

▌

論點 3：若經過一節線 $(\mathrm{i},\mathrm{j_x})\in \mathrm{A}$ *前後屬於不同時段，只有當* $\frac{\bar{t}^{1}_{i,j_x}}{t^{1}_{i,j_x}} \le \frac{\bar{t}^{1}_{i,j_y}}{t^{1}_{i,j_y}}$ *成立時，節線* $(\mathrm{i},\mathrm{j_x})\in \mathrm{A}$ *仍為最短路徑。*

證明：

相同於論點一之原則，跨時段後所屬時段的旅行時間亦符合式（16）。

$$\bar{t}^{1}_{i,j} < \bar{t}^{2}_{i,j} \le \bar{t}^{3}_{i,j} < \bar{t}^{4}_{i,j} \quad \text{.....} \quad (16)$$

假設在未跨時段前，選擇節線 $(\mathrm{i},\mathrm{j_x})\in \mathrm{A}$，$\mathrm{j_x}\in\Gamma\ (\mathrm{i})$，為最快之最短路徑，根據式（17）所示，跨時段後的旅行時間重新計算如下

$$\bar{\tau}^{1}_{i,j_x} = nt^{\mathrm{x}}_{j} - Q_{\mathrm{i,j}} + \bar{t}^{1}_{i,j_x} \times \frac{t^{1}_{i,j_x} - (nt^{\mathrm{x}}_{j} - Q_{\mathrm{i,j}})}{t^{1}_{i,j_x}} \quad \text{.....} \quad (17)$$

則從節點 i 出發之時間 nt^{x}_{i} 修正為式（18）所示

$$\begin{aligned} nt^{\mathrm{x}}_{i} &= nt^{\mathrm{x}}_{j} - \bar{\tau}^{1}_{i,j_x} \\ &= nt^{\mathrm{x}}_{j} - [nt^{\mathrm{x}}_{j} - Q_{\mathrm{i,j}} + \bar{t}^{1}_{i,j_x} \times \frac{t^{1}_{i,j_x} - (nt^{\mathrm{x}}_{j} - Q_{\mathrm{i,j}})}{t^{1}_{i,j_x}}] \quad \text{.....} \quad (18) \\ &= Q_{\mathrm{i,j}} - \bar{t}^{1}_{i,j_x} \times \frac{t^{1}_{i,j_x} - (nt^{\mathrm{x}}_{j} - Q_{\mathrm{i,j}})}{t^{1}_{i,j_x}} \end{aligned}$$

若假設跨時段後，不存在一條節線 $(\mathrm{i},\mathrm{j_y})\in \mathrm{A}$，$\mathrm{j_y}\in\Gamma\ (\mathrm{i})$，使得從節點 i 最晚出發時間為 nt^{y}_{i}，如式（19），則必須滿足式（20）才構成條件。

$$\begin{aligned} nt^{\mathrm{y}}_{i} &= nt^{\mathrm{y}}_{j} - \bar{\tau}^{1}_{i,j_y} \\ &= nt^{\mathrm{y}}_{j} - [nt^{\mathrm{y}}_{j} - Q_{\mathrm{i,j}} + \bar{t}^{1}_{i,j_y} \times \frac{t^{1}_{i,j_y} - (nt^{\mathrm{y}}_{j} - Q_{\mathrm{i,j}})}{t^{1}_{i,j_y}}] \quad \text{.....} \quad (18) \\ &= Q_{\mathrm{i,j}} - \bar{t}^{1}_{i,j_y} \times \frac{t^{1}_{i,j_y} - (nt^{\mathrm{y}}_{j} - Q_{\mathrm{i,j}})}{t^{1}_{i,j_y}} \end{aligned}$$

$$nt_i^{\mathrm{x}} = Q_{\mathrm{i,j}} - \bar{t}_{i,j_{\mathrm{x}}}^{1} \times \frac{t_{i,j_{\mathrm{x}}}^{1} - (nt_j^{\mathrm{x}} - Q_{\mathrm{i,j}})}{t_{i,j_{\mathrm{x}}}^{1}}$$
$$\geq nt_i^{\mathrm{y}} = Q_{\mathrm{i,j}} - \bar{t}_{i,j_{\mathrm{y}}}^{1} \times \frac{t_{i,\mathrm{j_y}}^{1} - (nt_j^{\mathrm{y}} - Q_{\mathrm{i,j}})}{t_{i,j_{\mathrm{y}}}^{1}} \quad \text{……（20）}$$

經簡化後為式（21）。

$$\bar{t}_{i,j_{\mathrm{x}}}^{1} \times \frac{Q_{\mathrm{i,j}} - (nt_j^{\mathrm{x}} - t_{i,j_{\mathrm{x}}}^{1})}{t_{i,j_{\mathrm{x}}}^{1}} \leq \bar{t}_{i,j_{\mathrm{y}}}^{1} \times \frac{Q_{\mathrm{i,j}} - (nt_j^{\mathrm{y}} - t_{i,\mathrm{j_y}}^{1})}{t_{i,j_{\mathrm{y}}}^{1}} \quad \text{……（21）}$$

由於原本未跨時段前，節線$(\mathrm{i,j_x}) \in \mathrm{A}$最短路徑，因此

$$nt_i^{\mathrm{x}} = nt_j^{\mathrm{x}} - \bar{\tau}_{i,j_{\mathrm{x}}}^{1} \geq nt_i^{\mathrm{y}} = nt_j^{\mathrm{y}} - \bar{\tau}_{i,j_{\mathrm{y}}}^{1} \quad \text{……（22）}$$

根據式(21)及式(22)可知，當式(23)成立時，則跨時段之後節線$(\mathrm{i,j_x}) \in \mathrm{A}$必定仍為最快之最短路徑，否則有可能會被其他路徑取代，因此論點 3 成立。

$$\frac{\bar{t}_{i,j_{\mathrm{x}}}^{1}}{t_{i,j_{\mathrm{x}}}^{1}} \leq \frac{\bar{t}_{i,j_{\mathrm{y}}}^{1}}{t_{i,j_{\mathrm{y}}}^{1}} \quad \text{……（23）}$$

論點 4：A1 演算法之時間複雜度為$\mathrm{O}(v^2)$*，其中 v 為網路節點之個數。*

證明：

本演算法從迄點開始計算，計算時須檢查其所有上游點，假設最糟情況為完整路網（complete graph），則迄點的上游點最多可能包含 v-1 個節點，因此須將這些節點將入至 SE list 中，亦即為第一個 Iteration 結束後 SE list 有 v-1 個節點。

接著計算 SE list 中每個節點時，同樣地在最糟情況下，任一節點的所有上游點最多為 v-1 個，則最多可能會有$(v-1)^2$個節點插入至 SE list。故得知時間複雜度為$\mathrm{O}((v-1)^2) = \mathrm{O}(v^2)$。

3.4.5 演算法之範例

本小節將以前述之演算法求解一簡單範例。首先，建立一具有方向性並包含四個節點之小型網路，如圖 3-2，其相關的網路資料見表 3-5。

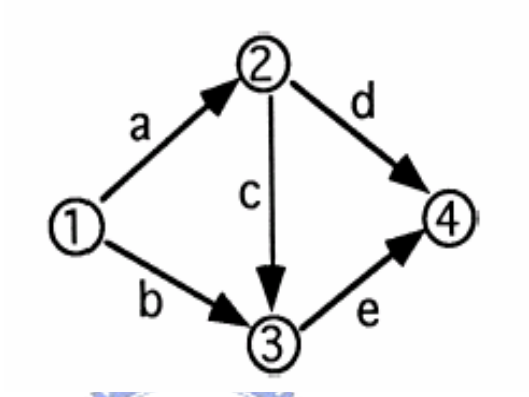


圖 3-2　Miller-Hooks and Mahmassani 演算法範例說明之網路圖[6]

表 3-5　範例網路之相關資料

節線	節點 i	節點 j	時段	旅行時間 1（機率）	旅行時間 2（機率）
a	1	2	上午尖峰	15～20（0.2）	20~25（0.8）
			下午尖峰	13～18（0.4）	18～23（0.6）
			離峰時段	10～15（0.1）	15～20（0.9）
節線	節點 i	節點 j	時段	旅行時間 1（機率）	旅行時間 2（機率）
b	1	3	上午尖峰	20～25（0.3）	25~30（0.7）
			下午尖峰	12～17（0.4）	17～22（0.6）
			離峰時段	7～12（0.8）	12～17（0.2）

c	2	3	上午尖峰	18～23（0.9）	23~28（0.1）
			下午尖峰	25～30（0.7）	30～35（0.3）
			離峰時段	12～17（0.1）	17～22（0.9）
d	2	4	上午尖峰	11～16（0.7）	16~21（0.3）
			下午尖峰	16～21（0.4）	21～26（0.6）
			離峰時段	8～13（0.5）	13～18（0.5）
e	3	4	上午尖峰	24～29（0.8）	29~34（0.2）
			下午尖峰	15～20（0.6）	20～25（0.4）
			離峰時段	11～16（0.1）	16～21（0.9）

求解若欲在早上九點十分前到達節點 4，可最晚從節點 1 出發之時間以及相對應的路徑。此外，以下為方便計算均將時間顯示轉化成以分鐘為單位表示。計算過程如下：

Iteration 1

步驟一：

V = {1,2,3,4}

A = {a,b,c,d,e}

定義尖離峰時間範圍

上午尖峰時段 = AM 7:30 ～ AM 9:00 = 450 ～ 540

下午尖峰時段 = PM 5:00 ～ PM 7:30 = 1020 ～ 1170

離峰時段 = 一天中尖峰時段以外的時間

迄點 N = 4

預計到達迄點時間 nt_4 = AM 9:10 = 550

預計到達其他節點的時間先設定為一極小值，nt_i = -1 for i = {1,2,3}

令迄點 4 的下游點為自己，故 $ns_4^2 = 4$

將迄點加入至 SE list 中，SE = {4}

步驟二： 從 SE list 取出第一個節點，設為 j = 4，則 SE = {}

步驟三： 對節點 4 而言，其上游點 $i \in \{2,3\}$，依序計算所有上游點的最晚出發時間，因此先選擇節點 2，i = 2，其下游點暫時標籤為 $\vartheta_2^2 = 4$。

首先判斷 nt_4 所屬時段，540 < nt_4 = 550 < 1020，故屬於離峰時段

節點 2 需經過節線 d 才能到達節點 4，因此根據 nt_4 所屬時段尋找經過節線 d 所需花費的最小旅行時間 $t_{2,4}^1$，$\vartheta_2^1 = nt_4 - t_{2,4}^1 = 550 - 8 = 542$

判斷 nt_2 所屬時段，540 < nt_2 = 542 < 1020，故屬於離峰時段

nt_4 和 nt_2 屬於同時段，因此不需改變旅行時間。

i = 3

$nt_3 = nt_4 - t_{3,4}^1 = 550 - 11 = 539$

450 < nt_3 = 530 < 540，故屬於上午尖峰時段

nt_4 和 nt_3 屬於不同時段，因此需重新計算旅行時間，此時分段的時間點 $Q_{3,4} = 540$

$$\bar{t}_{3,4}^1 = 550 - 540 + 24 \times \frac{540 - 539}{11} = 12.18$$

$$\bar{t}_{3,4}^1 = 550 - 12.18 = 537.82$$

步驟四：

比較出發時間的暫時標記和永久標記，若前者值較大，則需更新至永久標

籤，並將該節點加入至 SE list

$nt_2 = \max\{\vartheta_2^1 = 542, nt_2 = -1\} = 542$ ， $ns_2 = \vartheta_2^2 = 4$ ，故 SE = {2}

$nt_3 = \max\{\vartheta_3^1 = 537.82, nt_3 = -1\} = 537.82$ ， $ns_3 = \vartheta_3^2 = 4$ ，故 SE = {2,3}

依此類推，最後測試結果之路徑為節點 1→2→4，在各節點可最晚出發時間如表 3-6。

表 3-6　最快到達之最短路徑演算法之範例結果

節點	最晚出發時間	路徑
1	AM 8:48（528）	1→2→4
2	AM 9:02（542）	2→4
3	AM 8:57.82(537.82)	3→4

3.5　可準時抵達目的地之最早出發時間之分析

以本研究時間範圍之概念，最短路徑之旅行時間並非單一值，而是在某時間範圍內都可能發生，因此除了提供最快之最短路徑外，尚需提供最短路徑之可能花費最久之旅行時間，對旅行者而言較能掌握其遲到之風險。以下本節首先定義欲求解之問題，再說明求解方法和演算法架構，最後以簡單範例說明。

3.5.1　問題定義

此演算法主要在一個具有方向性之網路，已知欲到達迄點 N 之到達時間，且已得到最短路徑（見 3.4 節），求解此最短路徑上任一節點可準時到達迄點 N 之可能最早需出發時間之問題，亦即求取最短路徑之旅行時間上界，以下簡稱為「A2 演算法」。

3.5.2 求解方法

此演算法和 3.4 節求解最晚可出發時間演算法觀念相同，除了初始值之設定以及節線（i,j）之旅行時間設定值不同之外，其他步驟均一致。

3.5.3 求解步驟

步驟一：初始化並建立 Scan Eligible List。

設定上午尖峰、下午尖峰以及離峰時段。令欲到達迄點 N 之時間為 nt_N ，nt_i = M，其中 i $\subseteq$ {shortest path}，M 為一極大數。根據 3.4 節 A1 演算法所求得最短路徑，記錄此路徑上的各節點的上游點標籤 ns_i^1，其中起點 O 的 $ns_O^1 = O$，並且將各節點加入至 SE list，list 之順序從迄點開始，依序加入，最後為起點。因此演算法之計算從迄點 N 開始。

步驟二：選擇目前節點。

判斷 SE list 是否為空集合，若是則跳至步驟五；若否，則從 list 集合中選取第一個節點，為節點 j。

步驟三：建立暫時標籤 ϑ_i^1。

1. 令 $i = ns_j^1$

2. 判斷目前節點 j 所屬時段，並根據節線（i,j）在此時段所相對應的最大旅行時間 $t_{i,j}^4$，決定上游點 i 之出發時間 ϑ_i^1，如式（24）所示。

$$\vartheta_i^1 = nt_j - t_{i,j} \quad \text{.....(24)}$$

3. 判斷 η_i^1 和 nt_j 是否屬於同時段，若是則跳至步驟四；若否則往下一步驟。

4. 重新計算節線（i,j）之最大旅行時間，則如式（25）

$$\bar{\tau}_{i,j} = nt_j - Q_{\mathrm{i,j}} + \bar{t}^{4}_{i,j} \times \frac{Q_{\mathrm{i,j}} - \vartheta^{1}_{i}}{t^{4}_{i,j}} \quad \text{.....} \quad (25)$$

5. 根據式（26）重新計算上游點 i 之出發時間。

6. $\vartheta^{1}_{i} = \mathrm{nt_j} - \bar{\tau}_{\mathrm{i,j}}$ .....（26）

步驟四：更新節點標籤。

將目前節點 i 出發時間的永久標籤 nt_{i} 和步驟三節點 i 的暫時標籤 ϑ^{1}_{i} 比較，若暫時標籤 ϑ^{1}_{i} 較永久標籤值 nt_{i} 小，則需更新節點 i 標籤 nt_{i} 。接著，回到步驟二。

步驟五：停止。

演算法結束後，可根據每個節點的永久標籤 nt_{i} 得到最短路徑之最早應出發時間。

3.5.4 演算法之分析

論點 5：對任一節線 $(\mathrm{i_x, j}) \in \mathrm{A}$ ，$\mathrm{i_x} \in \Gamma^{-1}(j)$ *而言，若預計到達節點 j 之時間為* nt_j *，從節點* $\mathrm{i_x}$ *至節點 j 的最大旅行時間為* $t^{\mathrm{k+1}}_{\mathrm{i_x,j}}$ *，其中 k=1,3,..,(2n-1)，則在* nt^{x}_{i} *前從節點* $\mathrm{i_x}$ *出發必定能準時到達節點 j，其中* $nt^{x}_{\mathrm{i}} = nt_{\mathrm{j}} - t^{\mathrm{k+1}}_{\mathrm{i_x,j}}$ 。

證明：

根據論點 1 以及式（11）所示，可知對*任一節線* $(\mathrm{i, j}) \in \mathrm{A}$ 而言，所需花費最久的旅行時間為 $t^{\mathrm{k+1}}_{\mathrm{i,j}}$，因此當已知到達節點 j 的最早時間為 nt_j 時，則選擇節線 $(\mathrm{i, j}) \in \mathrm{A}$ ，*最早從節點 i 出發之時間必定為* $nt_{\mathrm{i}} = nt_{\mathrm{j}} - t^{\mathrm{k+1}}_{\mathrm{i,j}}$ 。

▌

論點 6：最短路徑的旅行時間最大值未必小於網路上所有路徑旅行時間最大值。

證明：

假設節線$(i, j_x) \in A$，$j_x \in \Gamma(i)$，包含在最快之最短路徑中，使得 $nt_i^x \geq nt_i^y = nt_{j_y} - t_{i,j_y}^1$ where $x \neq y, j_y \in \Gamma(i)$，然而當每個節線的每個時間範圍間距不同之情形下，無法保證$nt_i^x \geq nt_i^y = nt_{j_y} - t_{i,j_y}^4$；因此，唯有在所有節線的任一時間範圍間距均相同之下，假設其間距為 R 且$t_{i,j}^2 = t_{i,j}^3$，則$t_{i,j}^4 = t_{i,j}^1 + 2R$，故可得式（27），亦即最短路徑的旅行時間最大值為所有路徑旅行時間最大值中的最小值。

$$\begin{aligned} nt_i^x &= nt_{j_x} - t_{i,j_x}^4 = nt_{j_x} - (t_{i,j_x}^1 + 2R) \\ &\geq nt_i^y = nt_{j_y} - t_{i,j_y}^4 = nt_{j_y} - (t_{i,j_y}^1 + 2R) \end{aligned} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (27)$$

∎

論點 7：A2 演算法之時間複雜度為$O(v)$*，其中 v 為預設路徑中所經節點之個數。*

證明：

在本演算法中，任一節點只有被加入 SE list 一次的機會，一旦計算完被移除後，就不可能再加到 SE list。在初始化時，已經決定行走之路徑，並加入至 SE list，因此 SE list 之節點個數為 v。

計算任一節點時，必須檢查其所有上游點，以計算上游點的最晚到達時間；由於預設只有一條路徑，因此每一節點僅有一個上游點，故時間複雜度$O(v)$。

∎

3.5.5 演算法之範例

測試範例網路同 3.4.5 節之網路範例，並利用 3.4.5 節求得的最快之最短路徑作為測試之路徑，亦即為從節點 1 出發，經過節點 2，最後到達節點 4。

由上可知，本測試範例欲求解之問題為選擇節點 1－＞2－＞4 之路徑，最早應從節點 1 出發之時間，可保證在早上九點十分前準時到達節點，絕不會遲到。計算過程如下，此外為方便計算均將時間顯示轉化成以分鐘為單位表示：

Iteration 1

步驟一：

V = {1,2,4}

A = {a,d}

定義尖離峰時間範圍

上午尖峰時段 = AM 7:30 ～ AM 9:00 = 450 ～ 540

下午尖峰時段 = PM 5:00 ～ PM 7:30 = 1020 ～ 1170

離峰時段 = 一天中尖峰時段以外的時間

迄點 N = 4

預計到達迄點時間 nt_N = AM 9:10 = 550

先設定預計到達其他節點的最早時間為一極大值

nt_i = M = 9999 for i = {1,2}

設定預先選擇路線上所有節點的上游點，因此 $ns_1^1 = 1, ns_2^1 = 1, ns_4^1 = 2$

將預先選擇路線上的所有節點加入至 SE list 中，因此 SE = {4,2,1}

步驟二： 從 SE list 中取出第一個節點，設為 j = 4，則 SE = {2,1}

步驟三：在選擇的路線中，對節點 4 而言，其上游點 $i = ns_4^1 = 2$

首先判斷 nt_4 所屬時段，540 < nt_4 = 550 < 1020，故屬於離峰時段

節點 2 需經過節線 d 才能到達節點 4，因此根據 nt_4 所屬時段尋找經過

節線 d 所需花費最久的旅行時間，$\vartheta_2^1 = nt_4 - t_{2,4}^4 = 550 - 18 = 532$

判斷 ϑ_2^1 所屬時段，$450 < \vartheta_2^1 = 532 < 540$ ，故屬於上午尖峰時段

nt_4 和 ϑ_2^1 屬於不同時段，需重新計算旅行時間

此時分段的時間點 $Q_{2,4} = 540$

$$\bar{t}_{2,4}^4 = 550 - 540 + 21 \times \frac{540 - 532}{18} = 19.33$$

$$\vartheta_2^1 = 550 - 19.33 = 530.67$$

步驟四：比較出發時間的暫時標記和永久標記，若前者值較大，則需更新至永久標籤，$nt_2 = \max\{\vartheta_2^1 = 530.67, nt_2 = 9999\} = 530.67$

Iteration 2

步驟二： $j = 2, SE = \{1\}$

步驟三： $i = 1$

$nt_2 = 530.67 < 540$ 且 $nt_2 = 530.67 < 450$ ，故屬於上午尖峰時段

$\vartheta_1^1 = nt_2 - t_{1,2}^4 = 530.67 - 25 = 505.67$

$450 < \vartheta_1^1 = 505.67 < 540$ ，故屬於上午尖峰時段

nt_2 和 ϑ_1^1 屬於同時段，無須重新計算旅行時間

$nt_1 = \min\{505.67, 9999\} = 505.67$

Iteration 3

步驟二： $j = 1, SE = \{\}$

步驟四： SE list 為空集合，故停止演算。

選擇最短路徑為節點 1→2→4，將各節點最早應出發時間之結果與表 3-6 相比，如表 3-7。

表 3-7　最慢到達之最短路徑演算法之範例結果

節點	最早出發時間	最晚出發時間	路徑
1	AM 8:25.67（505.67）	AM 8:48（528）	1→2→4
2	AM 8:50.67（530.67）	AM 9:02（542）	2→4

3.6　各種到達時間之風險演算法

3.4 節以及 3.5 節所提出之演算法雖然能提供最短路徑之旅行時間誤差範圍，但均未考慮發生機率之問題；然而實際生活中，不論任一條路線其旅行時間誤差值可能極大，若能提供具體的旅行時間以及發生機率，對旅行者而言將有相當大的助益。因此本節將加入風險之考量，提出第三個演算法，主要在於探討在某一時間點從起點出發，可準時於預計時間範圍內到達的機率風險，藉由機率之參考值可讓旅行者依據其遲到之風險自行選擇出發時間。

以下本節將先定義欲求解之問題，再說明求解方法和演算法架構，最後以一簡單範例說明。

3.6.1　問題定義

此演算法主要在一個具有方向性之網路，且已知欲出發之時間以及欲選擇之路徑，探討到達迄點 N 的所有可能時間範圍以及其相對應的發生機率問題，以下簡稱為「A3 演算法」。

3.6.2　求解方法

由於利用反推之演算法無法解決時間範圍所相對應之機率問題，亦即為若設定預計到達迄點之時間範圍，往回推之計算結果為從出發點會有各種出發時間範圍，然而每個出發時間範圍到達迄點的時間範圍之機率應互相獨立，無法合併計算，故此演算法採用正推之方式。此外，標記設定法之時間複雜度優於標記修正法，因此本演算法主要修正自標記設定法[8]，並加入 Miller-Hooks and

Mahmassani [6]第一個演算法之機率概念。求解過程中，對每一節點而言，均具有一個永久標籤向量，其中每一個標籤均需紀錄兩項資訊，一項提供從該節點可能準時到達迄點的出發時間範圍，預設值為零，另一項則紀錄從該節點到達迄點的可能發生機率。此外，需設定 SE list 用來記錄永久標籤被更新的節點集合，由於本演算法採用反推之計算方式，因此一開始需先將迄點加入至 SE list。

基於此演算法須同時考慮各種旅行時間之情況，因此為避免造成時間複雜度呈指數型成長，故僅針對單一路徑做探討，且須先得知欲從起點出發之時間，故計算採用正推之方式，即從起點作為起始點，依設定之路徑順序往後推算。演算法中的每個步驟均需計算目前節點 i 之下游點 j 的每一可能出發時間範圍和相對應之機率，設為暫時標籤，並且和節點 j 目前已存在的永久標籤向量比較，若有重複的出發時間範圍，則將其相對應之發生機率和暫時標籤加總，並更新至目前的標籤中，否則即加入至節點 j 永久標籤向量；如此反覆計算直到 SE list 為空集合即可求解。

不同於 Miller-Hooks and Mahmassani [6]之概念在於本演算法不僅提供最短路徑之最小旅行時間可能性，並探討此路徑上之所有可能旅行時間以及相對應之發生可能性；此外，本演算法不侷限於針對最短路徑作討論，而可隨著起始設定路徑之不同而加以計算，故亦能探討所有路徑之各種可能性。

3.6.3 求解步驟

步驟一：初始化並建立 Scan Eligible List。

設定上午尖峰、下午尖峰以及離峰時段之範圍，令欲從起點 O 出發之時間為 nt_{O}^{1}，且 $nt_{\mathrm{O}}^{1} = nt_{\mathrm{O}}^{2}$，其發生機率 $p_{\mathrm{O}} = 1$。從迄點 N 開始計算，並加入至 SE list。

根據欲選擇之路徑，記錄該路徑中各節點之下游點標籤 ns_i^2，其中迄點 N 的下游點 $\mathrm{ns}_{\mathrm{N}}^{2} = \mathrm{N}$，將欲選擇路徑之各節點加入至 SE list，list 之順序從

起點 O 開始依序加入，最後為迄點 N，因此演算法之計算從起點 O 開始。

步驟二：選擇目前節點。

判斷 SE list 是否為空集合，若是則跳至步驟五；若否，則從 list 集合中選取第一個節點，為節點 i。

步驟三：建立暫時標籤$[\gamma_i^1,\gamma_i^2]$及ρ_{j}

1. 搜尋節點 i 在設定路徑中的下游點$\mathrm{j}=\mathrm{ns}_i^2$

2. 判斷目前節點 i 所屬時段，並根據節線（i,j）於此時段所相對應的所旅行時間範圍$[t_{i,j}^{\mathrm{k}},t_{i,j}^{\mathrm{k+1}}]$和發生機率$p_{i,j}^{\mathrm{s}}$，在此為簡化說明設定 k=1,3，決定下游點 j 之暫時時間範圍$[\gamma_i^1,\gamma_i^2]$和相對應的機率ρ_{j}，如式（6）所示。

$\gamma_{\mathrm{j}}^1=nt_{\mathrm{i}}^1+t_{i,j}^1$ 且 $\gamma_{\mathrm{j}}^2=nt_{\mathrm{i}}^2+t_{i,j}^2$，$\rho_{\mathrm{j}}=p_{\mathrm{i}}\times p_{i,j}^1$

到 $\gamma_{\mathrm{j}}^1=nt_{\mathrm{i}}^1+t_{i,j}^3$ 且 $\gamma_{\mathrm{j}}^2=nt_{\mathrm{i}}^2+t_{i,j}^4$，$\rho_{\mathrm{j}}=p_{\mathrm{i}}\times p_{i,j}^2$ ..... （6）

3. 判斷nt_{i}^2和γ_{j}^2是否屬於同時段，若是則跳至步驟四；若否則往下一步驟。

4. 判斷γ_{j}^1和γ_{j}^2是否屬於同時段。

i. 若是，則重新計算節線（i,j）之旅行時間範圍，如式（28）所示。

$$\bar{\tau}_{i,j}^1=Q_{\mathrm{i,j}}-nt_{\mathrm{i}}^1+\bar{t}_{i,j}^1\times\frac{t_{i,j}^1-(Q_{\mathrm{i,j}}-nt_{\mathrm{i}}^1)}{t_{i,j}^1}$$

$$\bar{\tau}_{i,j}^2=Q_{\mathrm{i,j}}-nt_j^2+\bar{t}_{i,j}^4\times\frac{t_{i,j}^2-(Q_{\mathrm{i,j}}-nt_j^2)}{t_{i,j}^2} \quad \text{..... （28）}$$

根據式（29）重新計算下游點 j 之時間範圍，其相對應之機率值不變。

$\pi_{\mathrm{j}}^{1} = nt_{\mathrm{i}}^{1} + \bar{\tau}_{i,j}^{1}$ ， $\pi_{\mathrm{j}}^{2} = nt_{\mathrm{i}}^{2} + \bar{\tau}_{i,j}^{2}$ ……（29）

ii. 若否，則將下游點 j 之時間範圍和機率分為兩部分記錄。

第一部份：下游點 j 之時間範圍為式（6）中之 π_{j}^{1}～$\mathrm{Q}_{\mathrm{i},i}$，其相對應之機率值如式（30）。

$$\rho_{\mathrm{j}}^{1} = \rho_{\mathrm{j}} \times \frac{Q_{\mathrm{i,j}} - \gamma_{\mathrm{j}}^{1}}{nt_{\mathrm{j}}^{2} - \gamma_{\mathrm{j}}^{1}} \quad \text{……（30）}$$

第二部份：根據式（28）及式（30）分別重新計算下游點 j 之時間範圍 $\bar{\tau}_{i,\mathrm{j}}^{2}$ 和 γ_{j}^{2}，則下游點 j 之時間範圍為 $\mathrm{Q}_{\mathrm{i},i}$～式（29）中之 γ_{j}^{2}，其相對應之機率值為式（31）。

ρ_{i}^{2} ＝式（6）中之 ρ_{j} －式（30）ρ_{j}^{1} ……（31）

步驟四：更新節點標籤。

判斷目前節點 j 之時間範圍標籤集合是否和步驟三節點 j 時間範圍之暫時標籤 $[\gamma_i^1, \gamma_i^2]$ 有重複，若否則將其時間範圍和機率加入至節點 i 標籤集合；若有重複者則將其原本之機率值 $p_{i,j}^{s}$ 加上步驟三中 j 機率值之暫時標籤 ρ_{j}，並更新到節點 j 之機率標籤。接著，回到步驟二。

步驟五：停止。

演算法結束後，可根據每個節點時間範圍以及機率的永久標籤得到於預計出發時間，到達預定路徑上各節點之可能時間範圍，以及相對應之發生機率。

3.6.4 演算法之分析

論點 8：A3 演算法之時間複雜度為 $\mathrm{O}(\alpha^{\beta} I^{v-1-\beta})$ *，其中* α *為跨時段後每一節點需紀錄的時間範圍個數，* β *為行經路徑過程中跨時段之次數，* I *為每*

一節線的時間範圍個數，v 為預設路徑中所經節點之個數。

證明：

在本演算法初始化時，需將預設路徑中的所有節點加入至 SE list，任一節點只有被加入 SE list 一次的機會，一旦計算完被移除後，就不可能再加到 SE list，因此假設該路徑的經過節點數為 v，節線數為 v−1，則 SE list 之節點個數最多為 v。

計算SE list任一節點時，未跨越時段之情況下，每一節線的時間範圍個數為I，整條路徑須計算的節線數為v-1，因此至多須計算 I^{v-1} 次，如圖 3-3 所示，節點X_1紀錄的到達時間範圍個數為 1，從節點X_1至X_2的節線時間範圍個數為 I，因此到達X_2的時間範圍增為I個，至節點X_3則增為I^2個，依此類推。

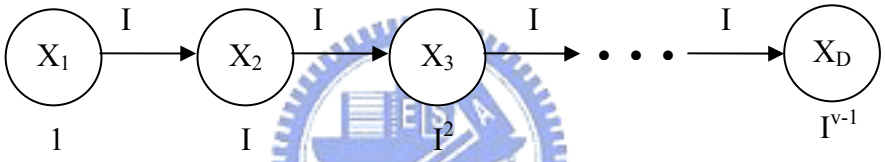


圖 3-3　A3 演算法未跨越時段之網路

若需跨越時段，經過分割後最糟情況，每一節線的時間範圍個數會由原本的I個增加為α個，若從起點至迄點總共跨越時段次數為β，則跨越時段的節線須紀錄之時間範圍個數為 α^{β} 個，未跨越時段須記錄個數減為 $I^{v-1-\beta}$ ，如圖 3-4 所示，節點X_1紀錄的到達時間範圍個數為 1，跨越節點X_1至X_2的節線屬於不同時段，因此該節線時間範圍個數為α，因此到達X_2的時間範圍增為α個，從節點X_2至X_3未跨越時段，因此節點X_3需記錄到達時間範圍增為 $I\alpha$ 個，依此類推，故得知時間複雜度為 $O(\alpha^{\beta} I^{v-1-\beta})$ 。

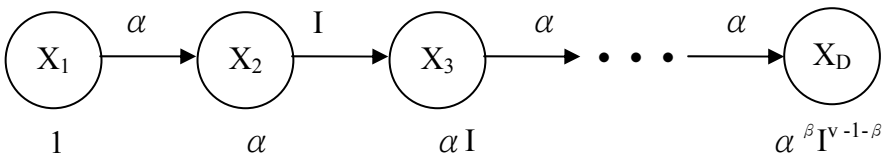


圖 3-4　A3 演算法含跨越時段之網路

根據論點 8 得知此演算法的時間複雜度過於龐大，因此為減少運算次數，並且避免不必要的資訊，可先過濾過小的機率值，即不予紀錄機率值太小的時間範圍；此外，若有重複的時間範圍，則將其可能發生的機率值合併，可有效減少每個節點所需紀錄的時間範圍個數，如此可避免隨著網路節點數的增加，造成最後迄點所需紀錄的時間範圍個數顯著的成長。

3.6.5　演算法之範例

測試範例網路同 3.4.4 節之網路範例，並以 3.4.4 節所求得之最短路徑 1－>2－>4 作為預設之選擇路徑。欲求解早上八點三十分從起點 1 出發，則到達迄點 4 之各種旅行時間範圍以及相對應之機率問題。以下為方便計算均將時間顯示轉化成以分鐘為單位表示。

Iteration 1

步驟一：

V = {1,2,4}

A = {a,d}

定義尖離峰時間範圍

上午尖峰時段 = AM 7:30 ～ AM 9:00 = 450 ～ 540

下午尖峰時段 = PM 5:00 ～ PM 7:30 = 1020 ～ 1170

離峰時段 =　一天中尖峰時段以外的時間

起點 O = 1

預計出發時間 $nt_1^1 = nt_1^2$ =AM 8:30 = 510

預計出發時間的機率值 $p_1 = 1$

記錄預設路線中各節點之下游點標籤，所以 $ns_1^2 = 2$，$ns_2^2 = 4$，$ns_4^2 = 4$

將預設路線的所有節點，依序加入至 SE list 中，故 SE = {1,2,4}

步驟二：從 SE list 中取出第一個節點，令為 i = 1, SE = {2,4}

步驟三：在預設的路線中，對節點 1 而言，其下游點 $j = ns_1^2 = 2$

判斷 nt_1^1 所屬時段，$450 < nt_1^1 = 510 < 540$，因此屬於上午尖峰時段

節點 1 需經過節線 a 才能到達節點 2，故根據 nt_1^1 所屬時段尋找經過節線 a 所需花費的旅行時間範圍 $[t_{i,j}^1, t_{i,j}^2]$，以及所相對應的機率值 $p_{1,2}^1$，

$\gamma_2^1[0] = nt_1^1 + t_{1,2}^1 = 510 + 15 = 525$

$\gamma_2^2[0] = nt_1^2 + t_{1,2}^2 = 510 + 20 = 530$

$\rho_2[0] = p_1 \times p_{1,2}^1 = 1 \times 0.2 = 0.2$

$450 < \gamma_2^2[0] = 530 < 540$，因此屬於上午尖峰時段

nt_1^2 和 $\gamma_2^2[0]$ 屬於同時段，故不需重新計算旅行時間

根據 nt_1^1 所屬時段尋找經過節線 a 所需花費的旅行時間範圍尚有 $[t_{i,j}^3, t_{i,j}^4]$，以及所相對應的機率值 $p_{1,2}^2$，

$\gamma_2^1[1] = nt_1^1 + t_{1,2}^1 = 510 + 20 = 530$

$\gamma_2^2[1] = nt_1^2 + t_{1,2}^2 = 510 + 25 = 535$

$\rho_2[1] = p_1 \times p_{1,2}^1 = 1 \times 0.8 = 0.8$

$450 < \gamma_2^2[1] = 535 < 540$，因此屬於上午尖峰時段

nt_1^2 和 $\gamma_2^2[1]$ 屬於同時段，故不需重新計算旅行時間

步驟四：更新節點標籤

得知 $nt_2^1[0]=\gamma_2^1[0]=525$，$nt_2^2[0]=\gamma_2^2[0]=530$，$p_2[0]=\rho_2[0]=0.2$；

$nt_2^1[1]=\gamma_2^1[1]=530$，$nt_2^2[1]=\gamma_2^2[1]=535$，$p_2[1]=\rho_2[1]=0.8$

Iteration 2

步驟二：i = 2, SE = {4}

步驟三&四：j = 4

$450 < nt_2^2[0]=530<540$ ，因此屬於上午尖峰時段

$\gamma_4^1[0]=nt_2^1[0]+t_{2,4}^1=525+11=536$

$\gamma_4^2[0]=nt_2^2[0]+t_{2,4}^2=530+16=546$

$\rho_4[0]=p_2[0]\times p_{1,2}^1=0.2\times 0.7=0.14$

$540 < \gamma_4^2[0]=546<1020$，因此屬於離峰時段

nt_2^2 和 $\gamma_4^2[0]$ 屬於不同時段，且 $\gamma_4^1[0]$ 和 $\gamma_4^2[0]$ 屬於不同時段

所以分為兩部分討論，並重新計算旅行時間

第一部份：

上午尖峰時段和離峰時段的分界時間點為 $Q_{2,4}=540$

因此 $\gamma_4^1[0]=536$，$\gamma_4^2[0]=540$

$$\rho_4[0]=0.14\times\frac{540-536}{546-536}=0.056$$

第二部份：

$$\bar{\tau}_{i,j}^2=540-530+18\times\frac{546-540}{16}=16.75$$

$$\gamma_4^2[1]=530+16.75=546.75$$

$\gamma_4^1[1]=540$，$\gamma_4^2[1]=546.75$

$\rho_4[1]=0.14-0.056=0.084$

$\gamma_4^1[2]=nt_2^1[0]+t_{2,4}^3=525+16=541$

$\gamma_4^2[2]=nt_2^2[0]+t_{2,4}^4=530+21=551$

$\rho_4[2]=p_2[0]\times p_{1,2}^1=0.2\times 0.3=0.06$

$540<\gamma_4^2[2]=551<1020$，因此屬於離峰時段

$\gamma_2^2[0]$和$\gamma_4^2[2]$ 屬於不同時段，所以需要重新計算旅行時間

其分界時間點為$Q_{2,4}=540$

nt_4^1和nt_4^2屬於同時段

$$\bar{\tau}_{i,j}^1=540-530+8\times\frac{541-540}{16}=10.5$$

$$\bar{\tau}_{i,j}^2=540-530+18\times\frac{551-540}{21}=19.429$$

得知$\gamma_4^1[2]=530+10.5=540.5$ ，$\gamma_4^2[2]=530+19.29=549.429$

$\rho_4[2]=0.06$

依此類推，最後測試結果如表 3-7。

表 3-7　各種到達時間風險之測試結果

路徑	到達時間範圍	相對應機率
1—2—4	536---540	0.056
	540---546.75	0.084
	540.5---549.429	0.06

	540.727---552.375	0.56
	543---553.714	0.24

假設在每個時間範圍內的每個時間點之機率為均值分配，將結果以每 15 分鐘為範圍表示，則經過合併簡化後，最後結果可顯示如表 3-8。

表 3-8　合併到達時間風險之測試結果

路徑	到達時段	相對應機率
1—2—4	536---540	0.056
	540---553.714	0.944

第四章　測試與分析

A1 演算法和 A2 演算法已分別在 3.3.4 節和 3.4.4 節驗證其正確性，故不再作測試。因此本章主要為測試 A3 演算法，採用虛擬路網驗證分析。

4.1　各種到達時間之風險演算法測試

由於 A3 演算法的計算隨著網路大小、網路資料不同及出發時間之設定等各種情形而異，因此本研究採用實測之方式；然而鑒於實際路網之旅行時間資料取得不易，故利用虛擬路網資料測試。本研究採用 Microsoft Visual C++ 6.0 編譯器撰寫求解演算法，測試環境配備為 Intel Pentium 4 CPU 2.4GHz，512MB RAM，以及 Microsoft Windows XP Professional Version 2002 之個人電腦上執行。

本測試目的主要為探討於演算過程中，跨時段次數對於演算次數之影響，以及不同時間範圍間距對於演算次數之影響，最後觀察各種到達時間機率的分佈結果。因此本測試路網之設計方式，首先設定每個節點 in-degree 和 out-degree 之個數均介於 1 到 5 個之間，節點個數分為 50、100、200、500、1000 個共五種層級，且網路上任一節點均可到達迄點，不會有死路之情況發生。每個層級的網路又依照各節線時間範圍間距不同再分為三種網路，間距大小依序為 2 至 8 分鐘、2 至 15 分鐘以及均固定為 5 分鐘，每種路網各條節線的最小旅行時間均一致，亦即為所產生的最快之最短路徑會相同；間距 2 至 8 分鐘的網路旅行時間範圍為 2 至 24 分鐘；間距 2 至 15 分鐘的網路旅行時間範圍為 2 至 37 分鐘；間距固定為 5 分鐘的網路旅行時間範圍為 2 至 21 分鐘；因此總共有 15 個網路。

每個網路的情境設定依照選擇路徑之跨時段次數，分別測試兩種不同出發時間，綜合上述情形，共有 30 種不同組合。此外，選擇路徑之設定，採用在不跨時段之情境下，各網路最快的最短路徑，計算方式則根據本研究提出的 A1 演算法作為演算基礎而得；各時段之區分設定依序為 AM 7：30 至 AM 9：00 為上午尖峰時段，PM 5：30 至 PM 7：30 為下午尖峰時段，其他時間

均則屬於離峰時段。

本研究為測試節線時間範圍間距不同對於演算次數之影響，以及跨越時段次數對於演算次數之影響。首先，將節點個數為 50、100、200、500 之網路，出發時間設定為 AM 11：40 以及 PM 4：40 兩種情境，前者屬於不跨時段，後者則會跨越時段一次；由於 1000 個節點之網路經過節點較多，通常會至少跨越時段一次，因此出發時間設定為 AM 9：00 以及 PM 1：20 兩種情境，前者跨時段一次，後者則會跨越時段兩次。

本測試的演算過程中，為減少運算次數，且避免不必要的資訊，將不予紀錄機率值小於千分之一的時間範圍，或者若有重複的時間範圍，則會將其可能發生的機率值合併，可有效減少每個節點所需紀錄的時間範圍個數，如圖 4.1 至圖 4.2 所示，隨著網路節點數的增加，最後迄點所需紀錄的時間範圍個數並沒有顯著的成長。以下圖 4.3 到圖 4.7 為各種網路層級測試之演算次數比較圖，圖 4.8 到圖 4.17 為各種網路層級於不同時間出發對於到達時間範圍之機率分布圖。

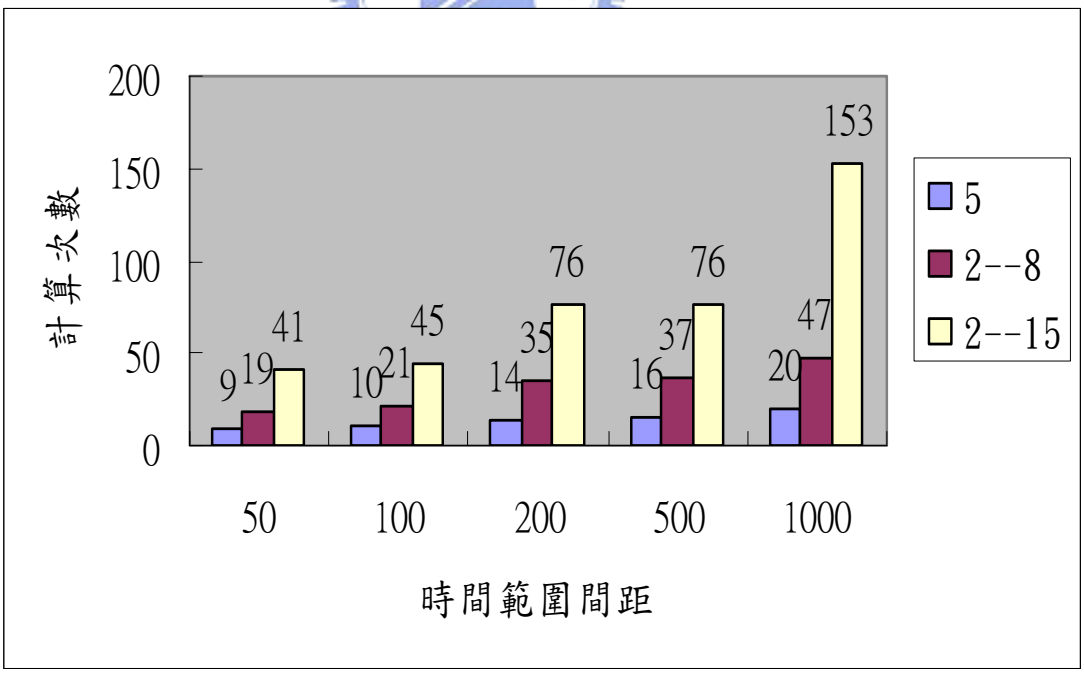


圖 4-1　迄點時間範圍個數比較圖（未跨時段）

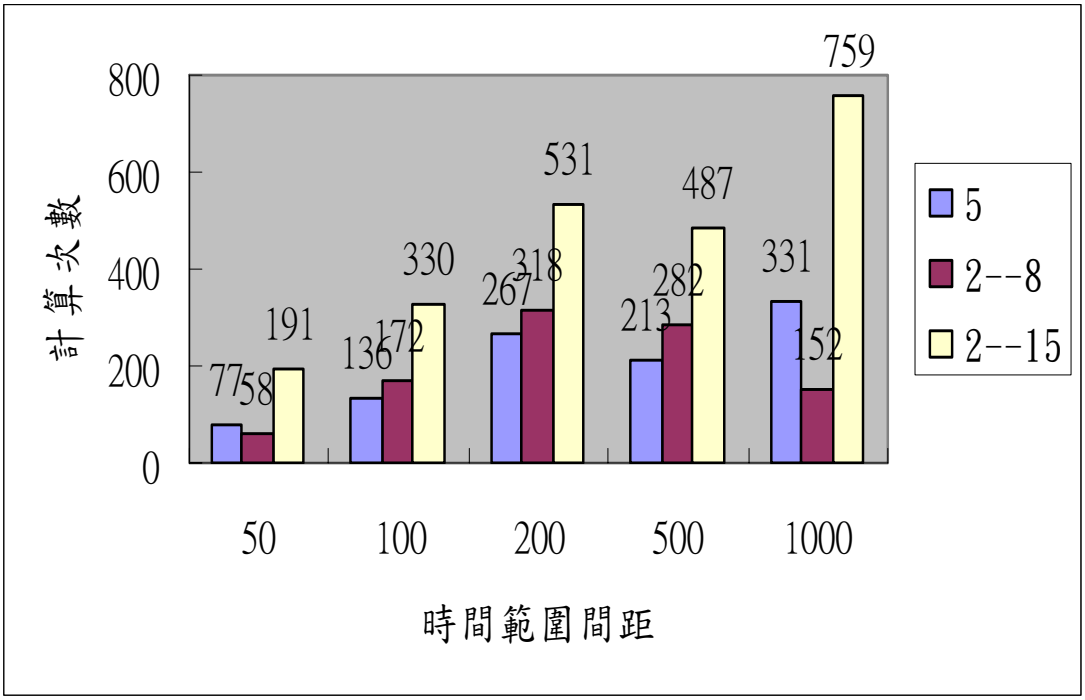


圖 4-2　迄點時間範圍個數比較圖（跨時段）

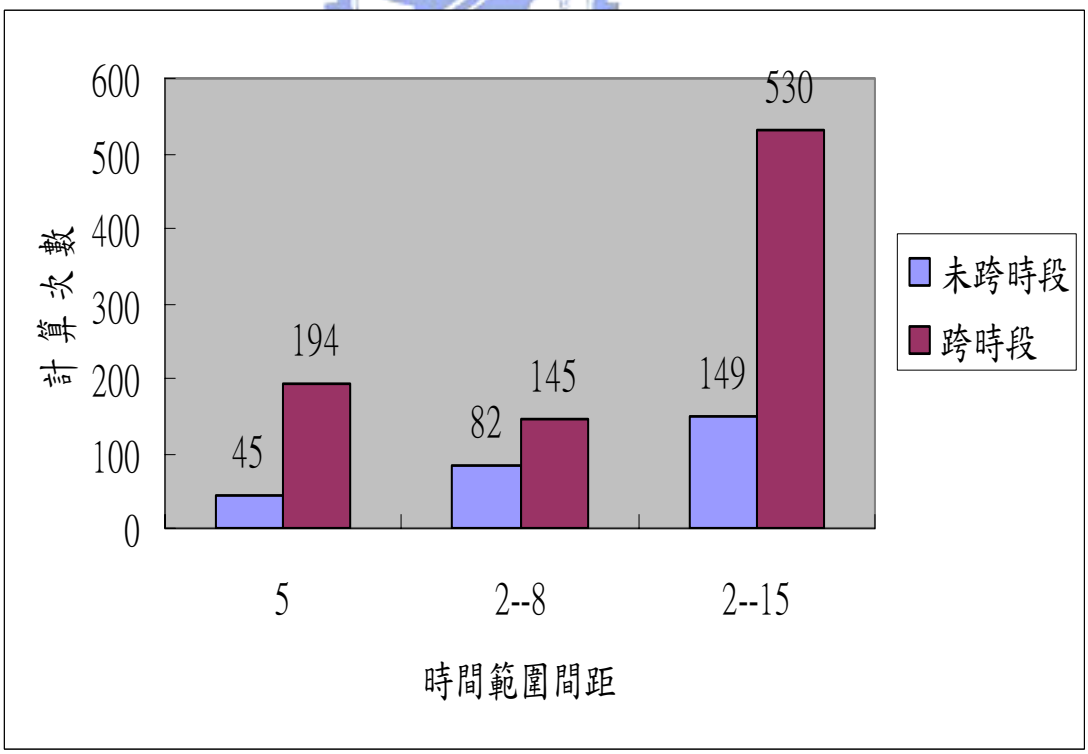


圖 4-3　50 個節點計算次數比較圖

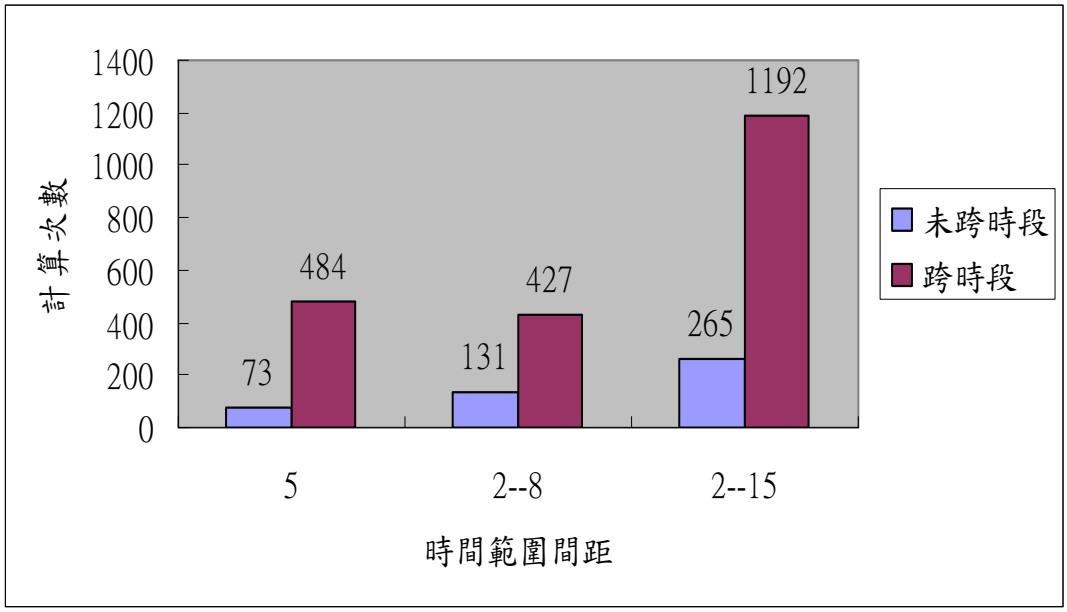


圖 4-4　100 個節點計算次數比較圖

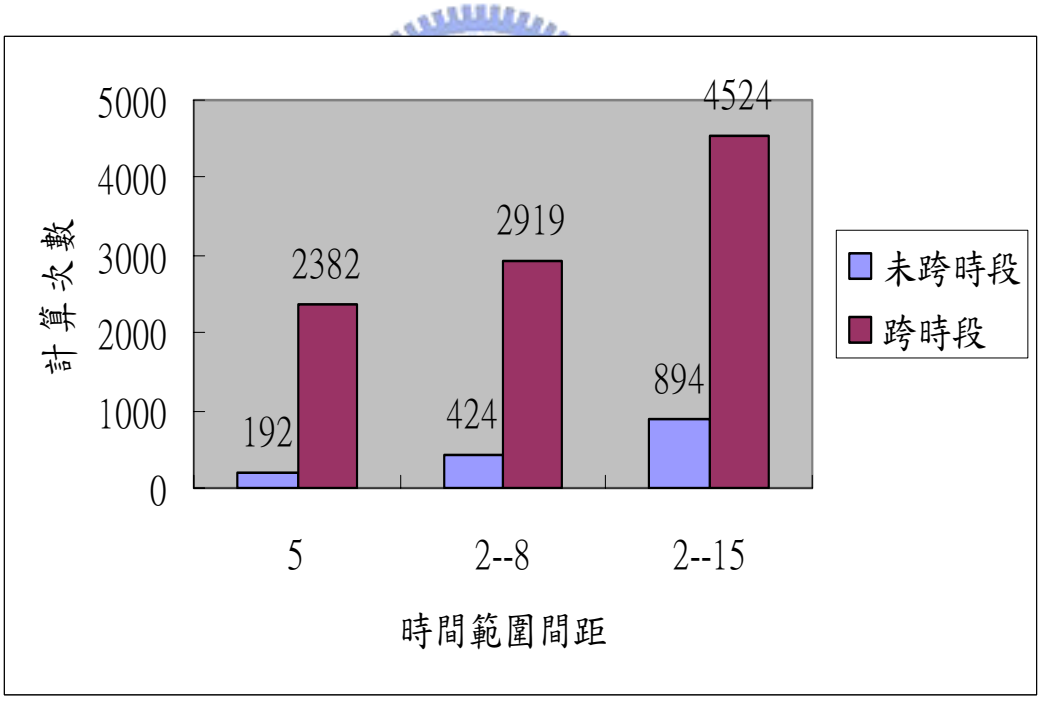


圖 4-5　200 個節點計算次數比較圖

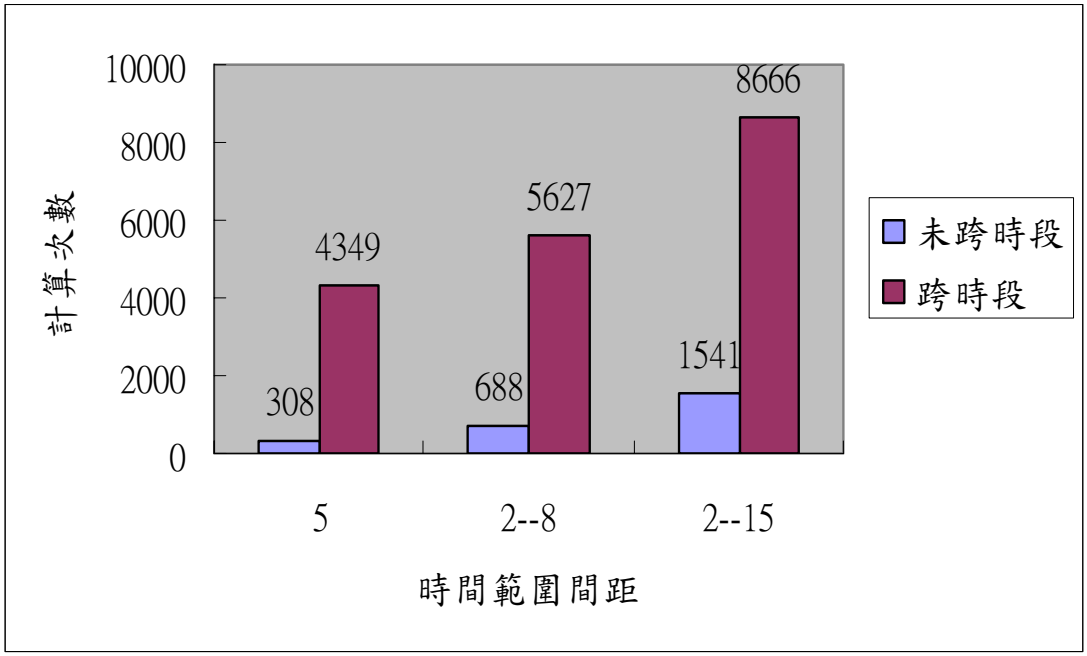


圖 4-6　500 個節點計算次數比較圖

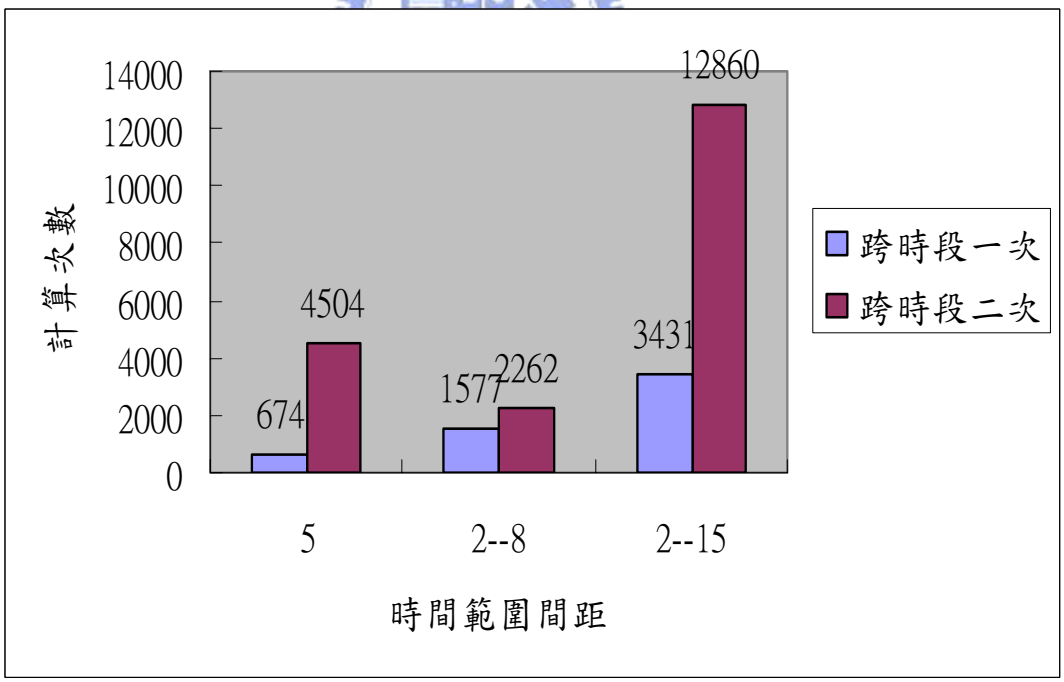


圖 4-7　1000 個節點計算次數比較圖

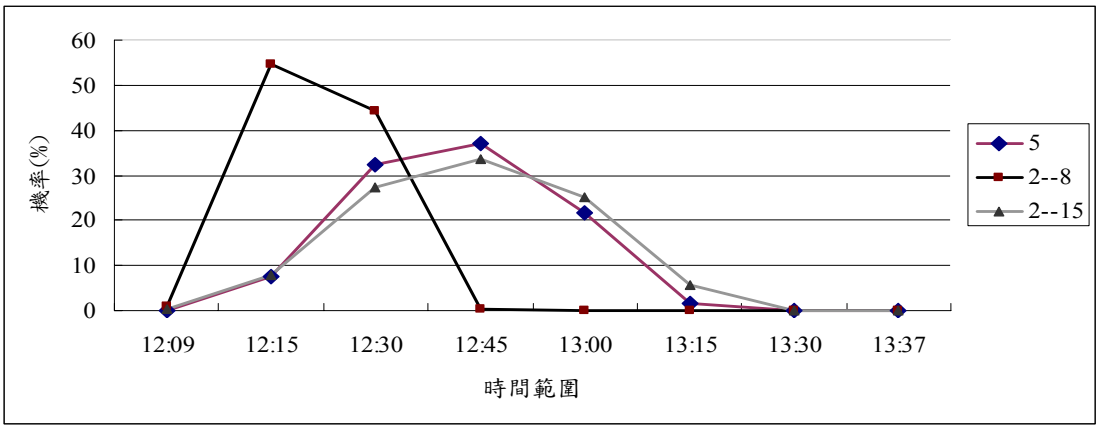


圖 4-8　50 個節點網路之各種到達時間機率分布圖（11：40 出發）

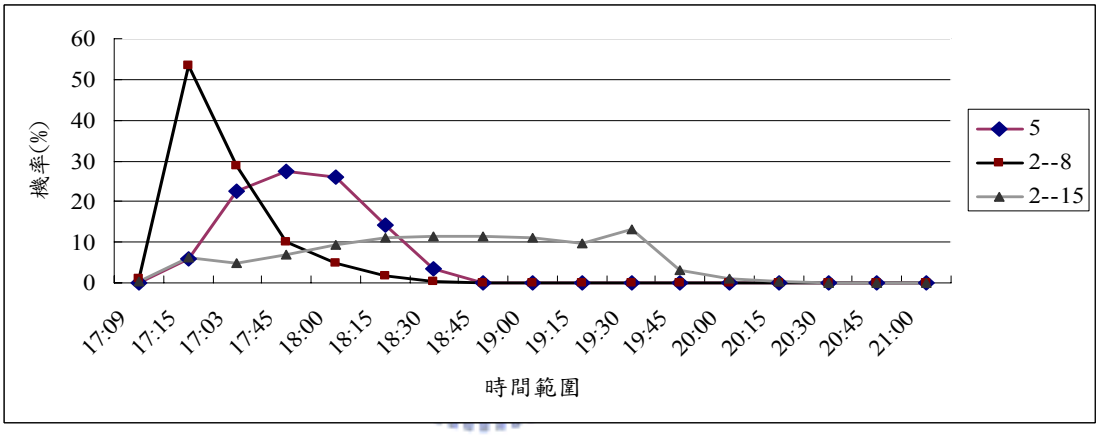


圖 4-9　50 個節點網路之各種到達時間機率分布圖（16：40 出發）

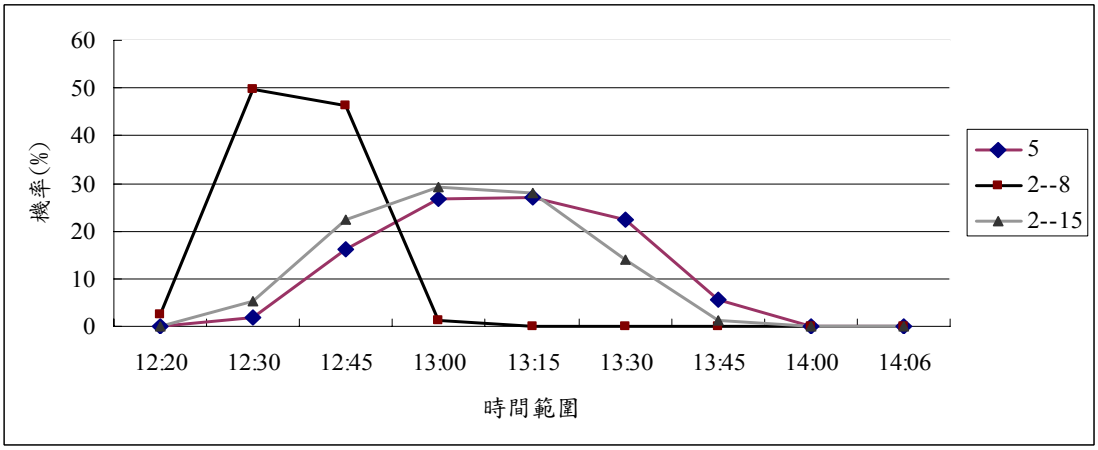


圖 4-10　100 個節點網路之各種到達時間機率分布圖（11：40 出發）

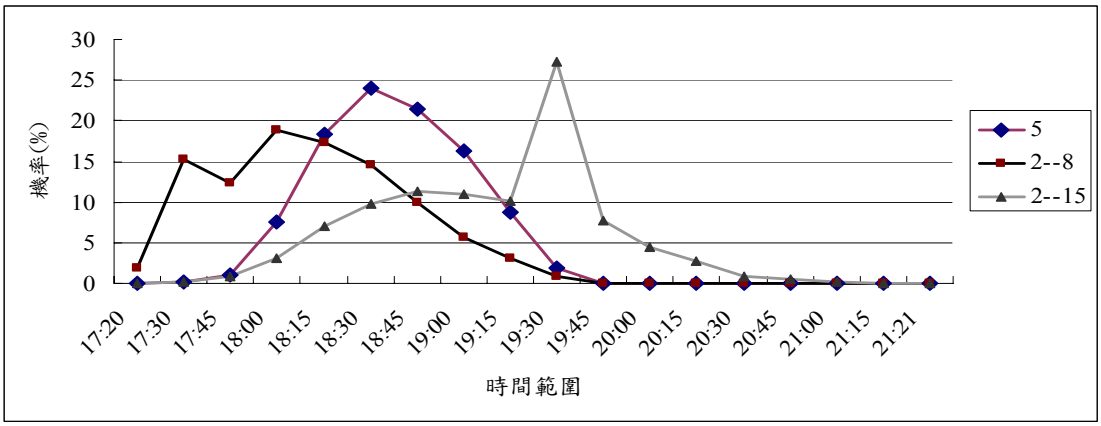


圖 4-11　100 個節點網路之各種到達時間機率分布圖（16：40 出發）

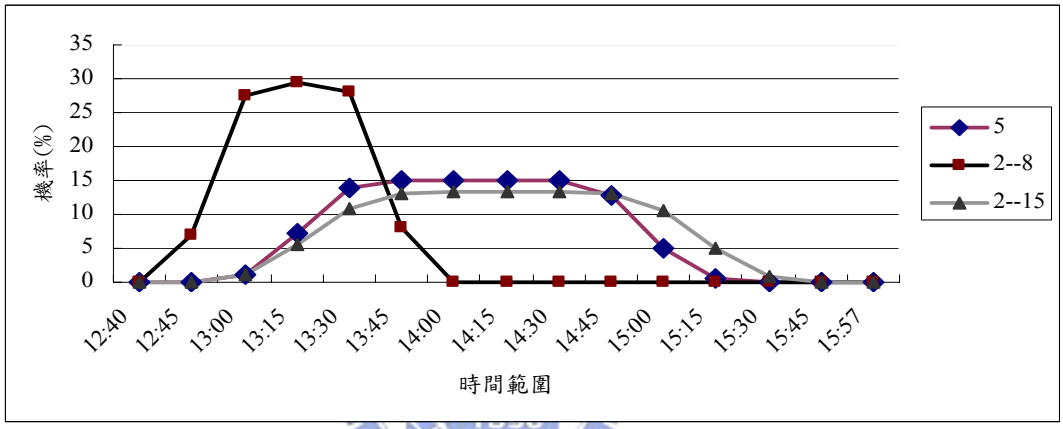


圖 4-12　200 個節點網路之各種到達時間機率分布圖（11：40 出發）

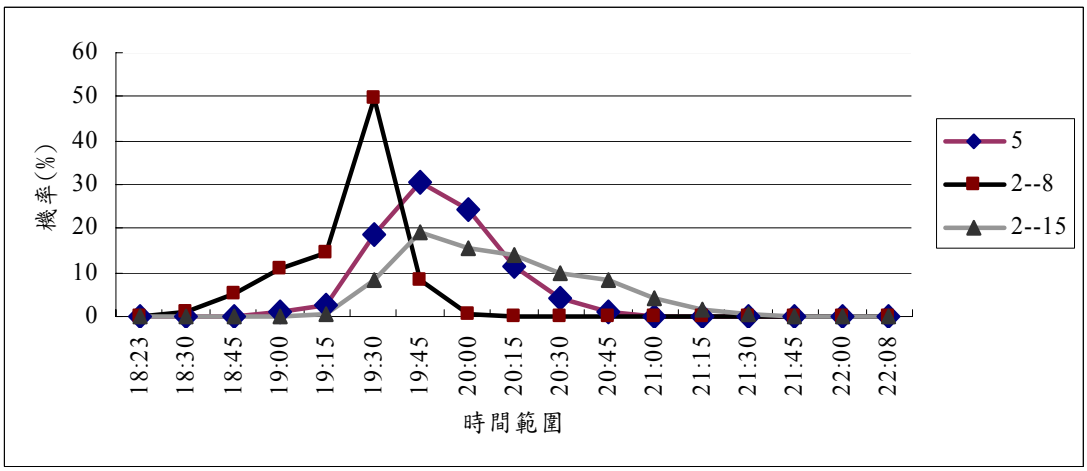


圖 4-13　200 個節點網路之各種到達時間機率分布圖（16：40 出發）

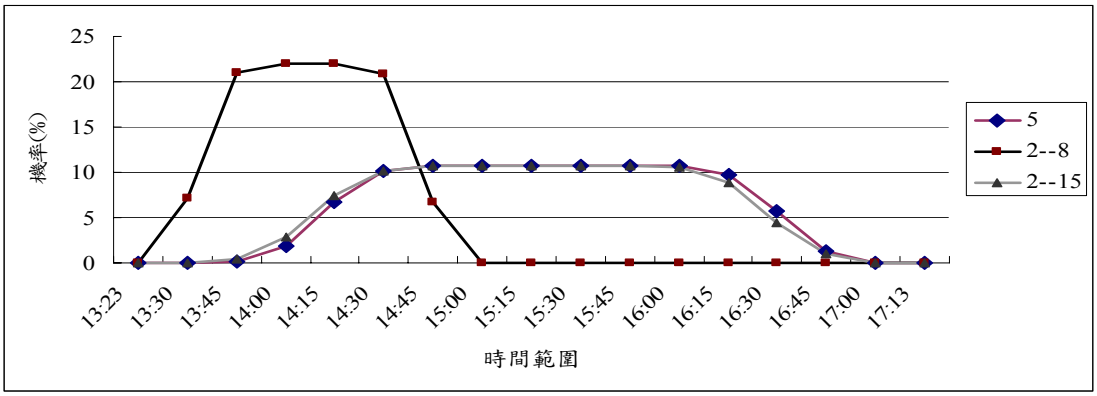


圖 4-14　500 個節點網路之各種到達時間機率分布圖（11：40 出發）

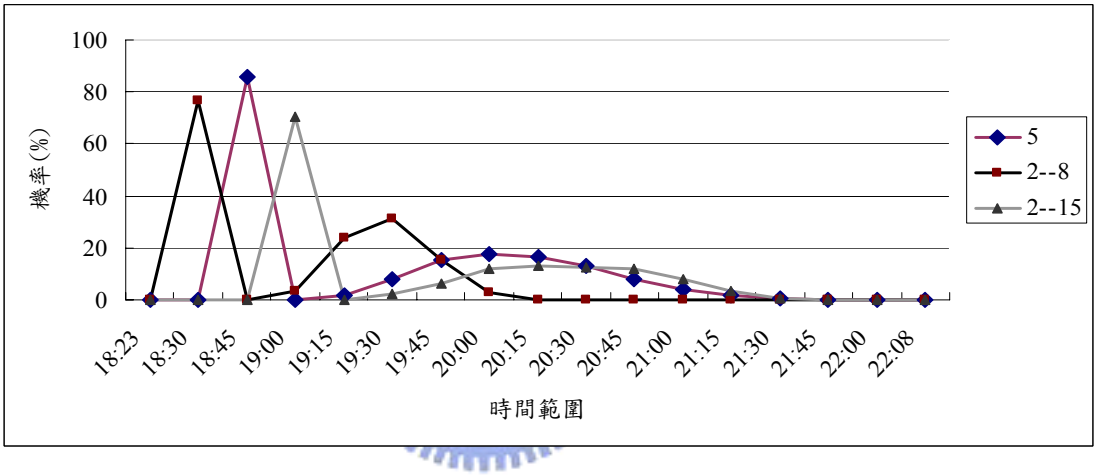


圖 4-15　500 個節點網路之各種到達時間機率分布圖（16：40 出發）

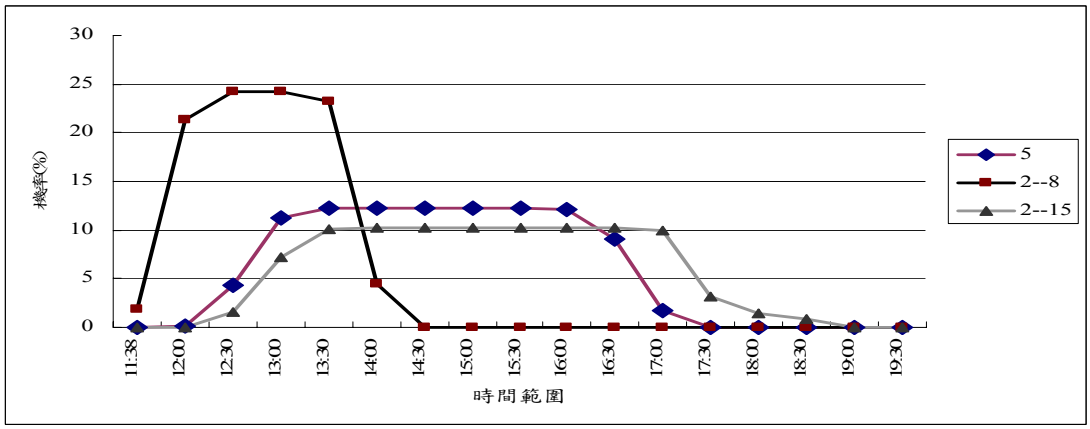


圖 4-16　1000 個節點網路之各種到達時間機率分布圖（9：00 出發）

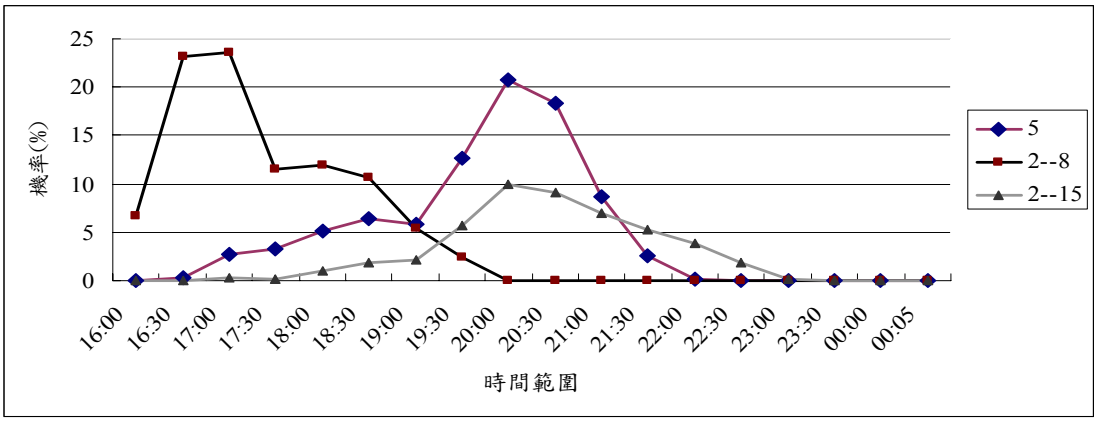


圖 4-17　1000 個節點網路之各種到達時間機率分布圖（13：20 出發）

經過實驗驗證，可發現以下五項情況：

1. 每個節點所需紀錄的時間範圍個數並不會隨著網路規模而成指數成長。主要原因為在演算過程中，已將機率值小於千分之一的時間範圍不予紀錄，因此當網路較大時，可刪除許多發生機率過低之情況；此外，當有重覆之時間範圍，可先將其機率值合併，故當節線時間範圍間距固定時，實際所需紀錄之時間範圍個數會明顯的減少。

2. 網路節點個數越大，將使得旅行時間範圍極度分散，導致各個時間範圍之發生機率相對減少，且需越久旅行時間之發生機率都趨近於零；然而在未跨越時段之情境下，仍會有較明顯的集中時間範圍。

3. 當發生跨越時段的情況，到達時間範圍的分布較不穩定，此乃為測試網路的節線旅行時間在不同時段差異較大之緣故。

4. 當各節線時間範圍間距差距均固定或範圍較小時，可有效減少運算次數。

5. 未跨越時段之情境下，節線時間範圍間距為 2 到 15 分鐘和固定為 5 分鐘的網路，所產生的到達時間範圍會較類似。

第五章　結論與建議

5.1　結論

本研究可歸納各項結論如下：

1. 本研究所設計之路網，節線成本以時間範圍呈現，並加入不同時段時間成本不同之觀念，較符合實際情況路網資料之呈現方式，且對將來實際路網資料之蒐集、統計也較易作業。

2. A1 和 A2 演算法可提供最短路徑之旅行時間誤差範圍，可提供用路人資訊，以自行判斷旅行時間之誤差是否在容忍值之內。

3. A3 演算法提供單一路徑的旅行時間範圍和相對應的機率值，可預估最有可能的到達時間範圍，提供較充足的旅行資訊。

4. 根據驗證結果可知，A3 演算法於節線時間範圍間距固定或範圍較小時，可有效減少運算次數。

5.2　建議

本研究仍有以下幾項缺失，以供後續研究者參考：

1. 實際路網資料仍可透過先進之科技技術以及統計分析取得，建議可採用實際路網資料驗證本研究提出之各演算法適用性。

2. A1 演算法仍無加入發生機率之考量，建議可配合各種到達時間之演算法概念，針對發生機率較高之最快的最短路徑作探討。

3. A3 演算法於時間範圍間距差異性較大之路網，演算次數會極遽增加，因此建議若蒐集實際路網資料，在統計分析時可盡量採用相同間距或縮小間距。

4. 當選擇之路徑過於龐大時，A3 演算法最終結果的時間範圍會呈現極

大的差距，造成各時間範圍之機率趨近於零，無法提供有效資訊；故建議針對大路網應另外提出適用之演算法較佳。

5. 當路網過於龐大時，A3 演算法計算次數成長雖不至於呈現指數爆炸，但增加速度仍極遽，故不適用於大路網。

6. A3 演算法計算過於複雜，因此一次僅能針對一條路徑作分析，若欲分析整個路網，則無法避免時間複雜度呈指數爆炸。

參考文獻

1. 陳慧琪,「時間相依最短路徑問題演算方法之研究」,國立交通大學,碩士論文,民國 89 年。
2. 劉偉賢,「汽車客運行前旅次規劃決策支援系統之規劃與設計」,中華大學,碩士論文,民國 94 年。
3. Amir Azaron and Farhad Kianfar, " Dynamic Shortest Path in Stochastic Dynamic Networks: Ship Routing Problem", *European Journal of Operational Research*,144, pp. 138-156, 2003.
4. Athanasios K. Ziliaskopoulos and Hani S. Mahmassani, "Time-Dependent, Shortest-Path Algorithm for Real-Time Intelligent Vehicle Highway System Applications", *Transportation Research Record.*, Vol.1408, pp. 94-100, 1993.
5. Cedric Davies, Pawan Lingras, "Genetic Algorithms for Rerouting Shortest Paths in Dynamic and Stochastic Networks", *European Journal of Operational Research*, 144, pp.27-38, 2003.
6. Elise D. Miller-Hooks and Hani S. Mahmassani, "Least Possible Time Paths in Stochastic, Time-Varying Networks", *Computers Ops Res.*, Vol. 25, pp. 1107–1125, 1998.
7. Elise D. Miller-Hooks, "Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks", *Transportation Science, Vol. 34*, No. 2, May 2000.
8. E. W. Dijkstra," A Note on Two Papers in Connection with Graphs", *Numeriske Mathematics, 1,* pp. 269-271, 1959.
9. Liping Fu, "Real-Time Vehicle Routing and Scheduling in Dynamic and Stochastic Traffic Networks", Department of Civil and Environmental Engineering, University of Alberta, Edmonton, Ph. D. dissertation, 1996.
10. Liping Fu and L.R. Rilett, "Dynamic O-D Travel Time Estimation Using An

Artificial Neural Network", in Proc. Vehicle Information & Navigation. Systems, 6th Annu. VINS, pp. 236–242, Seattle, Washington, July 1995.

11. Liping Fu and L.R. Rilett, "Estimation of Expected Minimum Paths in Dynamic and Stochastic Traffic Networks", Proceedings of IEEE's 6th International Conference on Vehicular Navigation and Information Systems (VNIS), pp. 200-205, Seattle, Washington, August 1995.
12. Liping Fu and L. R. Rilett, "Expected Shortest Paths in Dynamic and Stochastic Traffic Networks", *Transportation Research PartB*, Vol.32, No.7, pp. 499-516, 1998.
13. LR Ford., "Network Flow Theory", Santa Monica, California：The RAND Corp., 1956.
14. Nicholas Koncz, Joshua Greenfeld, and Kyriacos Mouskos, "A Strategy For Solving Static Multiple-Optimal-Path Transit Network Problems", *Journal of Transportation Engineering*, Vol. 122, No. 3, May/June 1996.
15. Robert W. Floyd, "Algorithm 97：Shortest paths", *Communications of the ACM,* v.5 n.6, pp. 345, June 1962.
16. S. Travis Waller and Athanasios K. Ziliaskopoulos, "On the Online Shortest Path Problem with Limited Arc Cost Dependencies", *Networks*, Vol. 40(4), pp. 216-227, 2002.