

逢甲大學
交通工程與管理學系碩士班
碩士論文



混合車流下模擬式動態交通指派模式之研究
-使用者均衡原則

A Study of Micro-simulation Based Dynamic Traffic
Assignment Model Under Mixed Traffic Flow
Environment :The Principle of User Equilibrium

指導教授：胡大瀛

研 究 生：陳炯男

中 華 民 國 九 十 四 年 七 月

誌謝

在逢甲六年來，陸續完成我的學士與碩士學位，從大三下加入 Pluto Lab 以來，胡大瀛博士是一位對我是影響很深的恩師，無論在研究上的指導或是生活，老師總是不厭其煩的關懷與叮嚀，在此獻上學生最深的感謝之意。

在論文口試期間承蒙 廖彩雲博士與 魏慶地博士百忙之中抽空指導，提供許多寶貴意見，並指導論文缺失需修正地方，使本論文更為完整。論文撰寫期間，所上 李克聰博士及 邱裕鈞博士細心審閱及指正，使學生論文順利完成，在此一併致上最誠摯之謝意。在此也感謝所上其他老師與助教們於在學期間各方面的幫忙。

而在研究過程中，也要感謝 Pluto Lab 成員許多幫忙，感謝麗雯學姊提供論文相關意見，百賢學長於程式的大力指導與修正，達瑾學長論文相關問題答覆，蔚明學長依時性路徑程式指導，文能學長、庭銀、彥臻、大昕及其他許多學長姊及學弟妹們，感謝你們當我需要幫忙時提供即時援手。

另外，還要感謝 Pluto 研究室戰友德政與怡如，這兩年來的同甘共苦，一起執行計畫案，一起熬夜趕報告，一起參加講習會、研討會等等；還有其他同學品翰、曜彰、彥斐、宇軒、俊宏、光鎧、怡芳、冠樺、秋評、品宏、文隆、自強、雲慶、偉強、孟宗、宗憲及裕雯學姊互相鼓勵下順利完成論文。

最後謹將本論文獻給我最愛的家人，更要感謝辛苦的母親以及因病過世的父親，感謝你們苦心栽培及提供良好學習環境，在此將這份喜悅及成果與你們分享；在未來我會更加努力，因為在這個人生旅途之中有了你們的支持與鼓勵才能完成此論文，感謝大家！

陳炯男 僅誌于
逢甲大學丘逢甲紀念館
民國九十四年七月

摘要

智慧型運輸系統(Intelligent Transportation System ,ITS)，是結合電子、電腦、通訊等相關科技之下，整合人、車、路管理方式，以達到有效的交通管理。而在ITS架構中先進交通管理系統(advanced traffic management systems, ATMS)及先進駕駛者資訊系統(advanced traveler information system, ATIS)，期望透過對道路使用管理，對駕駛者提供即時性資料，以達到路網整體有效利用。靜態傳統運輸系統分析方法已無法考量與反應出動態交通車流變化，因此，動態交通指派理論自1990年代起已逐漸受到重視。

動態交通指派依其構建方法可分為數學規劃式、最佳控制化之動態指派模式、變分不等式與德州動態指派模式等四種動態指派模式。數學規劃式、最佳控制化之動態指派模式、變分不等式等指派模式在考量求解問題下，需將模式簡化及連續時間切割為片斷改以離散問題處理，造成所產生結果無法適當解釋交通量依時性變化，而德州動態指派模式，主要透過模擬過程，瞭解各模擬時段下路網上路段交通量變化。

本研究之模擬式動態交通量指派主要以使用者均衡原則為主，透過DynaTAIWAN模擬出路網中混合車流道路流量分佈型態，並計算出各路段之動態旅行成本，再依據動態旅行成本計算出各個指派時段下之依時性最短路徑，研究中路徑流量分配主要使用MSA(Method of Successive Averages)為更新路徑流量方法，再進行判斷兩個迴圈之路徑車流量分配比例是否小於收斂值。

本研究中模擬式動態交通量指派車種以小車與機車為主並於50節點路網中進行測試。探討動態使用者均衡之條件並討論動態使用者均衡之結果。

關鍵詞：模擬式動態交通指派模式、使用者均衡原則、DynaTAIWAN、MSA

ABSTRACT

The Advanced Traffic Management systems (ATMS) and Advanced Traveler Information systems (ATIS), aim at providing the real-time information to travelers and improving traffic congestion in the network. The traditional transportation system analysis cannot consider the dynamics of traffic flows, thus the dynamic traffic assignment mode is a critical issue in the development of ITS. Four different approaches, including mathematical formulations, optimal control theory, variational inequalities, and simulation-based approach, have been constructed in developing dynamic traffic assignment methods. Due to their basic assumptions, these models are unable to explain time-dependent traffic flow patterns. In this research, a microsimulation-based DTA model is applied to model dynamic user equilibrium under mixed traffic flow conditions. In the mixed traffic flows, different vehicle classes are considered, such as truck, passenger car, and motorcycles. The link travel times are calculated and applied in time-dependent shortest path calculation. The flow distribution follows the method of successive average method. Numerical experiments are conducted in a 50-node test network to illustrate the proposed approach, and observations on dynamic user equilibrium are made based on these numerical experiments.

Keywords: Dynamic Traffic Assignment, User Equilibrium, DynaTAIWAN, MSA

目錄

誌謝	I
中文摘要	II
英文摘要	III
目錄	IV
圖目錄	VI
表目錄	IX
第一章 緒論	1
1.1 研究背景與動機	1
1.2 研究目的	2
1.3 研究方法與流程	2
第二章 文獻回顧	5
2.1 靜態交通量指派	5
2.2 動態交通量指派	7
2.2.1 數學規劃式	7
2.2.2 最佳控制理論	7
2.2.3 變分不等式	8
2.2.4 動態模擬指派模式	8
2.2.5 TRANSIMS	9
2.3 旅行時間成本	10
2.4 依時性最短路徑	11
2.5 多車種動態交通指派	12
2.6 DynaTAIWAN 架構	13
第三章 動態交通量指派模式架構及演算流程	16
3.1 動態交通指派模式	16
3.1.1 動態交通量指派模式觀念架構	16
3.1.2 動態使用者均衡演算法	17
3.2 旅行成本	22

3.3 MSA 演算法	25
3.4 依時性最短路徑演算法	26
3.5 模擬式動態交通量指派程式結構及流程	29
3.5.1 模擬式動態交通量指派程式模組	29
3.5.2 DynaTAIWAN 模組	30
3.5.3 成本計算模組	31
3.5.4 依時性最短路徑模組	32
3.5.5 路徑流量分配模組	34
3.5.6 車輛產生模組	37
第四章 DYNATAIWAN 數值測試	39
4.1 數值測試環境	39
4.2 測試路網-50 節點路網	39
4.3 數值路網實驗設計	41
4.3.1 流量需求分佈測試	42
4.3.2 依時性最短路徑測試	53
第五章 動態使用者均衡指派實驗測試	56
5.1 動態使用者均衡指派實驗設計	56
5.2 單一車種實驗	58
5.3 混合車流實驗	80
第六章 結論與建議	85
6.1 結論	85
6.2 建議	86
參考文獻	87
附錄	90

圖目錄

圖 1.1	研究流程圖	4
圖 2.1	模擬式動態交通指派觀念架構	9
圖 2.2	DynaTAIWAN 延滯時間估計示意圖	11
圖 2.3	DynaTAIWAN 系統架構圖	15
圖 3.1	模擬式動態交通指派之觀念架構圖	17
圖 3.2	指派時段與模擬時段	19
圖 3.3	使用者均衡指派方法流程圖	21
圖 3.4	DynaTAIWAN 旅行時間估計示意圖	22
圖 3.5	汽機車停等車輛數示意圖	24
圖 3.6	路口延滯之依時性最短路徑流程圖	28
圖 3.7	模擬式動態交通量指派程式架構圖	30
圖 3.8	DynaTAIWAN 模組輸入輸出	31
圖 3.9	成本計算模組輸入輸出	32
圖 3.10	依時性最短路徑模組輸入輸出	33
圖 3.11	依時性最短路徑資料格式	33
圖 3.12	路徑流量分配輸入輸出	34
圖 3.13	指派時段下路徑流量資料格式	35
圖 3.14	路徑流量分配流程圖	36
圖 3.15	車輛產生模組輸入輸出	38
圖 3.16	車輛產生屬性資料格式	38
圖 4.1	50 節點路網圖	40
圖 4.2	均一流量需求分佈	43
圖 4.3	高尖峰流量需求分佈	43
圖 4.4	常態流量需求分佈	44
圖 4.5	均一流量需求分佈下各車種平均旅行時間	45
圖 4.6	均一流量需求分佈下各車種平均旅行距離	46
圖 4.7	高尖峰流量需求分佈下各車種平均旅行時間	47

圖 4.8	高尖峰流量需求分佈下各車種平均旅行距離	48
圖 4.9	常態流量需求分佈下各車種平均旅行時間	49
圖 4.10	常態流量需求分佈下各車種平均旅行距離	50
圖 4.11	小車旅行時間	51
圖 4.12	大車旅行時間	51
圖 4.13	機車旅行時間	52
圖 5.1	實驗操作步驟	58
圖 5.2	均一流量需求(0.3)時間趨勢圖	61
圖 5.3	均一流量需求分佈下(0.3)違反值	61
圖 5.4	均一流量需求(0.5)時間趨勢圖	62
圖 5.5	均一流量需求分佈下(0.5)違反值	62
圖 5.6	均一流量需求(0.7)時間趨勢圖	63
圖 5.7	均一流量需求分佈下(0.7)違反值	63
圖 5.8	尖峰流量需求(0.3)時間趨勢圖	67
圖 5.9	尖峰流量需求分佈下(0.3)違反值	67
圖 5.10	尖峰流量需求(0.5)時間趨勢圖	68
圖 5.11	尖峰流量需求分佈下(0.5)違反值	68
圖 5.12	尖峰流量需求(0.7)時間趨勢圖	69
圖 5.13	尖峰流量需求分佈下(0.7)違反值	69
圖 5.14	常態流量需求分佈下(0.3)時間趨勢圖	73
圖 5.15	常態流量需求分佈下(0.3)違反值	73
圖 5.16	常態流量需求分佈下(0.5)時間趨勢圖	74
圖 5.17	常態流量需求分佈下(0.5)違反值	74
圖 5.18	常態流量需求分佈下(0.7)時間趨勢圖	75
圖 5.19	常態流量需求分佈下(0.7)違反值	76
圖 5.20	增量因子 0.3 之旅行時間趨勢圖	76
圖 5.21	增量因子 0.5 之旅行時間趨勢圖	77
圖 5.22	增量因子 0.7 之旅行時間趨勢圖	77
圖 5.23	增量因子 0.3 之違反值趨勢圖	78

圖 5.24	增量因子 0.5 之違反值趨勢圖	78
圖 5.25	增量因子 0.7 之違反值趨勢圖	79
圖 5.26	混合車流下(0.5)旅行時間趨勢圖	82
圖 5.27	混合車流下(0.5)違反值趨勢圖	83
圖 5.28	混合車流下(0.7)旅行時間趨勢圖	83
圖 5.29	混合車流下(0.7)違反值趨勢圖	84



表目錄

表 3.1	汽機車速度差異比值 β 值	23
表 4.1	路網屬性設定值對照表	41
表 4.2	均一流量需求分佈下系統績效統計表	45
表 4.3	高尖峰流量需求分佈下系統績效統計表	47
表 4.4	常態流量需求分佈下系統績效統計表	49
表 4.5	常態分佈下路徑之旅行時間與路徑	54
表 4.6	自由車流與依時性最短路徑旅行時間比較	55
表 5.1	均一流量需求分佈(0.3)實驗結果	59
表 5.2	均一流量需求分佈(0.5)實驗結果	59
表 5.3	均一流量需求分佈(0.7)實驗結果	60
表 5.4	尖峰流量需求分佈(0.3)實驗結果	64
表 5.5	尖峰流量需求分佈(0.5)實驗結果	65
表 5.6	尖峰流量需求分佈(0.7)實驗結果	66
表 5.7	常態流量需求分佈(0.3)實驗結果	70
表 5.8	常態流量需求分佈(0.5)實驗結果	71
表 5.9	常態流量需求分佈(0.7)實驗結果	72
表 5.10	混合車流下(0.5)實驗數值	80
表 5.11	混合車流下(0.7)實驗數值	81

第一章 緒論

1.1 研究背景與動機

國內在科技發展以及汽、機車車輛的快速成長發展下，對於都市交通環境已產生衝擊，因此該如何利用有效資源做好交通管理與分析預測是目前相關單位努力之目標。近年來智慧型運輸系統(Intelligent Transportation Systems, ITS)的發展受到相當之注視，此一系統主要是透過結合電子、電腦、通訊等相關科技，整合人、車、路之管理方式，以達到有效的交通管理(交通部，2001)。在智慧型運輸系統中，包含許多子系統，其中的先進交通管理系統(Advanced Traffic Management Systems, ATMS)及先進先進用路人資訊系統(Advances Traveler Information Systems, ATIS)係期望透過對道路使用的管理，對旅運者提供即時性資訊，對路網交通車流進行有效之管理，並提升路網績效。為了分析 ITS 架構下之路網車流，傳統的靜態分析方法因為缺乏考量車流於路段上隨時間的變化情形，無法滿足即時資訊交通路網之環境，難以應用在 ITS 環境中。因此，自 1990 年代起動態交通指派之理論已逐漸受到重視。主要是因為動態交通指派方法包含幾個特點：(1)能將即時性資料納入指派考量；(2)可反應出路網中路段於各時段之流量變化，使車流分佈較為符合現實世界的交通環境。

根據胡大瀛於 2001 年之研究，動態交通指派依其建方法可分為數學規劃式、最佳化控制之動態指派模式、變分不等式之動態指派模式與德州動態指派模式等四種方式(胡大瀛，2001)。該研究所提之模擬式動態交通指派觀念架構中，主要是分析車輛於不同指派行為假設下之路徑產生。在此一架構下，使用模擬方式來取代傳統路段績效函數之運用，如路段旅行時間與路口延滯之產生等。

本研究將透過 DynaTAIWAN 交通分析與預測系統模擬出路網中車流分布狀況，並透過交通量指派方式，以使用者均衡(User Equilibrium, UE)之原則導引路網車流，期以分派交通量的方式減緩路網擁擠程度。因此，本研究將針對動態交通指派模式下使用者均衡進

行相關探討與研究。

1.2 研究目的

本研究依據模擬式動態交通指派之架構，探討以使用者均衡為原則，對混合車流進行指派分配，並透過 DynaTAIWAN 模擬混合車流之分布。其中，混合車流係指研究中考量多車種於路網共同運行的情況，本研究所考量車種，主要是根據車輛實體型態來分類，車種考量包含大車、小車以及機車。

根據上述之說明，本研究之研究目的歸納為下列幾項：

1. 在使用者均衡理論基礎下，建構混合車流下多車種之使用者均衡理論。
2. 利用 DynaTAIWAN 建立模擬式動態使用者均衡交通量指派模式。
3. 以 Visual C++ 撰寫混合車流下動態交通量指派之使用者均衡演算法。
4. 透過 50 節點路網進行實測，以驗證模擬式動態交通指派應用於路網之可行性。

1.3 研究方法與研究流程

本研究進行之方法步驟如下：

1. 問題與研究範圍確認

透過對於研究課題的了解，確認問題以及研究範圍。本研究主要探討混合車流下之交通量指派模式。

2. 文獻回顧

根據相關課題進行文獻蒐集回顧，包含：

(1) 靜態交通量指派

- (2) 動態交通量指派
- (3) 旅行成本
- (4) 依時性最短路徑
- (5) 多車種動態交通指派
- (6) DynaTAIWAN 架構

3. 模擬式動態交通量指派構建

本研究之模式主要是開發出模擬式動態交通指派模式，其包含旅行成本、依時性最短路徑計算及車流模擬之核心軟體 DynaTAIWAN 等三部分，而 DynaTAIWAN 主要負責模擬路網車流行進分布與提供路段車流型態資料。

4. 系統開發

根據前述構建之模式，使用 Visual C++ 程式語言開發本研究之模擬式動態交通指派模式。

5. 建立收斂條件

建立動態交通指派模式收斂條件，參考李達權(2002)建立之收斂條件為基礎訂定收斂條件。

6. 路網實測

以一 50 節點測試路網，進行測試與資料分析，並觀察路網車流量變化及驗證所構建之模擬式動態使用者均衡指派模式。

7. 結論與建議

統整研究成果，並提出結論與未來研究發展之建議。

綜上所述，本研究之研究流程如圖 1.1 所示。

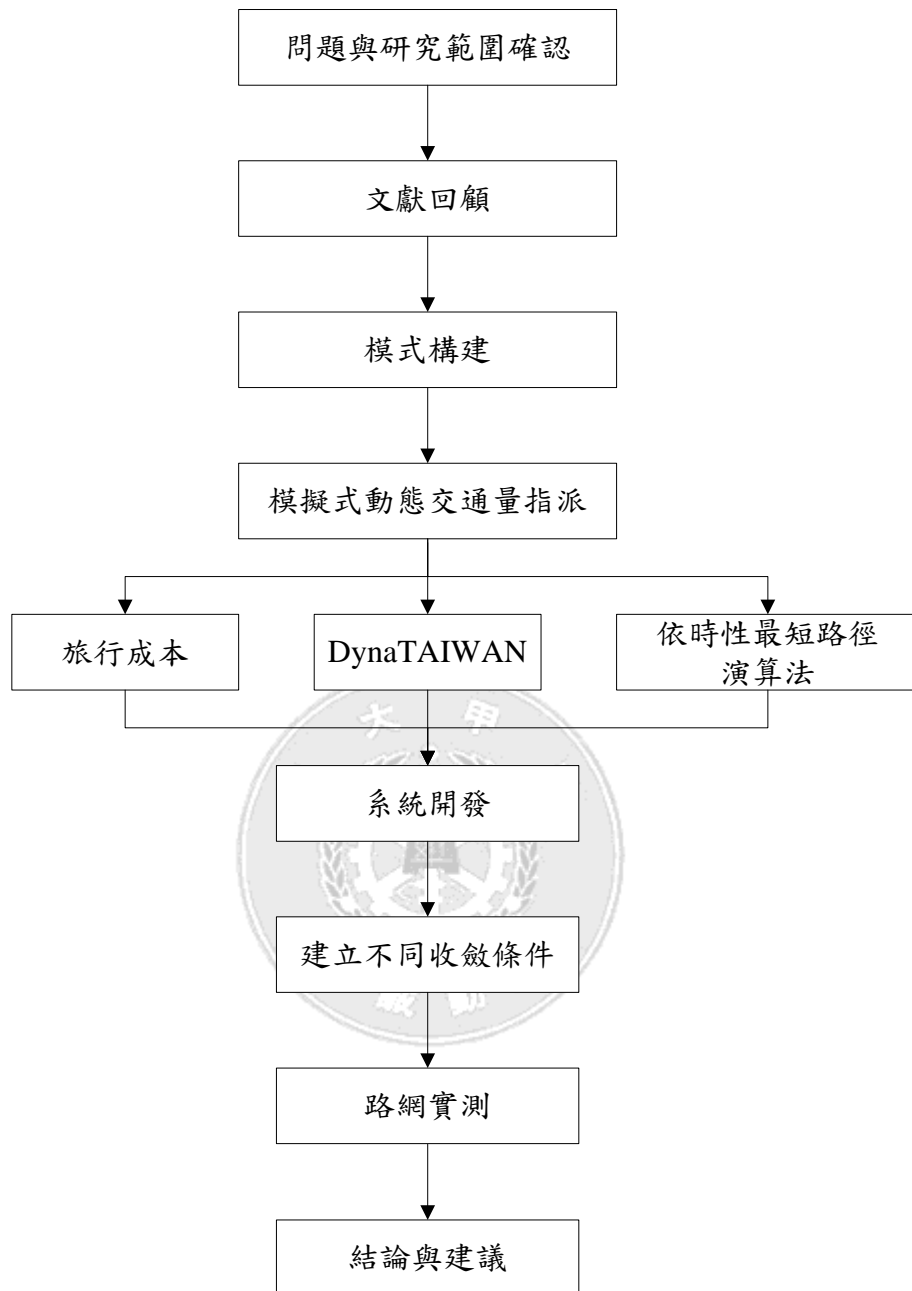


圖 1.1 研究流程圖

第二章 文獻回顧

本章進行相關文獻之回顧。2.1 回顧靜態交通量指派，2.2 節針對動態交通量指派模式求解方法進行回顧；2.3 節回顧旅行成本相關文獻，2.4 節說明依時性最短路徑，2.5 節回顧討論多車種動態交通指派，而 2.6 節則說明本研究所使用之交通分析與預測系統 DynaTAIWN。

2.1 靜態交通量指派

在胡大瀛(胡大瀛，2001)的研究中，對於靜態交通指派的方法作一完整的回顧說明。該文獻指出，靜態交通指派係假設路段的流量和旅行時間不隨時間之變動而改變。因此，分析中將一個靜態起迄 OD 旅次表指派於路網中的路段，透過累計的方式獲得車流在路網分布的情形。

1956 年 Beckmann et 提出第一個使用者均衡(User Equilibrium, UE)指派數學模式，且其模式所得到解，具存在(Existence)與唯一(Uniqueness)等數學特性，並且滿足 Wardrop (1952)所提出的使用者均衡的條件，亦即「沒有駕駛者可以藉由路徑的改變改善旅行時間或成本」。

一般路網均衡模型主要根據 Wardrop(1952)的用路人或系統均衡原則，以及事先校估的路段績效函數，將路網上起迄對間的運輸需求，有系統地分配到路網各個路段上所構建的模型。下列數學式 2-1 至 2-4 表示使用者均衡指派數學模式，式 2-5 至 2-8 表示系統最佳化數學模式：

$$\min Z(w) = \sum_a \int_0^{w_a} t_a(w) dw \quad (2-1)$$

subject to

$$\sum_k f_{ijk} = f_{ij}, \forall i, j \quad (2-2)$$

$$f_{ijk} \geq 0, \forall i, j, k \quad (2-3)$$

$$w^a = \sum_i \sum_j \sum_k f_{ijk} \delta_{ijk}^a, \forall a \quad (2-4)$$

$$\min Z(w) = \sum_a w^a t_a(w^a) \quad (2-5)$$

subject to

$$\sum_k f_{ijk} = f_{ij}, \forall i, j \quad (2-6)$$

$$f_{ijk} \geq 0, \forall i, j, k \quad (2-7)$$

$$w^a = \sum_i \sum_j \sum_k f_{ijk} \delta_{ijk}^a, \forall a \quad (2-8)$$

其中，

w ：代表路段流量之值；

w^a ：為路段 a 上的流量；

$t_a(.)$ ：代表路段之時間函數，或稱績效函數(link performance function)，

$t_a(.)$ ：一般是正、遞增，且為凸形函數。

f_{ij} ：代表由起點 i 至迄點 j 的指派量，

f_{ijk} ：為 $(i \sim j)$ 間第 k 路徑的流量，

δ ：靜態路段一路徑之指標，可用於路徑流量(path flow)與路段流量(link flow)的換算。

使用者均衡(UE)與系統最佳化(SO)之求解過程非常類似，使用者均衡(UE)主要是考量起、迄點間使用路徑平均成本(旅行時間)相等，而系統最佳化則是考量起、迄兩點間之路徑邊際成本相等。

一般而言，靜態指派模式的主要缺點可歸納如下(胡大瀛，2001)：

1. 無法描述交通擁擠或尖峰間的交通狀況。
2. 無法考慮在 ATIS/ATMS 的系統下，駕駛者可能的個別反應。

而動態交通指派模式則主要考量靜態指派之缺點並顧及系統最佳化(SO)與使用者均衡(UE)的特性。

2.2 動態交通量指派

經由交通量指派，可以產生路網上車流量之分佈，應用於交通管理規劃使用，而靜態交通量指派過程中忽略不同時間下車流量之變化，因此，動態交通量指派自 1990 年起逐漸受到重視，且隨著 ITS 下先進交通管理系統(ATMS)以及先進用路人資訊系統(ATIS)之發展，交通管理可透過動態交通量指派，於短時間內提供即時資訊給予駕駛者使用。

動態交通量指派模式，依據其研究內容與建構方式可分為數學規劃式、最佳控制化之動態指派模式、變分不等式動態指派模式與模擬式動態交通指派模式（德州動態指派模式），說明如 2.2.1、2.2.2、2.2.3、2.2.4 各小節。

2.2.1 數學規劃式

數學規劃式最早是由 Merchant 與 Nemhauser(1978a, 1978b)所提出。M-N 模式主要改良傳統靜態交通量指派，把旅次產生視為時間函數，主要針對單一迄點路網以數學規劃法建立一個片斷連續、非線性(non-linear)且非凸(non-convex)函數之動態系統最佳化交通量指派模式。由數學規劃建立之動態交通量指派模式，其模式較為簡易但限制較多，且無法處理連續時間之問題。後續則有 Ho(1980)、Ghali and Smith(1992)針對動態模式進行探討與研究。

2.2.2 最佳控制理論

最佳控制理論的前身為變異微分(Calculus of Variations)，而此一理論主要以最佳化控制理論(Optimal Control Theory)為基礎，導出連續性之最佳控制方程組。而許多控制理論已被成功應用於航空與太空領域中。此一研究方法假設研究期間已知連續性、依時性之起迄點流量(continuous time-dependent OD flows)，其 Friez et al.(1989)理論的發展以系統最佳化與使用者均衡的動態指派為主，於數學式中將路段的

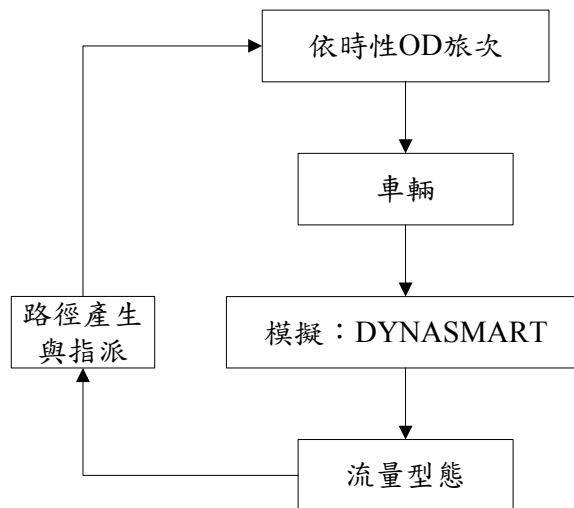
流入率視為控制變數，將路段流出率視為路段上車輛的函數，因此模型較為複雜。

2.2.3 變分不等式

近年來有多位學者以變分不等式(Variation Inequality, VI)建構動態交通指派模型。最早由 Friez et al.(1989)、Ran et al.(1993)提出，其主要是架構於最佳控制理論下，求解過程大半能轉換為一般非線性數學規劃模型來求解。Ran 與 Boyce(1996)之著作中除討論最佳控制理論之動態指派模式外，並討論變分不等式之動態指派模型與演算法。而後還有 Chen(1999)針對變分不等式與動態旅運需求模型進行動態路網均衡模型之探討，依據交通資訊型態，Chen 發展出三大類動態使用者最佳化旅運模型，包括確定性、隨機性與模糊性等模型，並可依據旅運決策來細分動態路徑指派、出發時間與路徑選擇等之模式求解與應用，模型具有簡單以及數學證明嚴謹性，求解演算法則以巢狀對角法(Nested Diagonalization Method, NDM)為中心。

2.2.4 模擬式動態交通指派模式

模擬式動態交通指派模式是在 1990 年於德州大學奧斯汀分校(The University of Texas at Austin)開始發展，其主要根據 Carey(1986)的觀念發展數學模式，但是此一方程式無法以現有數學方法求解。因此，為了能夠將車流特性與行為描述出來，將以模擬方式取代傳統路段績效函數(Link Performance Function)與路段流出函數(Exit Function)。而此一模擬式動態交通指派模式的問題在於如何依據指派法則將旅次分配於路網中。此一動態模擬指派模式架構如圖 2.1 所示，架構中車輛會依據依時性起迄點資料產生，此一架構每輛車會依定義指派規則分配路徑，而這些路徑則可經由個體路徑選擇模式或演算法求得，如系統最佳化(SO)與使用者均衡(UE)。



資料來源：(胡大瀛，2001)

圖 2.1 模擬式動態交通指派觀念架構

2.2.5 TRANSIMS

TRANSIMS (Transportation Analysis and SIMulation System)為一模擬指派軟體(Rickert and Nagel, 2001)，此軟體主要為 Los Alamos 國家實驗室在 1995 年所發展之微觀模擬模式。該模式是以細胞自動機(Cellular Automata, CA)為主要構想，將路網中車輛視為一個細胞，進而模擬出某一區域下每一個車輛個體之活動，並且利用個體間互動所產生之交通狀態，進而計算出車輛排放物來獲得相關結果，其功能主要提供區域性旅行與預測交通績效以及環境衝擊之微觀模擬模式。目前此一軟體已能以美國 Dallas 為模擬實驗路網，模擬四個小時、200 萬部車輛於 25 平方英哩之交通型態，但由於微觀模擬需要大量的計算能力配合，故此模擬軟體需使用超級電腦來進行模擬。

在 TRANSIMS 模擬軟體中，其交通量指派方式為，旅運者先依循 OD 矩陣之起迄點，根據 OD 矩陣所產生之路徑進行移動，但在路網中之旅運者會隨著路徑狀況(例如當旅運者遇到壅塞情況或是意外事件時)進行路徑的反覆修正。其中路徑計算主要是依據依時性的路段旅行時間來估算出當時的路徑。

2.3 旅行時間成本

本節針對相關文獻對於旅行時間成本之使用與計算進行回顧。交通量指派中，如要將車流狀況實際反應出來，則需將旅行時間成本加入考量，於此所指旅行時間成本包含行車時間(Running Time)以及路口停等造成的時間延滯(Intersection Delay)兩部分；而在路網交通量指派實務上，國內最廣泛使用數學式為美國公路局(U.S Bureau of Public Roads, BPR)所提出，稱為 BPR 公式(李俊賢，1996)，如下所示。

$$T = T_0 \left[1 + \alpha (Q/C)^\beta \right] \quad (2-9)$$

其中，T：在流量為 Q 時之路段平均旅行時間

T_0 ：自由車流速率之路段旅行時間

Q：路段流量

C：實際容量(Practical Capacity)

α 、 β ：參數值

曾莉莉(1995)利用 FRESIM 模擬軟體進行高速公路動態路段旅行時間函數探討，利用 FRESIM 模擬出路段上車輛數、流入與流出車輛數後，再計算出該路段平均速率，由平均速度換算出該路段車輛旅行時間。

伍靜怡(2002)主要參考 Ran et. al(1997)等人估算路段旅行時間之研究，將實際路段區分為兩抽象性的區域，將路段區分為路口端與路段兩個區域，其計算車輛旅行時間則包含車輛於路口之延滯時間與路段巡行時間，其中路段巡行時間是將路段長度扣除路口等候車輛長度除以車輛於該路段之平均速率。

邱華敏(2003)針對高速公路事故路段旅行時間進行估算，其使用方法主要參考 Ran and Boyce(1994)所提出之動態旅行時間函數並加入微觀車流，進而發展高速公路事故路段動態路段旅行成本函數。

運輸研究所與逢甲大學於 2004 年所合作開發之 DynaTAWIN 系統(運研所，2004)則是將旅行時間成本分為旅行時間與延滯時間兩區間

進行估算，而為了能正確計算出路段上與路口車輛旅行時間，將實際路段區分為兩個抽象性的節線，分別代表上游路段移動中的車輛與下游端停等中車輛，如圖 2.2 所示。

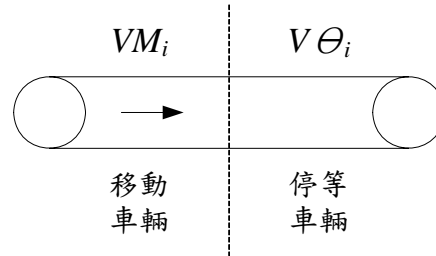


圖 2.2 DynaTAIWAN 延滯時間估計示意圖

2.4 依時性最短路徑

一般的最短路徑問題(Shortest Path Problem)之路段之間的旅行成本為固定常數，但在依時性最短路徑問題(Time-dependent Shortest Path Problem)中，各路段的旅行時間成本會隨著時間的不同而有所差異(林蔚明，2004)。以下針對相關文獻回顧討論。

Hall (1986)主要是為了求解隨機且與時間相依的旅行時間之最小期望旅行時間路徑問題。文獻中主要將旅行時間分類為定性與隨機變數兩類，定性旅行時間主要是收集歷史資料經由統計獲取定性資料；隨機變數旅行時間則是利用到達節線時間以及滿足統計分配的隨機變數為依據。經由定性旅行時間資料可以求得最小可能時間最短路徑，而根據隨機變數的旅行時間則是求得其最小期望時間最短路徑。

Ziliaskopoulos 與 Mahmassani (1996)除了考量路段旅行時間外，加入轉向所造成之延滯時間，並利用修正後 Label Correcting 演算法來進行求解。

Ziliaskopoulos et al. (1997) 則介紹一種平行的依時性最短路徑演算法。其所使用的演算法與 Ziliaskopoulos 與 Mahmassani (1996) 相同，不同之處在於求解最短路徑時，採用多台的處理器同時進行計算，僅在讀取旅行時間成本時使用共同記憶體 (shared memory)。

林蔚明(2004)則為反應出真實交通狀況，其依時性最短路徑之計算加入車輛通過路口時所產生之延滯成本來反應出真實交通狀況。

2.5 多車種動態交通指派

本研究主要針對混合車流進行指派考量，因此本節將回顧之文獻對於路網中多車種(Multiclass)的分類作整理說明。回顧之相關文獻中，多車種分類主要可分為兩類：以旅運行為分類及車輛型態(Vehicle Types)之分類。

依據旅運行為所進行的多車種分類，主要考量旅運者會因其對交通路網認知情況有差異導致選擇之路徑有所不同來分類。此一分類法最早由 Dafermos(1971, 1972)所提出，其假設為每一個旅運者對路網狀況具有一定認知下，再針對旅運者所進行之分類，後續則有 Wynter(1995)、Chen(1999)、Bliemer(2001)與 Lo(1999)、Lo et al.(1996)、Ran et al.(1996)，等學者針對多車種旅運行為進行分類，分別為固定路徑(Predetermined or Fixed Routes)、隨機使用者最佳(Stochastic Dynamic Traffic Assignment Optimum, SDUO)、動態使用者最佳(Dynamic User Optimum, DUO)等三種分類。而胡大瀛(2001)所使用之多車種則係依據旅運行為分為四種車種，包含：系統最佳化(SO)、使用者均衡(UE)、可獲得即時資訊者與固定路徑使用者。

另一分類方式是以車輛實體型態為主，Hoogendoorn 和 Bovy(2000,2001)等學者將分類分為大車(Truck)、小車(Passenger Car)，此一分類方式主要考量每一車輛類型擁有個別的特性，如最高速度、車輛大小以及可使用之道路公路設施(Bliemer,2003)。Bliemer (2003)則針對大車與小車進行 UE 指派研究，此文獻說明，在同一道路上，小汽車與大型車之速度不同，且彼此之間會互相影響。Bliemer 還選取鹿特丹港市(Rotterdam)西邊環城高速公路(Beltway)A16 區域進行實驗，實驗之車道是由南向北方向並包含一個大車車道。選定實驗範圍後，依序計算各路段小車速度後，再將小車速度乘以一比例 γ 後進行計算出大車速度，以此方式計算出同一路段上小車與大車之速

度後，再進行使用者均衡計算。

2.6 DynaTAIWAN 架構

民國 92 年 3 月交通部運輸研究所委託逢甲大學交通工程與管理學系執行「區域級智慧型運輸系統示範計畫-核心交通分析與預測系統(第一年期)」研究計畫，此一計畫主要開發一能反應本土化車流、具備描述性(Descriptive)與規範性(Normative)之核心交通分析與預測系統，此系統即為 DynaTAIWAN(Dynamic Traffic Assignment and Information in Wide Area Network)。DynaTAIWAN 系統之基本功能需求歸納為五大項目：系統考量之交通特性、本土化系統特色、系統分析與預測功能、電腦軟硬體限制與考量以及外部應用。系統架構分為模擬層與即時控制層，其整體架構可參考圖 2.3。

DynaTAIWAN 主要是以中觀車流為理論基礎進行交通模擬，系統中包含車流、行為模式之運作以及路徑之計算；。此外，對於路段移動(Link Movement)、路口轉向控制(Node Transfer)、機車車流模擬、交控措施以及延滯時間計算等均進行細部的考量與處理。為能描述真實車流狀況，DynaTAIWAN 考量了不同的道路型態以及車種分類，其中路網道路型態分為四個種類，車種分為八類，說明如下：

1. 道路型態：

- (1) 高速公路，可分為高速公路路段、上匝道及下匝道
- (2) 快速公路，如中彰快速公路
- (3) 一般道路，可分為無中央分隔島，有中央分隔島但無快慢分隔，有中央分隔島且有快慢分隔及機車專用道

2. 車種

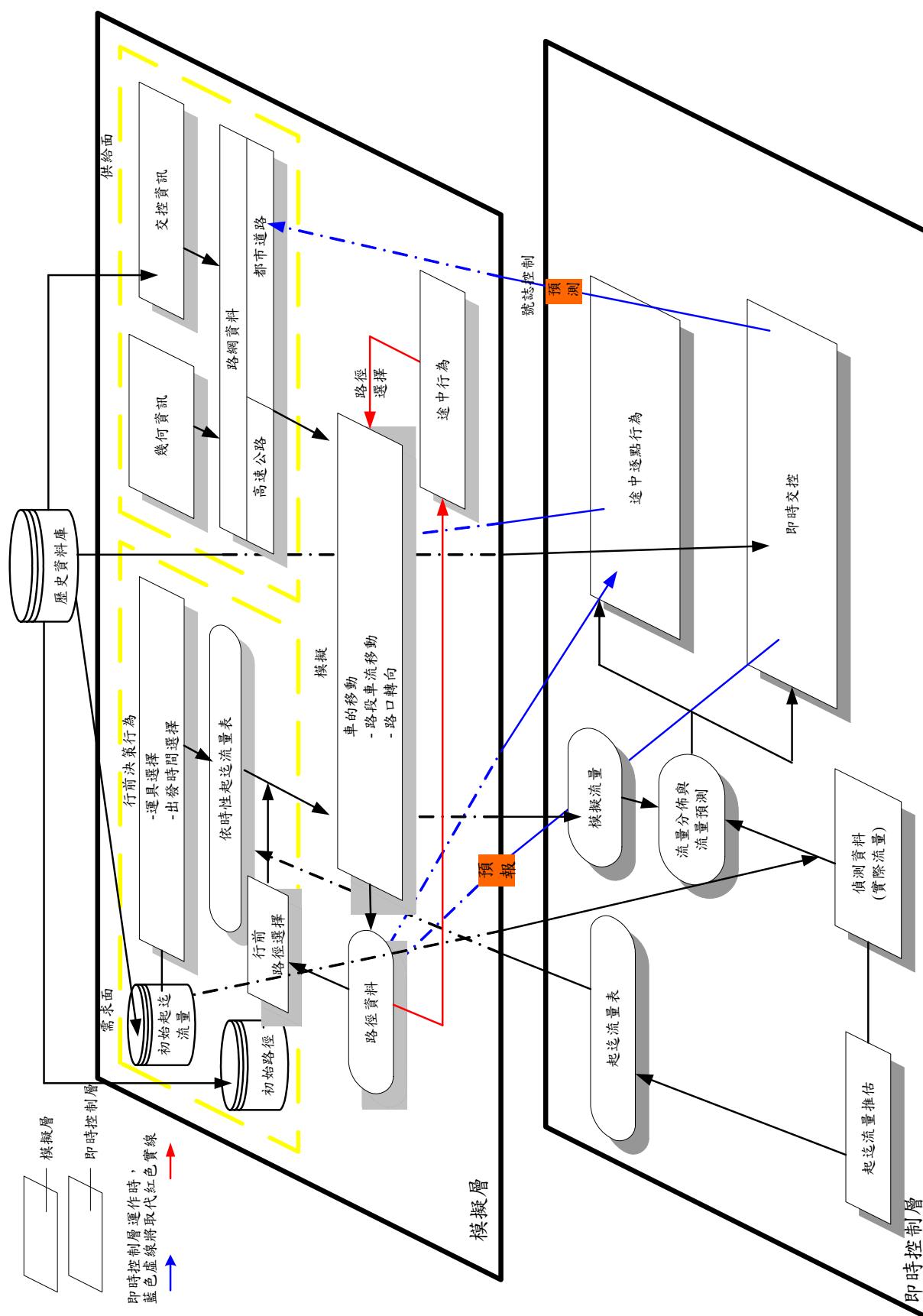
- (1) 未裝有車上接收資訊設備之一般小客車
- (2) 未裝有車上接收資訊設備之大型車
- (3) 未裝有車上接收資訊設備之高乘載小客車

- (4) 車上裝有接收資訊設備之一般小客車
- (5) 車上裝有接收資訊設備之大型車
- (6) 車上裝有接收資訊設備之高乘載小客車
- (7) 機車
- (8) 公車

即時層則是主要透過現有模擬資料並配合即時資料(如：偵測器蒐集而來之路段流量)進行交通流量推估與預測，發展出一個適合於即時交通資訊下分析方法，而其預測方法主要使用滾動平面方法(Rolling Horizon Approach)及動態 OD 推估分析方法進行資料更新與預測。

由以上資訊可歸納整理出 DynaTAIWAN 之核心模擬系統主要特性以下幾點：

1. 此核心交通分析與預測系統考量本土化車流型態，將機車車流加入考量及模擬，模擬包含機車兩段式左轉及機車於路口之紓解狀況。
2. 系統中車流模式、旅運決策行為之模式與參數，係依照國內交通環境調查校估與構建，使其較能反應出國內交通行為特性。
3. 建構之核心交通分析與預測系統，未來可提供給交通相關單位(如高速公路局、各縣市交通局等)進行交通模擬與預測使用，以提高交通規劃與預測之能力。



資料來源：(運研所，2004)

圖 2.3 DynaTAIWAN 系統架構圖

第三章 動態交通指派模式架構及演算流程

本研究主要目的為構建一混合車流下以使用者均衡原則進行指派之動態交通量指派模式。本章針對模式之架構與使用之演算方法進行說明。其中，3.1 節說明模式的架構，又分兩節進一步說明，包含 3.1.1 節介紹模式觀念架構，以及 3.1.2 節說明動態使用者均衡交通指派演算法；3.2 節介紹旅行成本計算方式，3.3 節說明 MSA 演算法，3.4 節介紹依時性最短路徑之演算法，3.5 節說明動態交通指派程式模組及輸入輸出。

3.1 動態交通指派模式

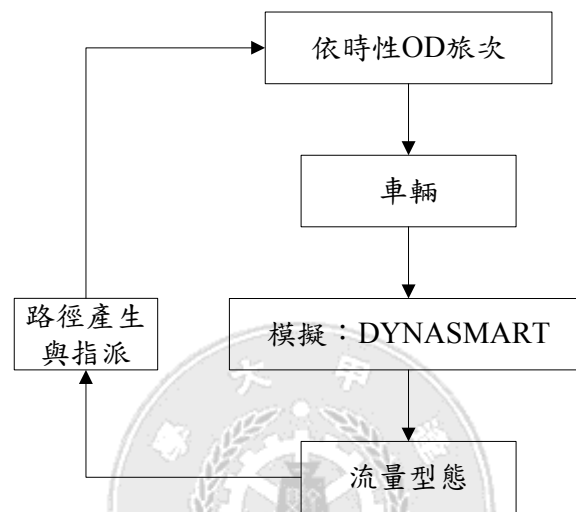
本研究所構建之模擬式動態交通指派模式，主要觀念係依據胡大瀛之研究（胡大瀛，2001）(運研所，2004)所進行，此模式架構主要以模擬方式來取代傳統道路績效函數計算，並取得每一模擬時段下路網中每一路段之流量情形，再依此流量情形進行依時性最短路徑之計算。在此架構下，為考量實際路網環境中混合車流情形，本研究加入實體多車種之觀念。本研究所謂混合車流係指，有兩種以上之運具，可共用路網空間行駛於路網路段上並可能交互影響之車流。本研究所構建之模式架構中，所考量之運具主要依據車輛實體特性分為大車、小車及機車，其中大車包含了公車、卡車等大型車輛。

以下就主要觀念架構於 3.1.1 節，3.1.2 節說明動態使用者均衡演算流程。

3.1.1 動態交通指派模式觀念架構

混合車流下模擬式動態交通指派之觀念架構如圖 3.1 所示。模式在開始進行動態交通量指派時，路網車輛會依照依時性 OD 旅次表產生，車輛路徑可透過指派方式的使用以及個體路徑選擇行為模式之應

用來產生。路網中之車流配合路徑進行模擬移動，以模擬產生流量分布情形，並包含路口延滯以及路段之旅行時間。模式透過模擬產生這些資料來取代以往路段績效函數計算之數值，本研究構建之模式並進一步依據各時段下車流分布分別計算動態旅行成本，再透過 UE 指派原則方式的使用，產生 UE 路徑提供指派車輛使用。



資料來源：(運研所，2004)

圖 3.1 模擬式動態交通指派之觀念架構圖

3.1.2 動態使用者均衡演算法

本小節將針對模擬式動態使用者均衡(Dynamic User Equilibrium, DUE)演算法及動態使用者均衡(DUE)定義進行說明，而動態使用者均衡則是依據 Wardrop 使用者均衡之條件進行修改，其定義為「使用者無法藉由改變選擇出發時間或路徑來達成縮短旅行時間的目的」(胡大瀛，2001)，說明使用者除路徑選擇外，還會選擇不同出發時間來縮短旅行時間。本研究之動態使用者均衡主要定義為「對任何一旅次起迄對而言，當路網達到使用者均衡時，所有被使用之路徑，其旅行成本皆相同，且小於或等於其他未被使用路徑之旅行成本，並且無法藉由改變選擇出發時間或路徑來達成縮短旅行時間的目的」。

Bliemer 在 2003 年針對動態均衡條件之定義為：“For each user-class and for each OD pair, the route travel costs for all users traveling between a specific OD pair and departing at the same time are equal, and less than (or equal to) the route travel costs which would be experienced by a single user on any unused feasible route for that user-class.”，此定義主要延伸 Wardrop 第一準則靜態交通量指派，且路徑和依時性選擇都是依據真實路徑旅行成本。而 Bliemer 將最小路徑旅行成本函數以數學式表示如下：

$$\pi_M^{OD}(k) = \min_{r \in R_m^{OD}} c_M^{rOD}(k) \quad \forall O, D, M, k \quad (3-1)$$

其中，O 代表起點，D 代表迄點，M 代表使用者類別，r 為路徑集合，k 代表某一瞬時時間(Departing at Time Instant)， R_M^{OD} 代表類別 M 從 O 到 D 之所有路徑集合， $c_M^{rOD}(k)$ 代表類別 M 於 k 時間下從 O 到 D 使用路徑集合 r 之路徑旅行成本， $\pi_M^{OD}(k)$ 表車種 M 於 k 時間下從 O 到 D 之最小路徑成本。Bliemer 並將動態多車種交通量均衡以數學方式表示如下：

$$(f_M^{rOD}(k) > 0 \Rightarrow c_M^{rOD}(k) = \pi_M^{OD}(k)), \quad \forall O, D, M, r \in R_M^{OD}, k \quad (3-2)$$

其中 $f_M^{rOD}(k)$ 代表 M 類別於 k 時間下從 O 到 D 使用路徑 r 之路徑流率。

本研究針對多車種下動態使用者均衡條件定義為「對任一起迄對而言，當路網達到使用者均衡時，於指派時段 k 下所有被使用之路徑，其路徑上各車種旅行成本總和皆相同，並且小於或等於其他未被使用到之路徑旅行成本」。公式如下所示(公式中 M 代表車種)：

$$c_M^{rOD}(k) \begin{cases} = \pi_M^{OD}(k), & \text{假如 } f_M^{rOD}(k) > 0 \\ \geq \pi_M^{OD}(k), & \text{假如 } f_M^{rOD}(k) = 0 \end{cases} \quad \forall O, D, r \in R_M^{OD}, M \quad (3-3)$$

以下針對模擬式動態使用者均衡(DUE)模式之演算進行說明，分為初始階段與指派模式運算兩部分。首先針對流程中所使用到之變數進行說明與定義如下：

I 代表遞子迴。T 為指派時段。K 為路徑。XP(O,D,M,T,K,I+1)代

表 M 車種於 $I+1$ 遞子迴更新後路徑之車輛數。 $YP(O,D,M,T,K,I)$ 代表 M 車種於 I 遞子迴新路徑上車輛數。 $XP(O,D,M,T,K,I)$ 代表 M 車種於 I 遞子迴原路徑上車輛數。模擬時段(Simulation Interval)為指使用 DynaTAIWAN 車流模擬每 6 秒(0.1 分鐘) 進行路網狀態之更新，產生各項資料例如：車輛於路段速度、路口延滯時間、使用路徑資料等。指派時段(Assignment Interval)，此指 DUE 指派執行時間，指派時段於本研究設定為三分鐘為一時間區間，即每三分鐘更新每一個路段的旅行成本，如 3 分鐘、6 分鐘等以此類推進行旅行成本更新，如圖 3.2 代表一個三分鐘時間示意圖。

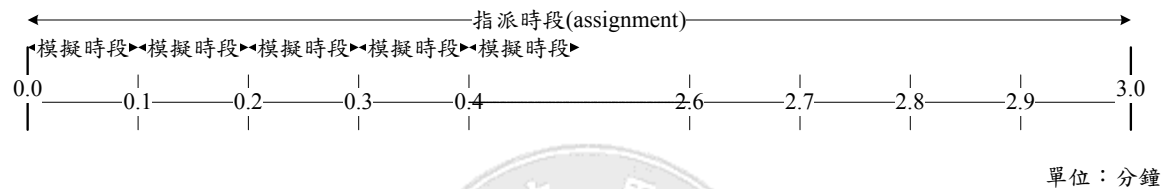


圖 3.2 指派時段與模擬時段

模擬式動態使用者均衡流程如圖 3.3 所示，演算法流程如下：

步驟0：初始階段

此一階段主要利用 DynaTAIWAN 模擬出車輛於路網之交通狀況。DynaTAIWAN 首先讀入相關 Input 資料，如：路徑資料、路網資料及依時性 OD 表，DynaTAIWAN 依據 Input 資料進行交通模擬後，產生本研究模式所需之資料，包含路徑資料、各路徑流量、各路段速度及各車種於路口之延滯資料。

步驟1：歷史路徑資料

將 DynaTAIWAN 模擬所產生之路徑資料作為歷史路徑存入路徑資料。

步驟2：UE 資料讀入

指派模式進行運算時，需將各車種所使用路徑、路徑流量 $XP(O,D,M,T,K,I)$ 與路網資料讀入。

步驟3：旅行成本

讀取 DynaTAIWAN 模擬出之路段速度與路口延滯資料並合併計算出車輛於該路段所使用之旅行成本。

步驟4：依時性最短路徑演算法

依據步驟 3 所產生之旅行成本，計算出車輛於指派時段 T 下各車種依時性最短路徑。

步驟5：路徑資料更新

檢查所產生路徑，如果路徑未在歷史路徑中，則設此一路徑為新路徑。若已包含在歷史路徑中，則表示此階段未有新路徑。

步驟6：MSA 演算法

以 MSA 演算法(Method of Successive Averages)重新分配各路徑之車輛數。

$$XP(O,D,M,T,K,I+1) = \frac{1}{I+1} YP(O,D,M,T,K,I) + (1 - \frac{1}{I+1}) XP(O,D,M,T,K,I) \quad (3.4)$$

步驟7：收斂測試

其收斂測試為遞子迴 I 階段下流量與 I+1 階段流量，即 $XP(O,D,M,T,K,I)$ 與 $XP(O,D,M,T,K,I+1)$ 進行比較，若兩者差大於 0.05 則給予違反值，收斂與否則依據所有 OD 對之違反值累計進行判斷。如所有 OD 對之累計違反值小於 50，則此動態使用者均衡視為達到收斂，如違反值大於 50，則再將路網車流以 DynaTAIWAN 進行下一階段之模擬，模擬出 I+1 階段車流模擬資料再進行動態使用者均衡。

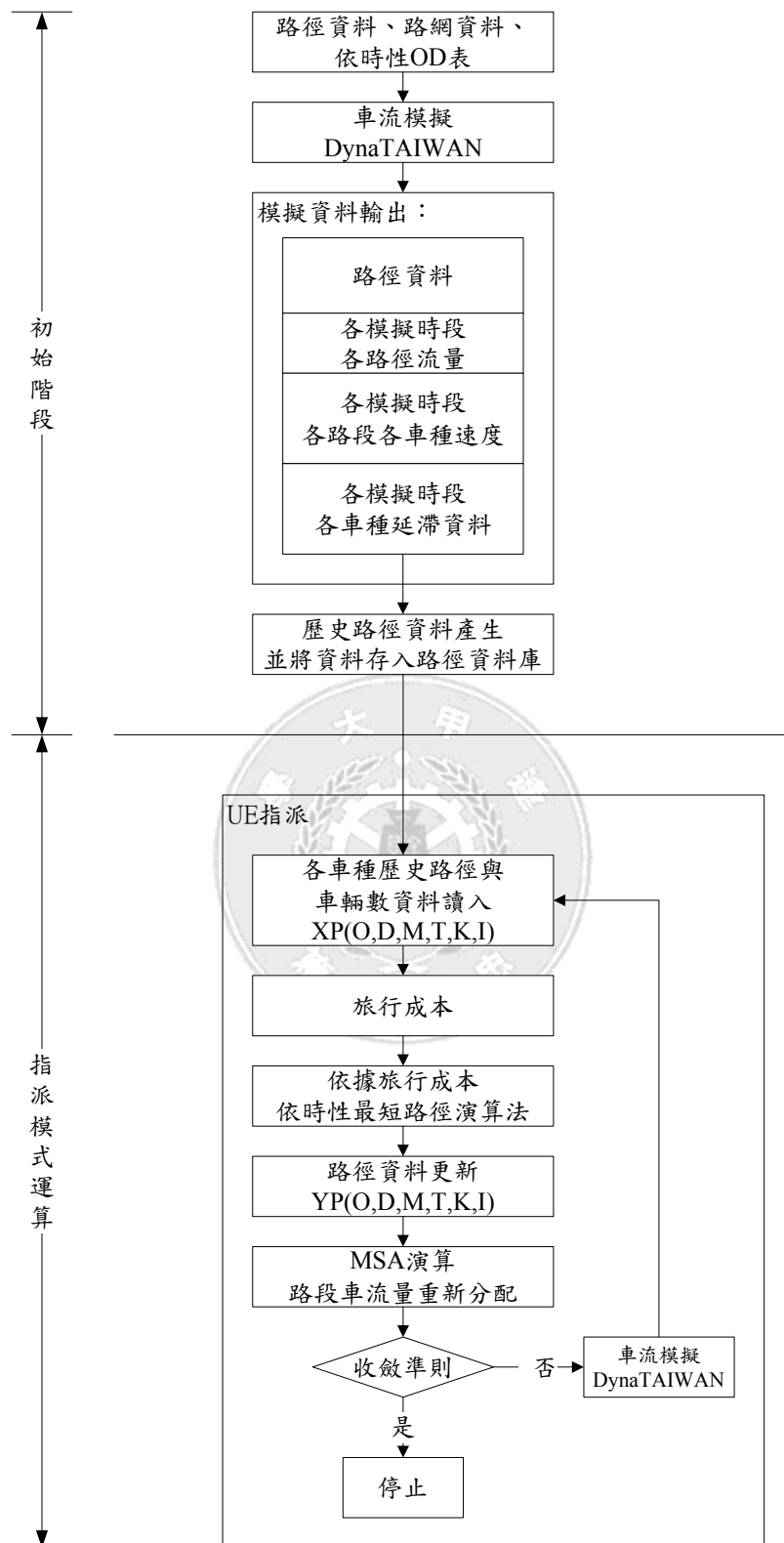


圖 3.3 模擬式動態使用者均衡指派流程圖

3.2 旅行成本

本節針對計算使用者均衡指派時須使用之旅行成本計算進行說明。

本研究中之路段旅行成本計算係依據 DynaTAIWAN 之考量，將旅行時間成本分為路段與路口旅行時間分別計算之，將實際路段區分為路段移動中的車輛與路口停等中車輛，如圖 3.4 所示。

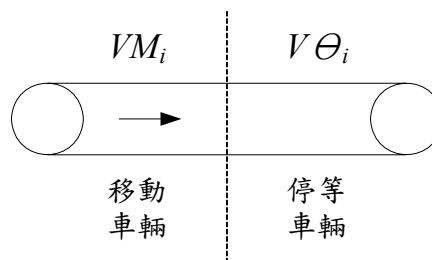


圖 3.4 DynaTAIWAN 旅行時間估計示意圖

1. 路段旅行時間

旅行成本之路段旅行時間係藉由 DynaTAIWAN 中車流模式來產生，DynaTAIWAN 會提供每一個模擬時間下移動區域的密度與速度。因此，本研究構建模式所需之旅行時間可經由路段長度與路段移動速度計算求得。此外，在混合車流多車種考量下，本研究之路段旅行時間又區分為小車、機車與大車之路段旅行時間。以下分別針對各車種路段旅行時間進行說明。

(1) 小車旅行時間

小車之動態旅行時間是依據 DynaTAIWAN 模擬出各模擬時段下各路段所使用之速度，並取得該路段長度後，依據路段旅行時間 = 路段長度 / 路段速度，進而得到各路段之小車旅行時間成本。

(2) 機車旅行時間

DynaTIWAN 考量小車與機車之速度差異，依據不同道路型態給予不同速度差異 β 值，作為路網車流中之路段機車速度，同樣依據旅行時間之計算方式(路段旅行時間= 路段長度 / 路段平均速度)，計算出機車於各路段之旅行時間。其 β 值，如表 3.1 所示。

表 3.1 汽機車速度差異比值 β 值

道路型態	β 值
有中央分隔島與快慢分隔島	0.8
有中央分隔島與無快慢分隔島	0.84
無中央分隔島與無快慢分隔島	0.82

資料來源：(運研所，2004)

(3) 大車旅行時間

大車速度則參考 Bliemer(2003)中所使用大車速度計算方式，依據 大車速度=小車速度 / (1+ γ ×小車速度) 與大車速限兩速度值下選取兩者最小速度為大車速度，公式中 γ 為小車與大車速度之差異值，該值為 0.002 小時/公里，其公式如下所示：

$$V_{\text{大車}} = \min \left\{ \frac{V_{\text{小}}}{1 + \gamma V_{\text{小}}}, V_{\text{大車速限}} \right\} \quad (3-5)$$

2. 路口延滯時間

DynaTAIWAN 以平均紓解率(Average Discharge Rate)作為估算延滯時間之依據，此一計算方式可以反應出路口容量控制，延滯時間表示方法如下所示。

$$\text{延滯時間} = \frac{VQ_i}{AS_i} \quad (3-6)$$

VQ_i ：在路段*i*上的停等車輛數

$$AS_i, : \text{平均紓解率} = \frac{\sum_{m=t}^{t+T} \sum_{j \in \text{OUTBOUND}} AVM_{ij}(m)}{T} \quad (3-7)$$

$AVM_{ij}(m)$ ：在 m 的時段中，實際由 i 移至 j 的車輛總數，這兩者間的時間，可用於路徑的計算。

T ：計算紓解率的時段長度

本研究之延滯時間估算，同樣分為三車種之分別估算，包含大車、小車與機車，依據式 3-6，需取得各路段上各車種之停等車輛數目。本研究考量各車種所使用路段車道之特性，將小車、大車與機車延滯時間區分為汽車延滯時間與機車延滯時間兩種，其中大車與小車均採用汽車延滯時間。於汽車延滯計算中主要計算路段上等候直行與左轉之停等車輛，包含大車與小車車輛數，依據此停等車輛數計算出汽車延滯時間。而機車停等車輛數之計算，係統計待轉區、停等區與路段車道上之機車停等數量，再將機車停等車輛數帶入式 3-6，求得機車延滯時間，研究中機車兩段式左轉之旅行時間暫不納入考量。停等車輛範圍示意圖，如圖 3.5 所示。

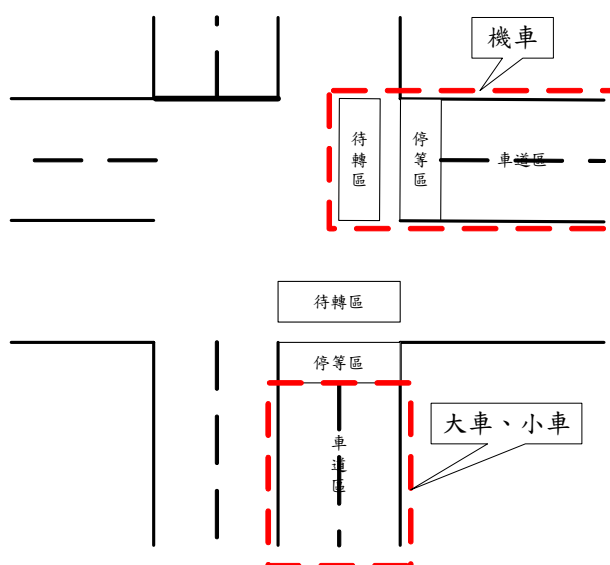


圖 3.5 汽機車停等車輛數示意圖

3.3 MSA 演算法

MSA(Method of Successive Average)演算法係用來指派過程中路徑更新之演算法，主要是用以避免將過多交通量分派至擁擠路段上 (Sheffi, 1985)。

因此，在 3.1.2 節之步驟中，所提及之 $I+1$ 遞子迴路徑流量更新後流量會等於 I 遞子迴新路徑上車輛數乘以 $\frac{1}{I+1}$ 加上 I 遞子迴原路徑上車輛數乘以 $(1-\frac{1}{I+1})$ ，如式 3-8。

$$XP(O,D,M,T,K,I+1) = \frac{1}{I+1} YP(O,D,M,T,K,I) + (1-\frac{1}{I+1})XP(O,D,M,T,K,I) \quad (3-8)$$

本研究所使用之 MSA 演算法進行路徑更新收斂準則，主要參考李達權(2002)中所使用之「違反值」累計總數來判斷，違反值主要是用來決定演算是否該收斂或路徑是否需更新之動作。其收斂準則為：於指派時段 T 下，由起點 O 至迄點 D 於 I 階段使用路徑 K 之 M 車種車輛數 $XP(O,D,M,T,K,I)$ ，與指派時段 T 下由起點 O 至迄點 D 於 $I+1$ 階段使用路徑 K 之 M 車種車輛數 $XP(O,D,M,T,K,I+1)$ 進行比較，若兩者之差與原車輛數 $(XP(O,D,M,T,K,I))$ 之比值大於 0.05 時給一違反值 (違反值為 1)。其收斂準則計算方式詳列如下列步驟：

步驟1：確認路徑流量變化

確認方式為本次遞子迴路徑與上一個遞子迴路徑流量變化大於 0.05 則給一個違反值，如果沒有就不需要給予違反值。

$$\frac{|XP(O,D,M,T,K,I+1) - XP(O,D,M,T,K,I)|}{XP(O,D,M,T,K,I)} \geq 0.05 \quad (3-9)$$

步驟 2：違反值計算

當路徑流量變化超過0.05，給該路徑K一個違反值，此一違反值為 1，統計I+1階段OD對下路徑K之各車種違反值累計值。

步驟 3：收斂與否

如該階段累計違反值大於50，則進行到下一個遞迴。如小於50，則視為滿足收斂條件。

3.4 依時性最短路徑演算法

本研究將依循傳統最短路徑計算方法並加入時間之變數，計算出符合當時車流之依時性最短路徑演算法。此演算法主要以林蔚明(2004)所建立之依時性路徑演算法為基礎，其演算法已針對LSA(Label Setting Algorithm)與LCA(Label Correcting Algorithm)兩演算法進行修正，並加入路口號誌造成之延滯成本，使其路徑較符合真實世界交通狀況。依時性最短路徑所使用之未來時間，主要由DynaTAIWAN將所有時段下各路段之平均率模擬輸出後，依時性最短路徑之演算法則依據所模擬出之路段平均速率換算出各時間點下之旅行時間。

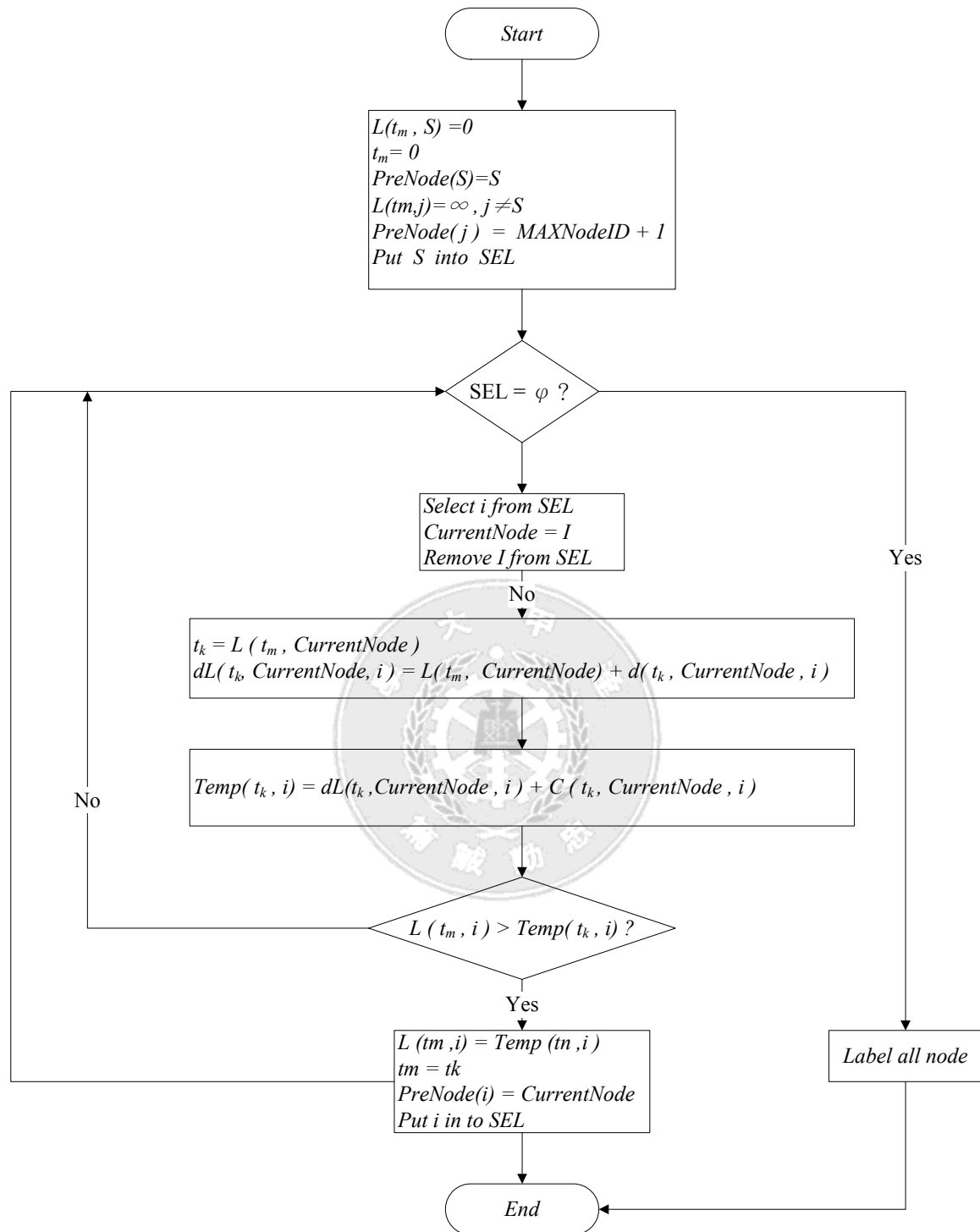
為說明林蔚明(2004)論文中 LCA 流程步驟，在此先說明參數定義：

- (1) 路網 $G(V, E)$ ： V 代表所有節點之集合、 E 代表所有節線之集合；
- (2) S ：表示起點；
- (3) $C(t_k, i, j)$ ：時間點 t_k 下任意兩相鄰節點 i 、 j 之路段旅行時間成本；
- (4) $L(t_m, CurrentNode)$ ：累積之旅行時間成本標號值；
- (5) $PreNode(i)$ ：點 i 之上游節點；
- (6) $CurrentNode$ ：處理中之節點；
- (7) $Temp(t_k, i)$ ：暫存之節點 i 標號值；
- (8) SEL ：待選取節點之集合；

- (9) $d(t_k, CurrentNode, i)$: 處理中節點 $CurrentNode$ 的路口延滯成本;
- (10) $dL(t_k, CurrentNode, i)$: $CurrentNode$ 的累積旅行時間成本標號值加上 $CurrentNode$ 到 i 路口延滯成本;

演算步驟如下:

- Step 0: 設定所有點之初始值: 起點 S 之標號值 $L(t_m, S) = 0$, $t_m = 0$, 起點 S 之上游節點 $PreNode(S) = S$, 路網上所有點 i ($i \neq S$) 之標號值 $L(t_m, i) = \infty$, 點 i 之上游節點 $PreNode(i)$ 為最大節點編號+1。
- Step 1: 將 S 置入 SEL 中。
- Step 2: 檢查 SEL 是否為空集合,
是, 到 Step 8;
否, 到 Step 3。
- Step 3: 依據 FIFO 的規則從 SEL 中選一點 i , 設 $CurrentNode = i$, 從 SEL 中移除 i 。
- Step 4: 計算 $CurrentNode$ 之路口延滯成本 $d(t_k, CurrentNode, i)$, 此時 $t_k = L(t_m, CurrentNode)$, $CurrentNode$ 到 i 之起始標號值 $dL(t_k, CurrentNode, i) = L(t_m, CurrentNode) + d(t_k, CurrentNode, i)$ 。
- Step 5: 取得所有與 $CurrentNode$ 相連之下游點 i 之路段成本 $C(t_k, CurrentNode, i)$, 並計算 $Temp(t_k, i) = dL(t_k, CurrentNode, i) + C(t_k, CurrentNode, i)$ 。
- Step 6: 檢查 $L(t_m, i)$ 是否大於 $Temp(t_k, i)$,
是, 到 Step 7;
否, 回到 Step 2。
- Step 7: 更新點 i 之標號值 $L(t_m, i) = Temp(t_k, i)$, 且使 $t_m = t_k$, 點 i 之上游節點為 $CurrentNode$, $PreNode(i) = CurrentNode$, 將點 i 置入 SEL 中, 到 Step 2。
- Step 8: 將所有點標記, 結束。



資料來源：(林蔚明，2004)

圖 3.6 路口延滯之依時性最短路徑流程圖

3.5 模擬式動態交通指派程式結構與流程

本節將針對模擬式動態交通指派開發模組進行說明，3.5.1 節先說明模擬式動態交通指派程式模組架構，3.5.2 節介紹 DynaTAIWAN 模組之輸入輸出，3.5.3 節針對成本計算模組之輸入輸出說明，3.5.4 節介紹依時性最短路徑模組、3.5.5 節說明路徑流量分配模組之輸入輸出，3.5.6 節說明車輛產生模組如何產生下一遞子迴模擬資料。

3.5.1 模擬式動態交通指派程式模組

模擬式動態交通指派程式架構，如圖 3.7 所示。整體程式架構主要由五個模組所組成，包含 DynaTAIWAN 模組、成本計算模組、依時性最短路徑模組、路徑流量分配模組及車輛產生模組。於模擬式動態交通量指派運算時，首先由 DynaTAIWAN 模組進行初步車流交通模擬，透過 DynaTAIWAN 模組輸出交通模擬資料，此模組輸出資料分別提供給成本計算模組、路徑流量分配模組及車輛產生模組使用。

成本計算模組負責路段旅行成本計算，主要在讀取 DynaTAIWAN 所模擬之各路段速率後，換算成旅行時間，再加入延滯時間作為依時性最短路徑計算依據。依時性最短路徑模組則在取得更新後之旅行成本後，進行依時性最短路徑計算，再將依時性路徑輸出至依時性路徑資料中。

路徑流量分配模組主要負責路徑流量分配計算，此路徑流量資料需取得依時性最短路徑模組之依時性路徑資料與 DynaTAIWAN 所模擬出各路徑流量，再依據 MSA 演算法進行流量分配。

車輛產生模組主要負責路徑流量分配模組所產生結果資料，配合 DynaTAIWAN 所模擬之車輛屬性資料，提供下一遞子迴 DynaTAIWAN 進行下一階段之模擬。

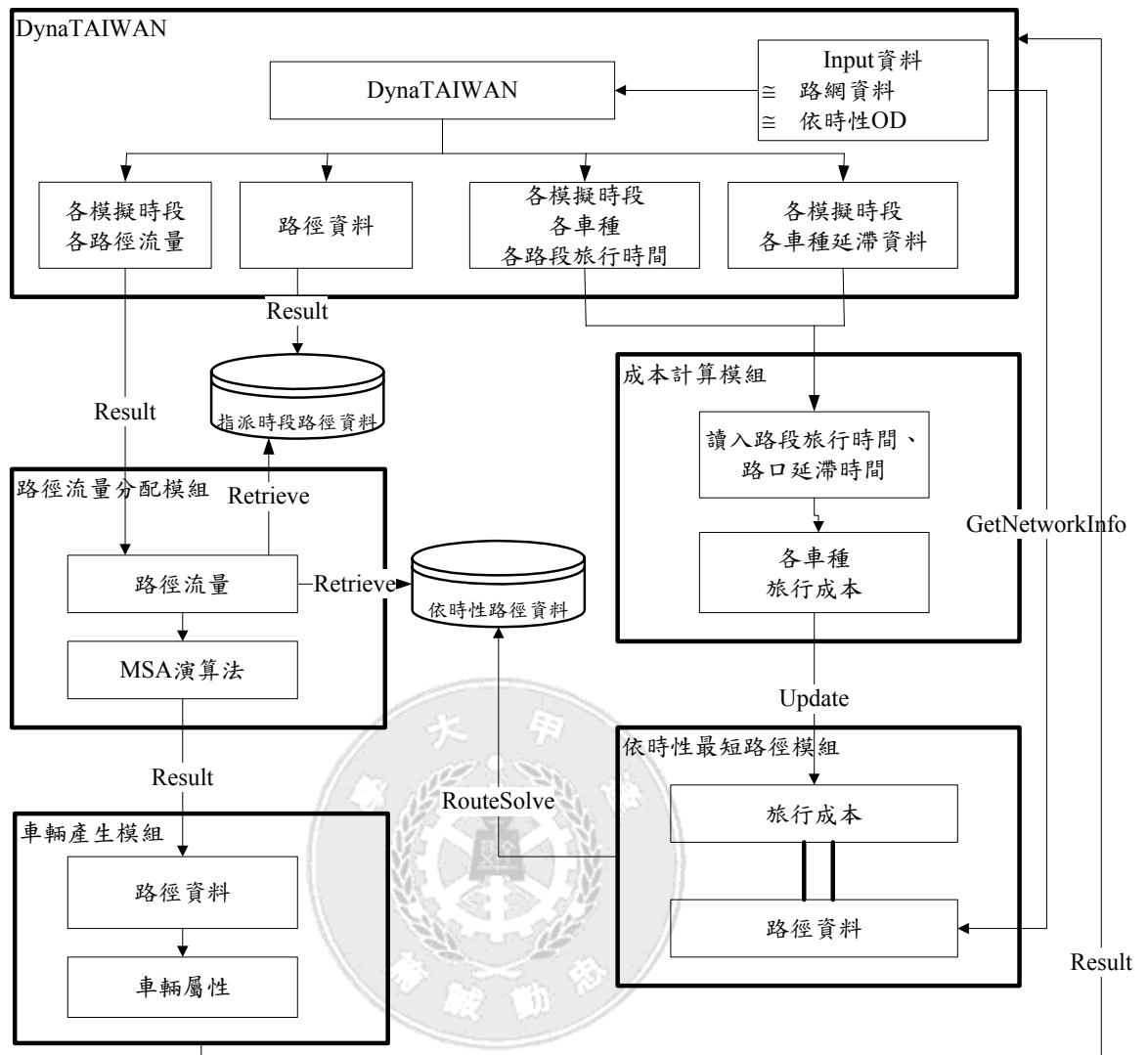


圖 3.7 模擬式動態交通量指派程式架構圖

3.5.2 DynaTAIWAN 模組

DynaTAIWAN 模組主要負責模擬混合車流下之車輛速度、流量及路徑。模組運作方式為 DynaTAIWAN 讀入 Input 資料，如路網資料與依時性 OD 資料，如圖 3.8 所示，DynaTAIWAN 則依據此資料進行交通模擬，模擬產生資料包含各模擬時段下各車種路段旅行時間與路口延滯資料，此資料主要提供給成本計算模組所使用。

除時間輸出外還有路徑相關資料需輸出，包含各模擬時段下所使用路徑及各路徑下各車種之流量，其中 DynaTAIWAN 所產生路徑資

料將提供給路徑資料，此路徑資料主要負責儲存於模擬過程中各時段各車種所使用之路徑資料，包含了模擬時間、起迄點、車種及所使用路徑節點編號。資料處理方式為將每三分鐘內(如 0-180 秒)路徑資料整理為指派時段路徑資料，將此時段內所使用之路徑依照時間、起迄點之大小由小至大排列。

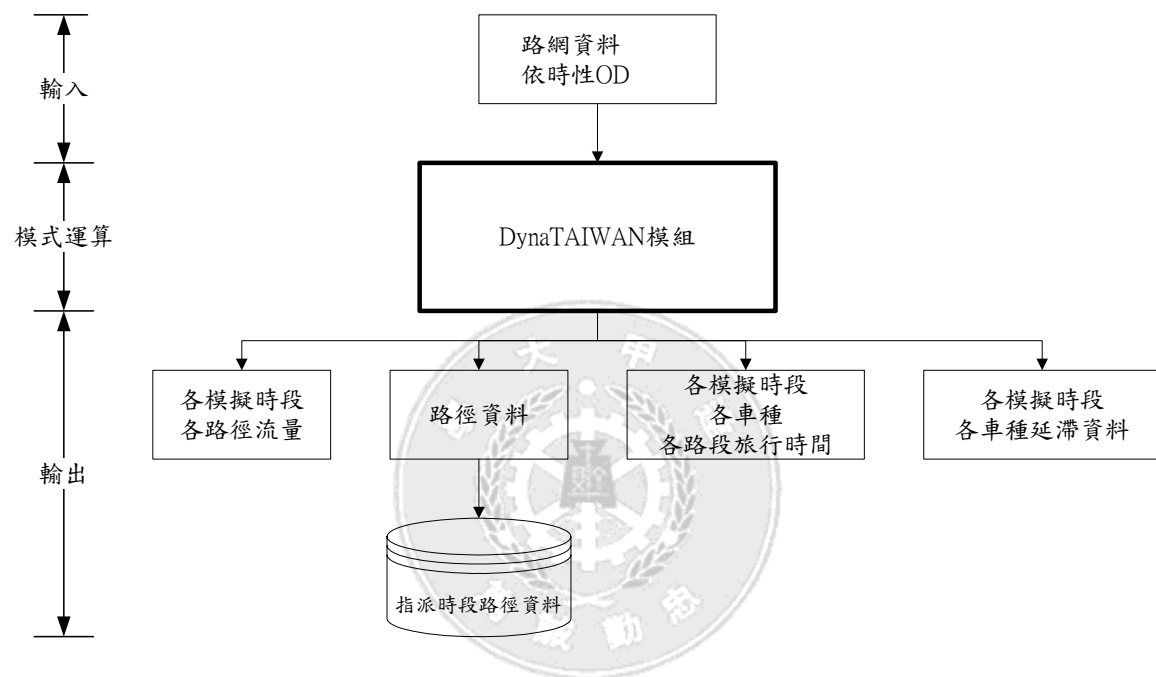


圖 3.8 DynaTAIWAN 模組輸入輸出

3.5.3 成本計算模組

此模組負責動態使用者均衡(DUE)指派過程中依時性最短路徑所需要旅行時間成本計算，主要將 DynaTAIWAN 模組所輸出之各車種於各路段上車輛平均速率讀入計算出各路段旅行時間，如圖 3.9 所示。

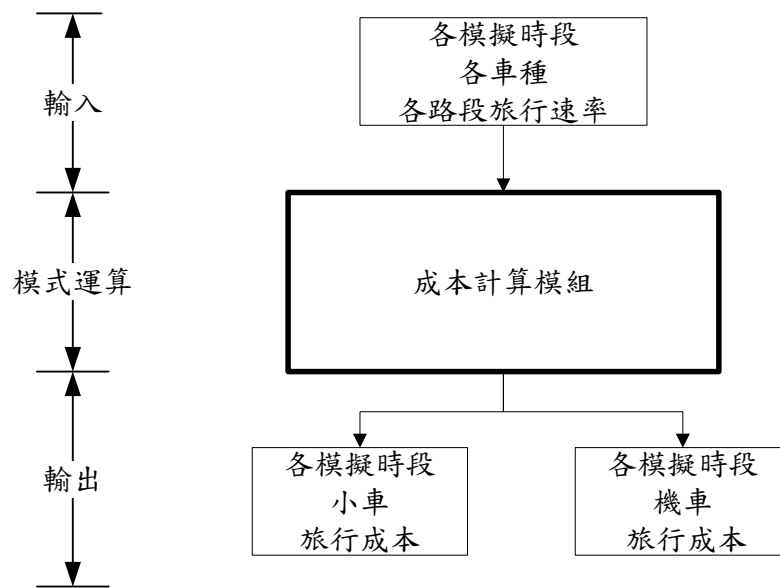


圖 3.9 成本計算模組輸入輸出

3.5.4 依時性最短路徑模組

依時性最短路徑模組主要負責產生動態使用者均衡所需路徑資料，因其為依時性資料，故於路徑計算時需取得各時段下各車種之路段旅行成本及路網資料。路徑模組依據指派時段間隔，每三分鐘取得動態旅行成本，再加入延滯時間，計算出該指派時段下各車種所使用依時性最短路徑，並將所產生路徑輸出至路徑資料，如圖 3.10 所示。

依時性最短路徑資料每一筆輸出格式，如圖 3.11 所示。資料格式存放分別以內、外層資料結構方式來存放。外層結構主要是負責資料結構大小，由 0 開始編號至資料最後一筆編號順序；內層資料結構負責存放依時性最短路徑資料，資料說明如下。

指派時段為將模擬時段以每三分鐘(0~180 秒)為一時段區間，例如模擬時間 0 到 180 秒為第一個指派時段，而指派時段選取 0 秒來代表第一個指派時段。模擬時間 180 到 360 秒為第二個指派時段，則以 180 秒來代表第二個指派時段，以此類推後續指派時段。車輛數則設定為 0，其主要原因為方便流量重新分配時計算使用。節點數則代表所使用路徑中有幾個節點。節點和為車輛所使用之路徑節點編號之總

和。此兩部分資料主要負責於依時性最短路徑與模擬時段所使用路徑進行比較時所需之資料，利用此資料判斷依時性最短路徑是否為新路徑。路徑比較規則為比較指派時段、起迄點是否相同，是則進入比較節點數，如節點數不同則代表新路徑，是則需比較節點和。如節點和不同，則表示為新路徑，如相同則需進行路徑比較。

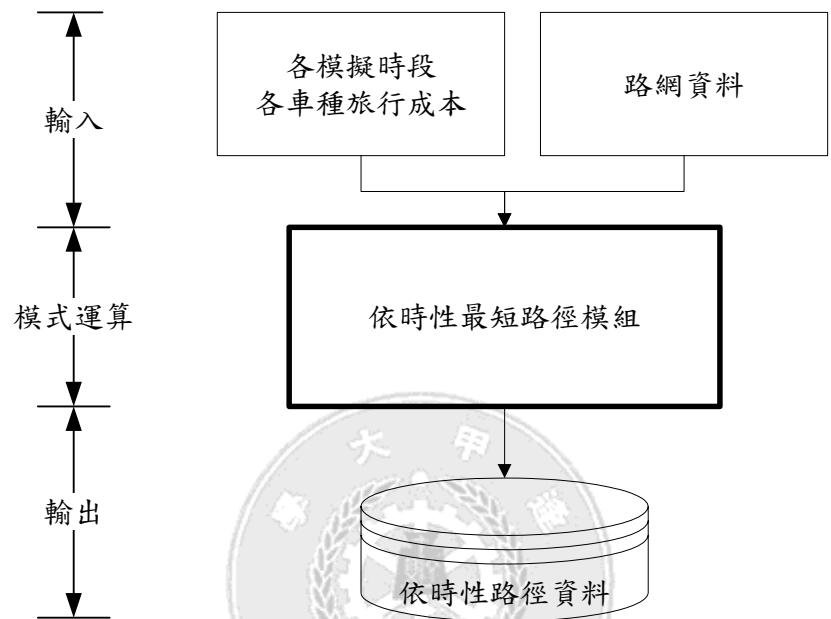


圖 3.10 依時性最短路徑模組輸入輸出

Vector 編號	指派 時段	起點	迄點	車輛 數	節點 數	節點 和	路徑			
0	0	1	2	0	2	3	1	2		
366	360	2	5	0	3	51	2	44	5	
3456	1980	17	5	0	3	33	17	11	5	
4467	2880	6	30	0	5	90	6	12	18	24 30
.

圖 3.11 依時性最短路徑資料格式

3.5.5 路徑流量分配模組

路徑流量分配模組在指派過程中主要是負責將車輛所使用路徑進行流量重新分配，模組運算程序為讀入依時性最短路徑資料與指派時段下路徑流量資料，再配合 MSA 演算法來完成流量分配，如圖 3.12 所示。其中，指派時段下路徑流量資料格式，如圖 3.13 所示。路徑流量資料格式可以區分為內層與外層資料格式，外層主要負責資料筆數編號，為快速取得同一組起迄點之相關資料，如車輛數、路徑等。而內層資料格式則切割為兩部份以存放個別資料，第一部分主要負責存放指派時段、路徑起迄點，第二部份之資料結構負責存放路徑編號、使用該路徑之車輛數、節點數、節點和、路徑機率值及路徑。其中資料結構中之機率值計算方式為該路徑車輛數除以起迄點所有車輛數後，累加同一起迄點之路徑機率值。

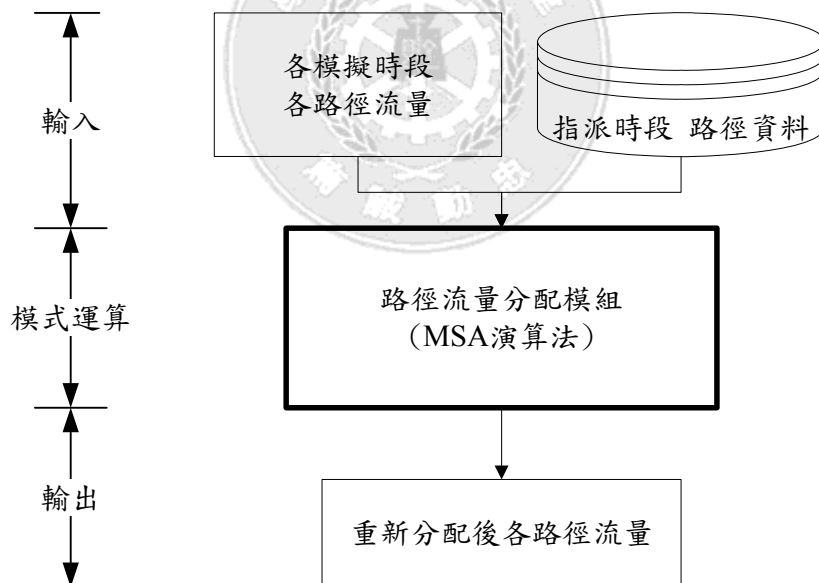


圖 3.12 路徑流量分配輸入輸出

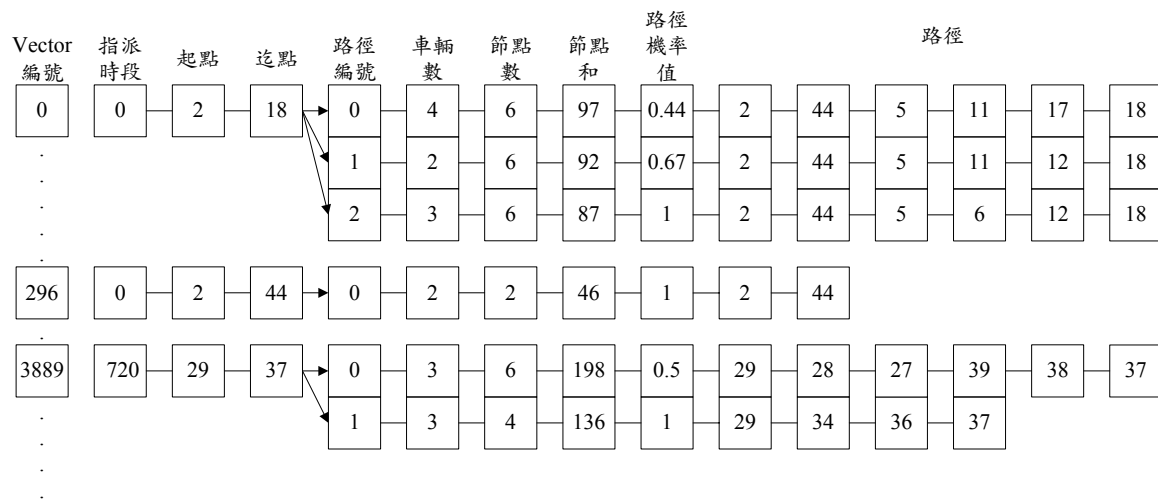


圖 3.13 指派時段下路徑流量資料格式

模組在讀入所需資料後，依據 MSA 演算法將各新產生路徑與原路徑流量進行重新分配計算，如式 3-10 所示。

$$XP(O, D, M, T, K, I+1) = \frac{1}{I+1} YP(O, D, M, T, K, I) + (1 - \frac{1}{I+1}) XP(O, D, M, T, K, I) \quad (3-10)$$

圖 3.14 為路徑流量分配流程，包含下列步驟：

- 步驟1：讀入依時性最短路徑資料。
- 步驟2：讀入各指派時段下各路徑及流量資料。
- 步驟3：判斷指派時段下第 i 筆起迄點與依時性路徑第 j 筆資料之起迄點是否相同。是相同則進入步驟 4。起迄點不相同則回到步驟 3。
- 步驟4：判斷指派時段下第 i 筆路徑與依時性路徑第 j 筆路徑是否相同，並依據 MSA 公式重新計算路徑流量。
- 步驟5：確認指派時段下第 i+1 筆是否存在？是，則進入步驟 6；否，則停止。
- 步驟6：判斷指派時段下第 i 筆與第 i+1 筆起迄點是否相同。否，則回到步驟 2。是，進行新路徑流量計算並將新路徑存入至路徑流量資料格式。

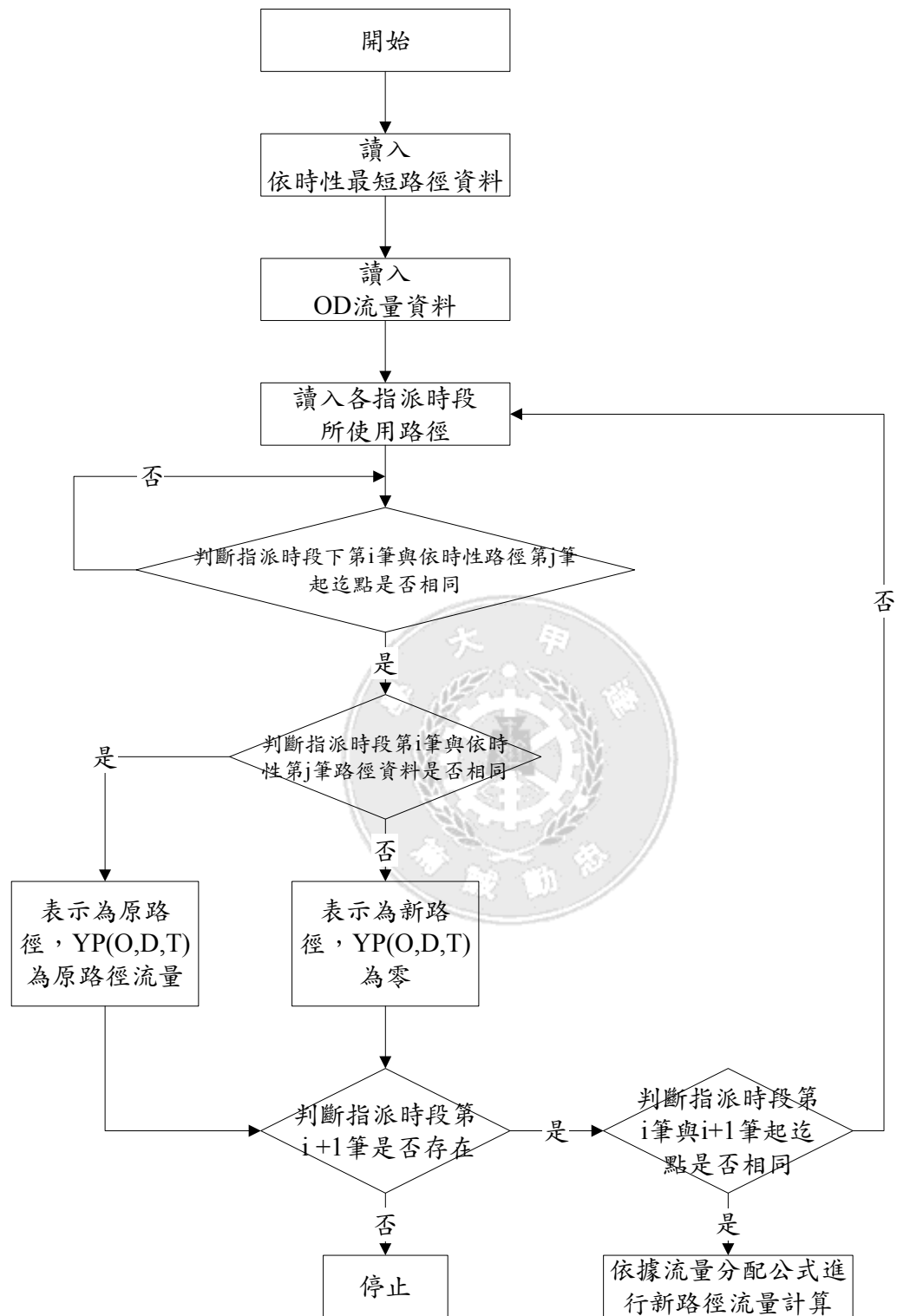


圖 3.14 路徑流量分配流程圖

3.5.6 車輛產生模組

此模組主要負責於 DynaTAIWAN 模擬過程中車輛產生時所需相關資料，如圖 3.15 所示。本模組使用第 I 遞子迴資料作為歷史資料，提供 I+1 遞子迴 DynaTAIWAN 資料進行模擬。資料包含車輛屬性、路徑資料及路徑流量等，其所需資料格式如圖 3.16 所示。此模組中資料包含車輛編號，車輛屬性(1 代表小車)，行為屬性及熟悉度，以及進入路網時間(單位：秒)，起、迄點編號，車輛數，節點數、節點和以及路徑。其資料存放以 vector 方式存放，模組亦將其資料分為內、外層，外層為資料 vector 編號，內層為存放車輛相關資料，如車輛編號、車輛屬性。

此模組將新產生路徑以隨機方式把路徑分配到 DynaTAIWAN 所輸出之原始車輛屬性檔案，其演算步驟流程如下：

步驟 1：讀入經流量分配公式所計算出各指派時段下各路徑之流量資料。

步驟 2：讀取 DynaTAIWAN 所輸出之車輛屬性檔案。

步驟 3：判斷車輛屬性檔中車輛產生時間是否介於指派時段，以及兩個檔案中起迄點是否相同。

是，進入步驟 4。否則回到步驟 2。

步驟 4：隨機產生車輛機率值，並判斷該機率值介於指派時段下哪一條路徑機率間。

步驟 5：判斷該指派時段下路徑流量是否為 0。

否，不為 0，則進行路徑更換，並將流量減 1。

是，則回步驟 4。

步驟 6：確認是否為檔案結尾。

是，則停止讀取；否則回到步驟 2。

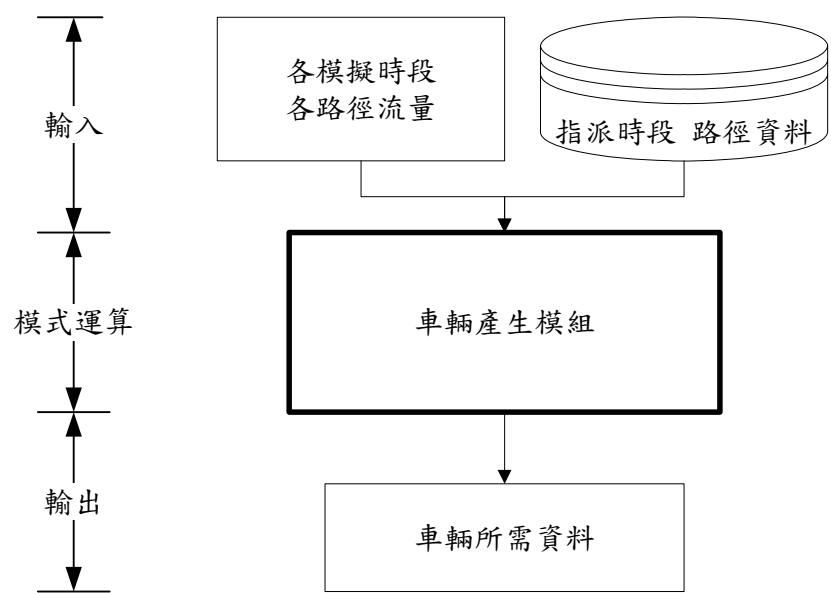


圖 3.15 車輛產生模組輸入輸出

Vector 編號	車輛 編號	車輛 屬性	行為 屬性	熟悉 度	進入 路網 時間	起點	迄點	車輛 數	節點 數	節點 和	路徑		
257	431	1	2	0	40	44	5	1	2	49	44	5	
356	447	1	2	0	46	5	44	1	2	49	5	5	
9765	9554	1	0	1	930	25	35	1	3	91	25	31	35
.

圖 3.16 車輛產生屬性資料格式

第四章 DynaTAIWAN 數值測試

本研究以 DynaTAIWAN 系統進行交通模擬，透過此一軟體模擬取得 UE 指派所需的資料(如指派過程為求路段旅行成本所需之路段速度)。因此本章先對 DynaTAIWAN 系統進行路網測試，以了解其模擬資料情形。本測試以 50 節點為測試路網，4.1 節先針對數值測試環境進行說明，4.2 節介紹 50 節點路網及相關屬性，4.3 節說明實驗目的與不同實驗設計結果說明。

4.1 數值測試環境

DynaTAIWAN 系統主要以 Microsoft Visual C++6.0 進行開發，且具備物件導向之特性。本研究所進行之測試環境規格如下：

軟體規格：

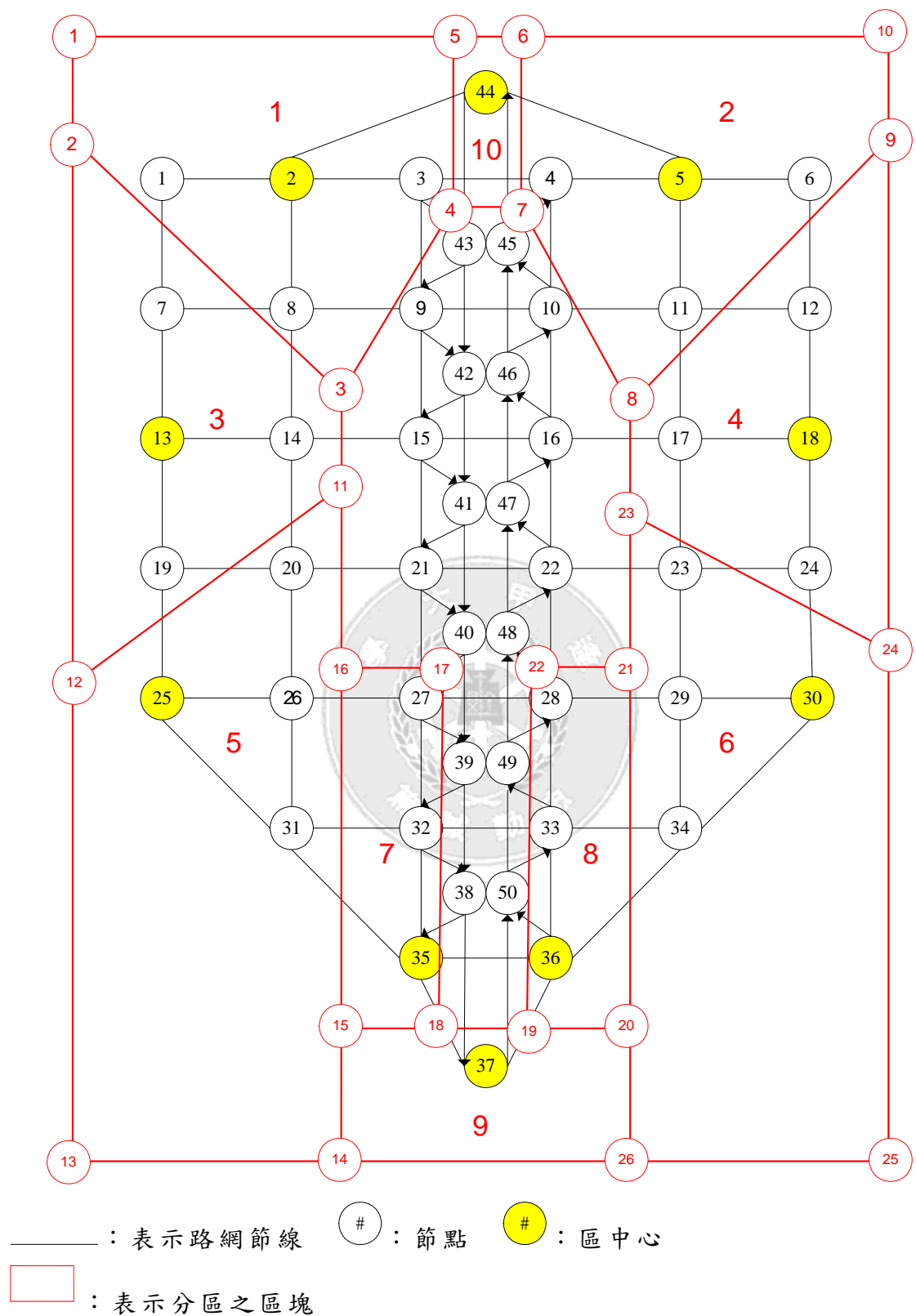
1. DynaTAIWAN；
2. Microsoft Windows XP。

硬體規格：

1. COMPAQ Presario X1000；
2. CPU：Inter Pentium M 1.4GHz；
3. RAM：512MB。

4.2 測試路網-50 節點路網

本實驗所使用之 50 節點如圖 4.1 所示，此路網由 50 節點所組成，並分為 10 個交通分區，172 條節線，路網中包含高速公路、匝道、市區道路等道路型態，道路速限係依據各道路型態一般規定速限所設定，號誌控制則有無號誌與定時號誌控制兩種型態。



資料來源：(運研所，2004)

圖 4.1 50 節點路網圖

道路相關屬性資料設定如表 4.1 所示，高速公路與市區道路長度設定為 500 公尺，上、下匝道則設定為 250 公尺，其中市區道路屬性再區分為三種道路型態，如無快慢且無中央分隔島、無快慢分隔島有中央分隔島、有快慢分隔島且有中央分隔島等三種型態。

表 4.1 路網屬性設定值對照表

屬性	項目	假設值
節線屬性	長度	高速公路：500 公尺 一般都市：500 公尺 匝道：250 公尺
	速限	高速公路：100 公里/小時 上匝道：40 公里/小時 下匝道：40 公里/小時 市區道路(型態 6)：50 公里/小時
	道路屬性	高速公路 上匝道 下匝道 市區道路（有中央無快慢）
	飽和流率	0.5veh/sec
背景流量設定 (起迄點流量需求)	每一分區至路網中其他分區	50 部/每 5 分鐘
號誌設定	無號制 定時號誌	
參數/情境設定	最長模擬時間	300min
	輸出路段結果模擬區段	0.1min

資料來源：(運研所，2004)

4.3 數值路網實驗設計

本節針對 50 節點路網數值實驗設計測試方式進行說明，考量在不同流量需求下更改增量因子進而觀察路網中車流分佈狀態，以下針對實驗測試所使用之增量因子進行說明，增量因子為 DynaTAIWAN 控制路網中車輛數方式，其功能主要為：在一個固定交通路網下，不同

的車輛數呈現出不同的交通狀況。當路網中車輛數越多，交通壅塞狀況越容易產生；車輛數越少時，路網中車流則會較接近自由車流狀況。因此，透過增量因子改變可以觀察路網中車流量分佈與容量。增量因子主要目的為控制路網中車輛數，以觀察於不同增量因子下車流量之變化與影響。

本實驗以均一、高尖峰及常態共三種流量需求分佈分別作為輸入以進行測試(如圖 4.2、圖 4.3、圖 4.4 所示)。增量因子調整範圍由 0.1 開始調整。車輛組成百分比分別為小車設定 40%，大車設定 10%，機車 50%。

此路網數值測試主要目的有下列三點：

1. 測試 50 節點網所能模擬車輛總數以及增量因子可調整之範圍。
2. 觀察不同流量需求下，各車種於路網中旅行時間變化情況。
3. 本實驗結果可做為進行動態使用者均衡指派測試時參考依據，比較動態使用者均衡指派時間變化。

4.3.1 流量需求分佈測試

以下針對三種流量需求分佈產生的方式進行說明：

均一流量需求分佈如圖 4.2 所示，此分佈主要假設各時段各交通分區所產生車輛數相同。小車、機車各產生共 45,000 輛車。

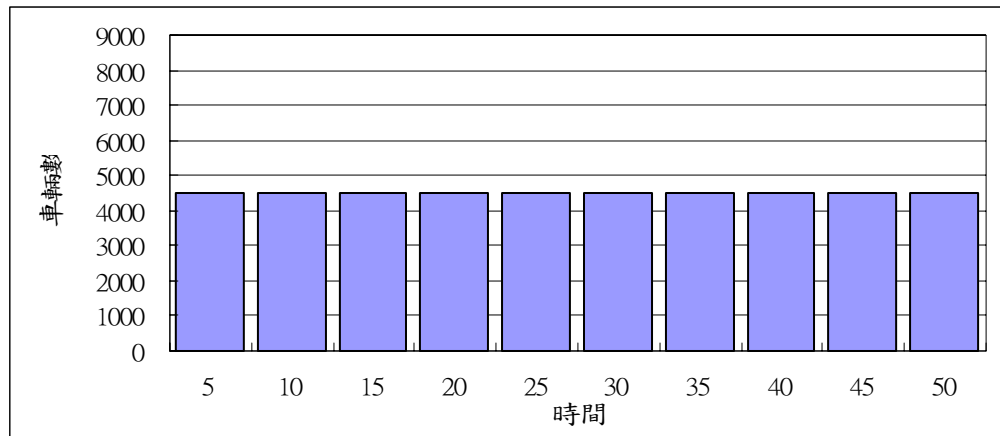


圖 4.2 均一流量需求分佈

高尖峰流量需求分佈如圖 4.3 所示，以均一流量需求分布之車輛數為基礎，則將 5~15 分鐘與 40~50 分鐘之車輛數乘以 0.5 倍，20~35 分鐘之車輛數則乘以 2.0 倍，以便觀察如突然增加車輛數對於路網之影響。在此流量需求分佈下，汽車與機車各分別產生共 42,750 輛。

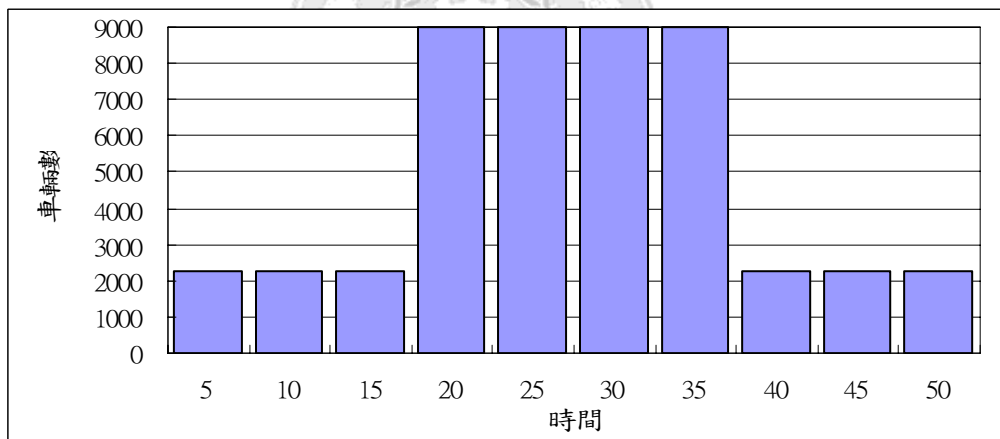


圖 4.3 高尖峰流量需求分佈

常態流量需求分佈如圖 4.4，以均一流量需求分布之車輛數為基礎，假設各時段產生不同車流量分佈，將各模擬時段下車輛數乘以常態機率值，以獲得 10 個時段常態分配流量圖。在此流量需求分佈下，小車、機車各產生共 45,000 輛車。

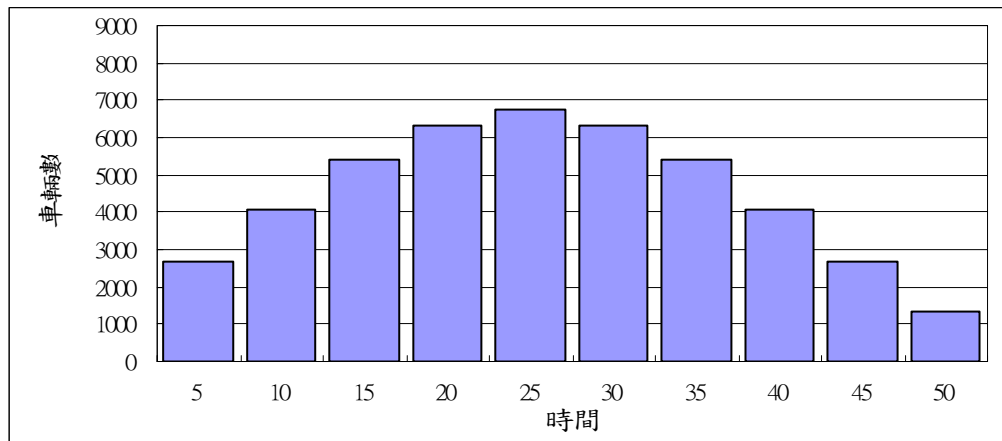


圖 4.4 常態流量需求分佈

以下針對三種流量需求分佈進行旅行時間與旅行距離討論以及實驗綜合討論進行說明。

1. 均一流量需求分佈

均一流量需求分佈下數值實驗結果如表 4.2 所示，小車、大車與機車之車輛數逐漸增加時，其總車輛數由 8,830 輛漸增至 61,879 輛，整體平均旅行時間則由 2.86 分鐘增加至 23.41 分鐘，三車種之旅行時間曲線呈現指數狀態曲線。再將三車種分別觀察可發現，小車旅行時間由原本 2.8 分鐘增加到 32.36 分鐘，大車則由 2.99 分鐘增加至 34.08 分鐘，機車則由 2.9 分鐘增加到 13.92 分鐘，由以上可以觀察到，當路網中車輛數不斷增加時，車輛於路網中的旅行時間會明顯增加。此外，各車種之旅行距離如圖 4.5 所示，當車輛數增加時，各車種之旅行距離亦會呈現小幅度增加。

表 4.2 均一流量需求分佈下系統績效統計表

增量因子	總車輛	平均旅行時間 (分鐘)	小車	小車 平均旅行時間 (分鐘)	小車 平均旅行距離 (公里)	大車	大車 平均旅行時間 (分鐘)	大車 平均旅行距離 (公里)	機車	機車 平均旅行時間 (分鐘)	機車 平均旅行距離 (公里)
0.1	8,830	2.86	3,747	2.80	2.38	668	2.99	2.41	4,415	2.90	1.58
0.2	17,678	3.19	7,244	3.26	2.38	1,595	3.47	2.42	8,839	3.09	1.58
0.3	26,554	3.63	10,534	3.92	2.39	2,743	3.83	2.36	13,277	3.35	1.59
0.4	35,338	4.61	14,060	4.28	2.41	3,609	4.38	2.44	17,669	3.93	1.60
0.5	44,206	7.09	17,607	8.48	2.44	4,496	8.99	2.47	22,103	4.60	1.67
0.6	53,005	13.71	21,267	17.88	2.43	5,223	19.78	2.46	26,515	9.17	1.69
0.7	61,879	23.41	24,961	32.36	2.44	5,993	34.08	2.46	30,925	13.92	1.71

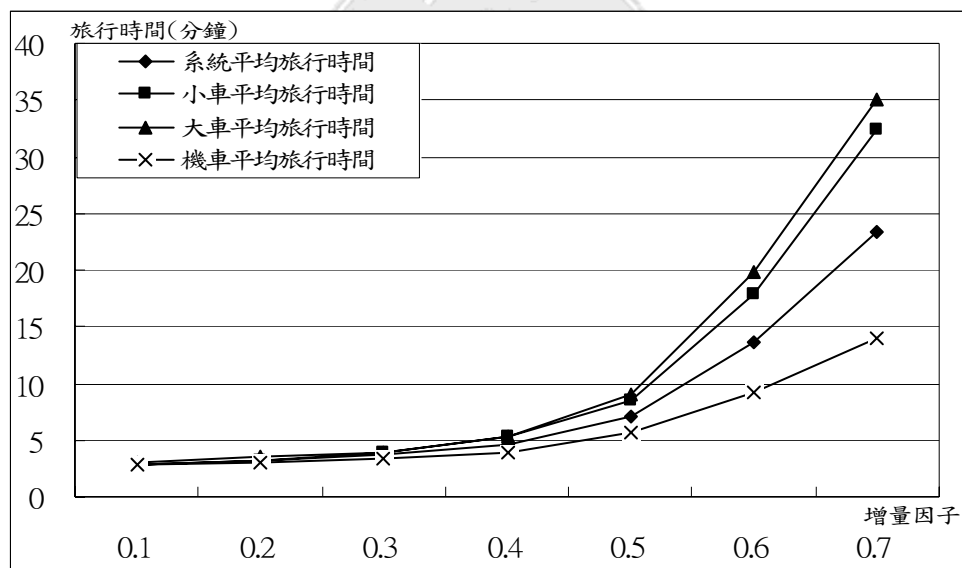


圖 4.5 均一流量需求分佈下各車種平均旅行時間

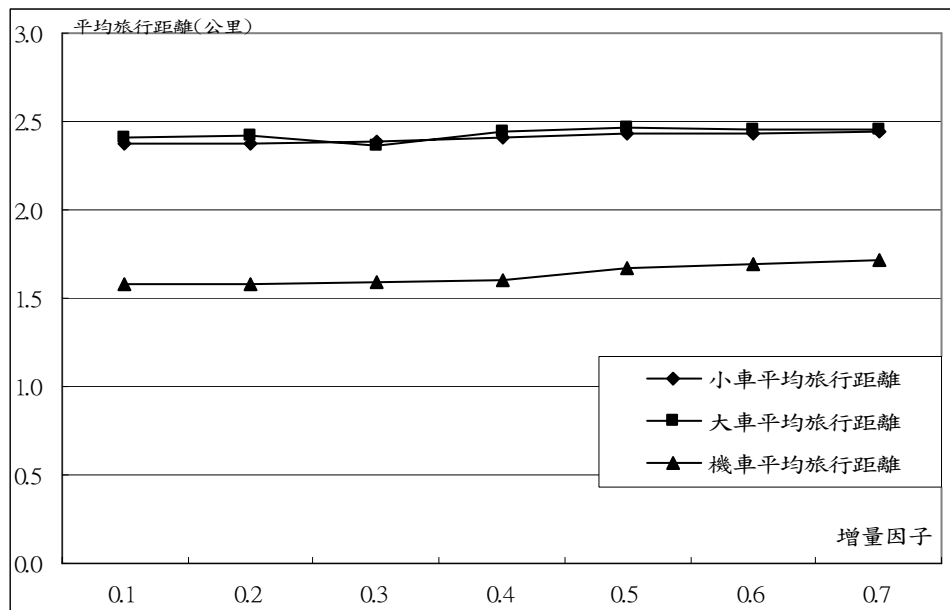


圖 4.6 均一流量需求分佈下各車種平均旅行距離

2. 高尖峰流量需求分佈

在高尖峰流量分佈狀態下，其數值結果如表 4.3 所示，當總車輛數由 7,917 輛增加至 58,194 輛時，其系統平均旅行時間由 3.04 分鐘增加至 30.19 分鐘，而其各車種之旅行時間明顯增加，如小車平均旅行時間由 3.07 分鐘增加至 38.02 分鐘，尤其於增量因子 0.5 時，其高尖峰流量需求所需之旅行時間比均一流量需求分佈旅行時間增加 2 倍左右，由此可以說明路網如於某一時段下，車輛數突然增加，對於路網中之車輛亦會造成車輛旅行時間明顯增加。就各車種之平均旅行距離則也是以小幅度增加。

表 4.3 高尖峰流量需求分佈下系統績效統計表

增量因子	總車輛	平均旅行時間 (分鐘)	小車	小車 平均旅行 時間 (分鐘)	小車 平均旅行 距離 (公里)	大車	大車 平均旅行 時間 (分鐘)	大車 平均旅行 距離 (公里)	機車	機車 平均旅行 時間 (分鐘)	機車 平均旅行 距離 (公里)
0.1	7,917	3.04	3,047	3.07	2.39	844	3.15	2.40	4,026	3.00	1.59
0.2	16,800	3.70	7,083	3.91	2.39	1,317	4.44	2.42	8,400	3.40	1.59
0.3	24,692	4.18	9,781	6.24	2.40	2,494	6.47	2.38	12,417	4.08	1.61
0.4	33,491	7.43	13,565	9.60	2.40	3,179	9.54	2.39	16,747	4.27	1.64
0.5	41,460	12.81	16,800	16.21	2.44	3,858	17.88	2.46	20,802	9.13	1.70
0.6	50,373	18.94	20,554	24.88	2.44	4,633	24.52	2.45	25,186	12.88	1.70
0.7	58,194	30.19	23,105	38.02	2.46	5,924	40.23	2.45	29,165	21.94	1.72

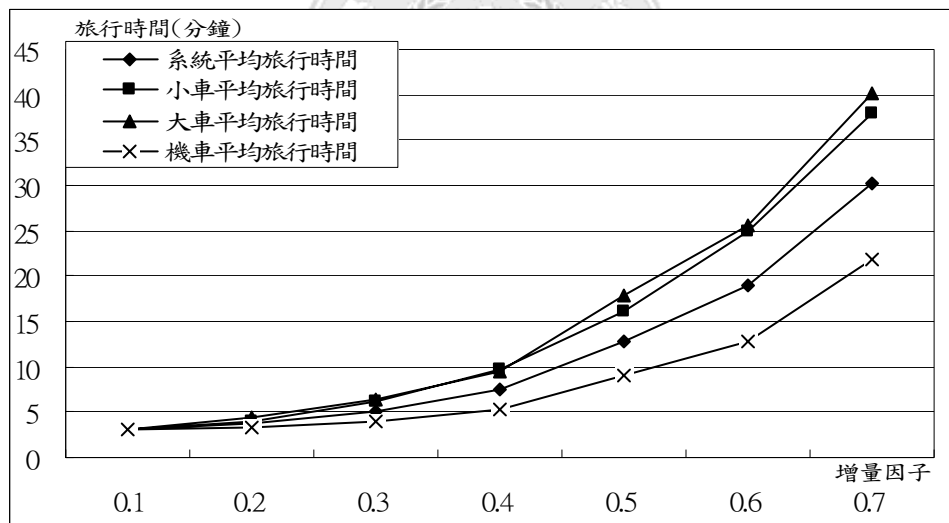


圖 4.7 高尖峰流量需求分佈下各車種平均旅行時間

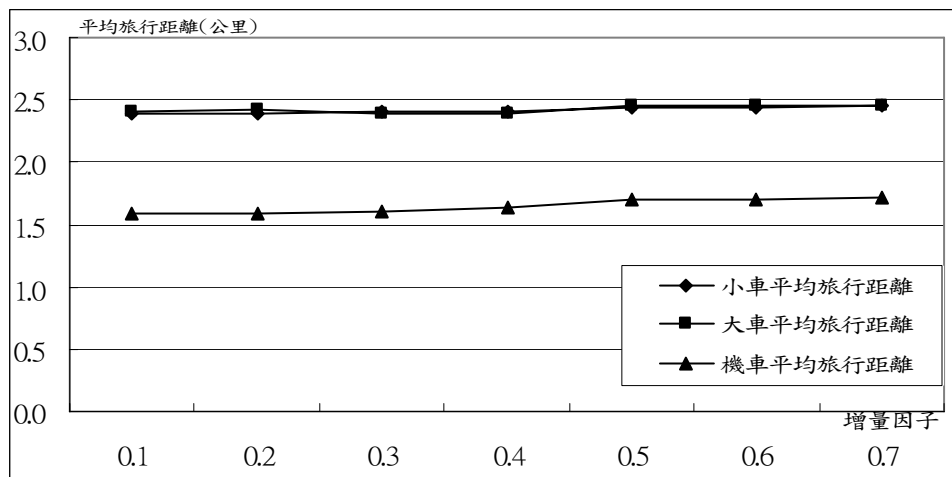


圖 4.8 高尖峰流量需求分佈下各車種平均旅行距離

3. 常態流量需求分佈

常態流量需求分佈數值結果如表 4.4 所示，總車輛數由 8,417 輛增加至 61,318 輛，系統平均旅行時間也由 2.9 分鐘增加到 36.44 分鐘，各車種之平均旅行時間亦有明顯增加，如增量因子 0.7 下，小車所需平均旅行時間為 49.53 分鐘，其時間與高尖峰之 38.02 分鐘與均一之 32.26 分鐘高出許多，大車與機車旅行時間亦有同樣情形。由圖 4.9 可以觀察到其指數型曲線較均一與高尖峰之曲線陡峭，由此可見，於常態分配流量較為符合現實交通狀況，而其車種之平均旅行距離呈現小幅度成長趨勢。

表 4.4 常態流量需求分佈下系統績效統計表

增量因子	總車輛	平均旅行時間 (分鐘)	小車	小車 平均旅行時間 (分鐘)	小車 平均旅行距離 (公里)	大車	大車 平均旅行時間 (分鐘)	大車 平均旅行距離 (公里)	機車	機車 平均旅行時間 (分鐘)	機車 平均旅行距離 (公里)
0.1	8,417	2.9	3,329	2.87	2.39	916	3.01	2.39	4,172	2.9	1.58
0.2	17,442	3.31	7,060	3.43	2.38	1,759	3.59	2.42	8,623	3.16	1.59
0.3	26,122	4.14	10,500	4.57	2.40	2,573	4.73	2.45	13,049	3.68	1.61
0.4	35,144	6.16	14,376	7.47	2.42	3,300	7.78	2.45	17,468	4.78	1.65
0.5	43,897	9.72	17,920	12.51	2.44	3,991	13.83	2.47	21,986	6.71	1.68
0.6	52,724	17.73	21,141	23.80	2.42	5,341	24.87	2.44	26,242	11.39	1.70
0.7	61,318	36.44	24,333	49.53	2.43	6,375	50.57	2.42	30,610	23.1	1.71

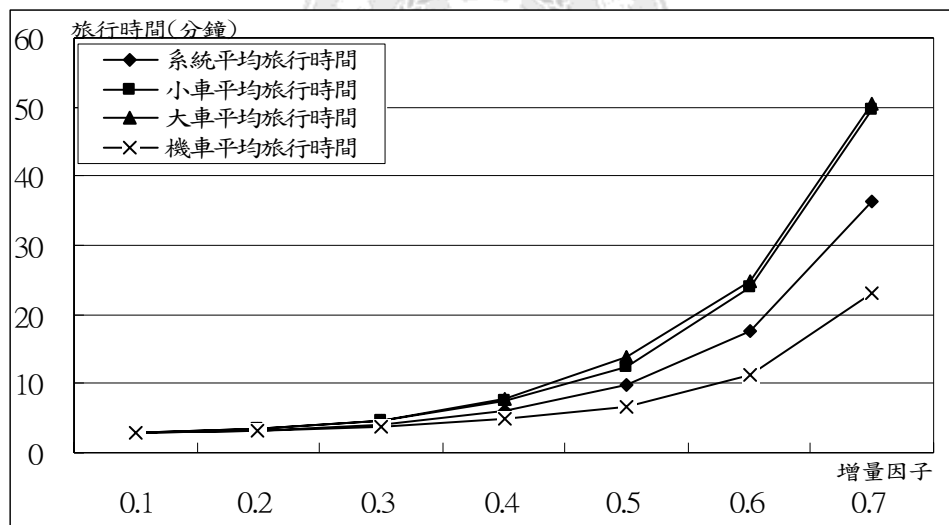


圖 4.9 常態流量需求分佈下各車種平均旅行時間

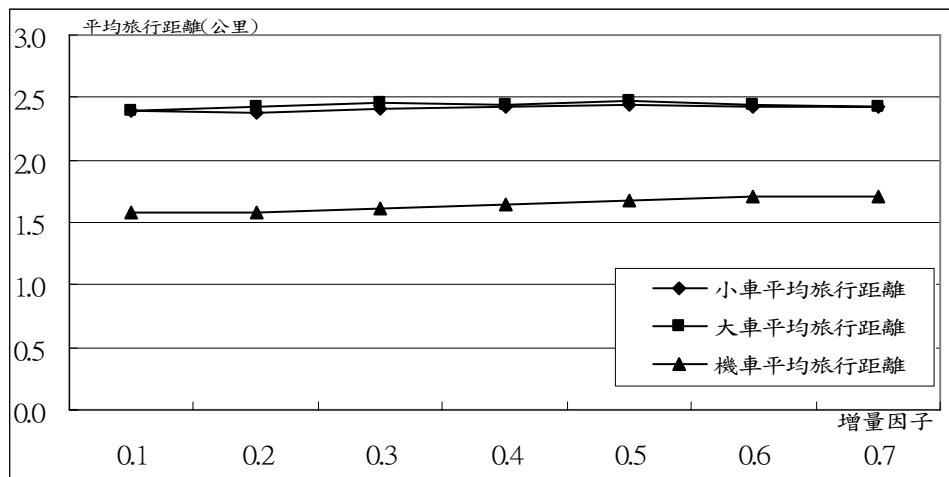


圖 4.10 常態流量需求分佈下各車種平均旅行距離

4. 綜合說明

若以各車種個別來看，由圖 4.11 與圖 4.12 可觀察到大車與小車之旅行時間於三種需求流量分佈中，旅行時間曲線型態極為類似，可能是因為大車、小車所使用路徑相同所導致。此外，在此二圖中尚可觀察到，增量因子在 0.4 至 0.6 時，高尖峰分布之旅行時間高於常態分布之旅行時間，主要是因為於高尖峰在需求時段中，有四個時段車輛數比均一分布之車輛增加 2 倍，因此在流量突然增加狀態下，造成路網中車輛無法即時紓解，再者，本實驗假設無提供即時資訊給路網中車輛使用，因此導致車輛之旅行時間增加。由數值測試觀察，DynaTAIWAN 確實可以模擬出路網中車輛突然增加狀況下，路網中車輛旅行時間之變化。常態流量需求分佈則因其車輛產生為逐步增加產生，因此曲線呈現穩定成長狀態。均一流量需求分佈車輛則依據增量因子放大倍數下呈現穩定的成長。

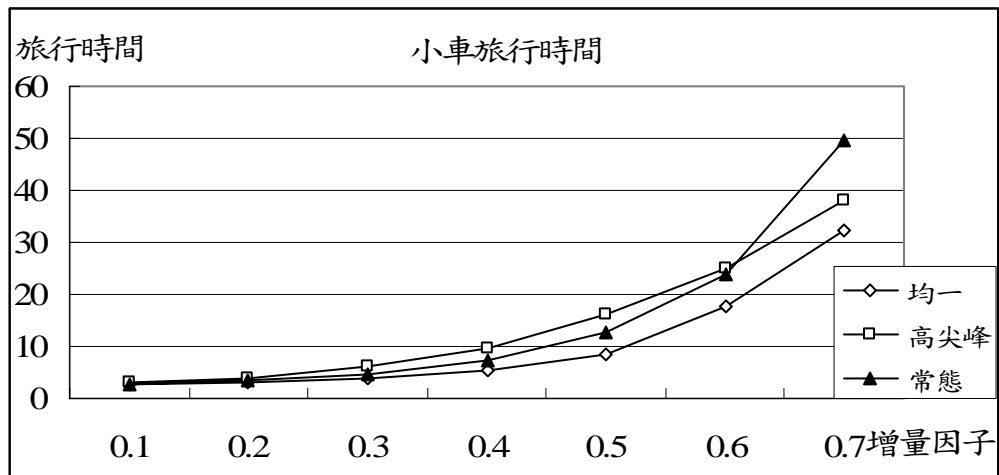


圖 4.11 小車旅行時間

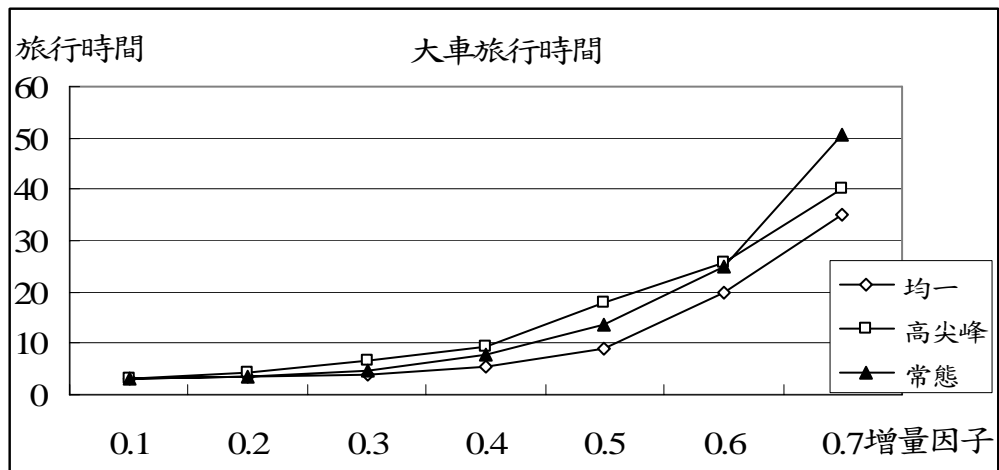


圖 4.12 大車旅行時間

在機車模擬結果方面，其旅行時間可由圖 4.13 中觀察，同樣三種流量需求分佈下，旅行時間之變化並不大，可說明機車於路網中縱使因道路呈現擁擠之狀態下，機車亦可利用車道中剩餘之足夠空間進行前進，對於機車之旅行時間影響似乎不大，其模擬出之結果亦符合機車於道路上之特性。

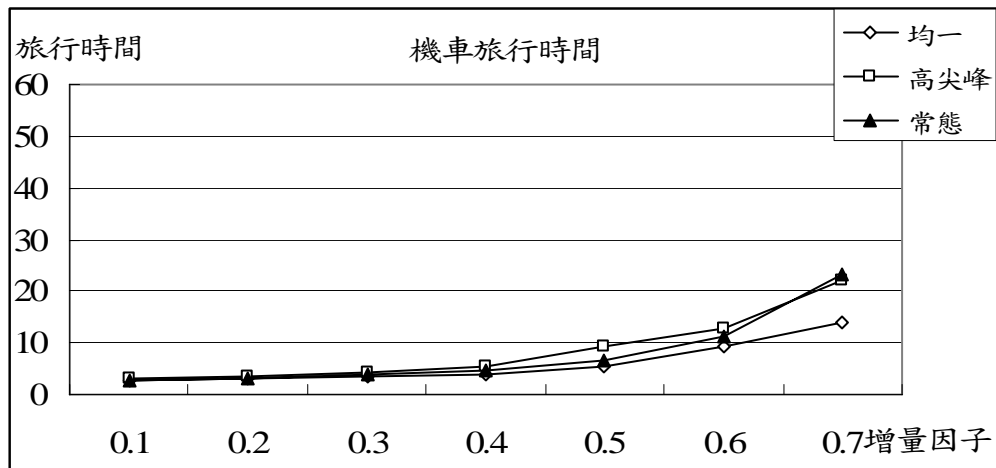


圖 4.13 機車旅行時間

根據以上增量因子於 50 節點之測試，觀察歸納成下列幾點：

1. 在三種流量需求分布下，高尖峰與常態需求分佈之旅行時間變化趨勢較為明顯。
2. 由輸出之結果觀察可發現，當路網車輛增加時，會造成汽車旅行時間增加。
3. 機車所需之旅行時間範圍由 0 至 23 分鐘，其中由圖 4.13 中可觀察到常態與高尖峰流量分佈下，機車所需旅行時間差異不如汽車變化差異大，此可說明，當道路壅塞時對於機車所造成影響不如汽車明顯。此可表現當道路壅塞時，機車因為可利用道路剩餘空間前進以不致於對其旅行時間造成太大之影響。
4. 50 節點增量因子測試中，觀察測試範圍以 0.1 至 0.7 為佳，如增量因子設定為 0.8 時，會造成路網車輛過多導致道路壅塞，各壅塞路段車輛只能以最小速度移動。
5. 比較三種流量需求分佈可以發現，高尖峰與常態流量分佈所反應之車輛旅行時間變化較為明顯。

4.3.2 依時性最短路徑測試

本節以 50 節點測試路網進行依時性最短路徑之測試，其測試目的為：

1. 觀察 DynaTAIWAN 所產生依時性旅行成本，並經由依時性最短路徑演算法計算出小車與機車路徑，並比較兩者路徑是否會使用一樣路徑。
2. 觀察於各指派時段下，各指派時段所產生之依時性最短路徑旅行成本是否比自由車流下旅行成本大。

測試方法係選取常態流量需求分佈之旅運需求資料為測試資料，增量因子設定為 0.5，以取得各模擬時段下車輛旅行成本。

本測試選定相隔最遠之起迄點進行比較，如節點 1 出發至節點 30、節點 1 出發至節點 36、節點 6 出發至節點 25 及節點 6 出發至節點 35，觀察這些起迄點於指派時段下系統所使用之最短路徑，結果如表 4.5 所示。其中分別各選取三個時間點進行路徑比較，分別為 1440 秒、2160 秒與 2700 秒三個時間點，如表 4.5 所示。

由選取之四組起迄點進行觀察，於 1440 秒時小車使用之節點以高速公路為優先路徑進行使用，而機車之路徑因無法使用高速公路只可使用當時最佳路徑。2160 秒與 2700 秒時小車避開壅塞路段(如高速公路)改使用市區道路，因此部份時段下機車與小車所使用路徑會有類似之路徑或相同路徑。

表 4.5 常態分佈下路徑之旅行時間與路徑

起點	迄點	車種	指派時段 (秒)	節點數	路徑
1	30	汽車	1,440	12	1,2,3,43,42,41,40,39,32,33,34,30
			2,160	10	1,2,3,4,10,11,17,18,24,30
			2,700	14	1,2,3,9,15,21,27,39,38,37,36,34,29,30
		機車	1,440	10	1,2,3,9,15,21,22,28,29,30
			2,160	9	1,2,44,5,6,12,18,24,30
			2,700	12	1,2,3,9,15,21,27,28,33,36,34,30
	36	汽車	1,440	11	1,2,3,43,42,41,40,39,32,33,34,30
			2,160	10	1,2,3,4,10,11,17,18,24,30
			2,700	14	1,2,3,9,15,21,27,39,38,37,36,34,29,30
		機車	1,440	10	1,2,3,9,15,21,22,28,29,30
			2,160	9	1,2,44,5,6,12,18,24,30
			2,700	12	1,2,3,9,15,21,27,28,33,36,34,30
6	25	汽車	1,440	10	6,5,44,43,42,41,21,27,26,25
			2,160	10	6,5,4,3,2,1,7,13,19,25
			2,700	10	6,5,4,3,2,1,7,13,19,25
		機車	1,440	10	6,5,4,3,2,1,7,13,19,25
			2,160	10	6,5,4,3,2,1,7,13,19,25
			2,700	9	6,5,44,2,1,7,13,19,25
	35	汽車	1,440	10	6,5,44,43,42,41,40,39,32,35
			2,160	12	6,5,4,3,2,1,7,13,19,25,31,35
			2,700	8	6,12,18,24,30,34,36,35
		機車	1,440	10	6,5,4,3,9,15,21,27,32,35
			2,160	8	6,12,18,24,30,34,36,35
			2,700	8	6,12,18,24,30,34,36,35

表 4.6 說明於自由車流下與各指派時間下所使用路徑之旅行成本差異。自由車流下之車輛旅行成本係採用 DynaTAIWAN 於模擬時段 $t=0$ 秒時所產生之車輛旅行成本，此時之旅行成本為所有時段中之最短路徑旅行成本最小。以此旅行成本為基礎，比較依時性最短路徑於 $t=1440$ 秒(24 分鐘)、 $t=2160$ 秒(36 分鐘)與 $t=2700$ 秒(45 分鐘)三個時間點下旅行成本之差異。依據林蔚明(2004)說明依時性最短路徑之旅行時間，其旅行時間應大於或等於自由車流下最短路徑旅行時間。由表 4.7 中可觀察其路徑旅行時間成本比自由車流下旅行時間成本符合測試假設。

表 4.6 自由車流與依時性最短路徑旅行時間比較

常態 分佈	起點			起點			
	1			6			
時間點 迄點	1440	2160	2700	時間點 迄點	1440	2160	2700
30	-299.96	-528.2	-304.51	25	-443.49	-660.47	-501.59
36	-368.54	-580.03	-301.56	35	-337.74	-617.36	-617.36
註：時間差=自由車流旅行時間－依時性最短路徑時間							

第五章 動態使用者均衡指派測試

本章針對動態使用者均衡指派進行相關測試，以第四章 DynaTAIWAN 測試實驗結果做為測試依據，本章架構如下 5.1 節說明動態使用者均衡指派實驗設計，包含實驗基本假設、實驗設計與實驗操作流程。5.2 節說明單一車種(小車) 實驗結果。5.3 節針對混合車流(小車與機車)說明實驗結果。

5.1 動態使用者均衡指派實驗設計

本研究之實驗測試之軟硬體規格與第四章之數值測試環境所使用之軟硬體規格相同。並使用 50 節點為測試路網。實驗主要分別測試單一車種與混合車流進行測試。以下分別針對實驗基本假設與設計、實驗操作流程進行說明。

1. 實驗基本假設與設計

- (1) 車輛初始路徑係依據車輛產生時之各路段旅行成本計算。
- (2) 旅行時間計算方式為路段平均速度算除以路段長度。
- (3) 測試車輛有小汽車與機車兩類。
- (4) 模擬時間長度為 300 分鐘。

實驗測試則分類為單一車種測試與混合車流測試以下分別針對兩組實驗設計進行說明。

(1) 單一車種

實驗以小汽車為主，流量需求亦以均一、高尖峰以及常態三種流量需求分佈資料進行測試，增量因子則分別設定為 0.3、0.5 與 0.7，而增量因子 0.3 產生約 13,000 車輛數、增量因子 0.5 產生約 21,000 輛、增量因子 0.7 產生約 30,000 輛。

(2) 混合車流

測試車種包含小汽車與機車，流量需求則以常態流量分佈為主，增量因子則設定為 0.5 與 0.7，增量因子 0.5 時小車產生 21,892 輛，機車產生 21,987 輛，合計 43,879 車輛，增量因子 0.7 時小車產生 30,771 輛，機車產生 30,629 輛，合計產生 61,400 輛。進行觀察擁擠狀況下動態使用者均衡之變化。實驗中收斂準則係依照第三章所建立之收斂條件來判斷，當路網中違反值低於收斂條件值 50 時，則不需再進行下一遞迴之模擬。

2. 實驗操作流程

實驗操作流程如圖 5.1 所示，首先以設定之增量因子執行 DynaTAIWAN 程式模擬路網中各路段之車輛速度，及各模擬時段下路徑車輛數，作為初步模擬資料。再將模擬所得車輛速度讀至依時性最短路徑演算法，計算出各模擬時段下之依時性最短路徑，此資料用以和原路徑相比較。因 DynaTAIWAN 模組所輸出資料為各車輛所使用路徑，因此需再將各指派時段下某一 OD 對路徑車輛數進行統計。MSA 模組則再依據依時性最短路徑模組與路徑流量資料進行 MSA 演算法計算，重新分配新舊路徑之流量，其計算過程包含收斂準則測試，判斷路徑流量變化量是否已符合收斂準則，如符合則停止，若不符合則遞迴加 1，進入車輛產生模組，將重新分配之路徑與流量提供 DynaTAIWAN 進行下一個遞迴模擬。

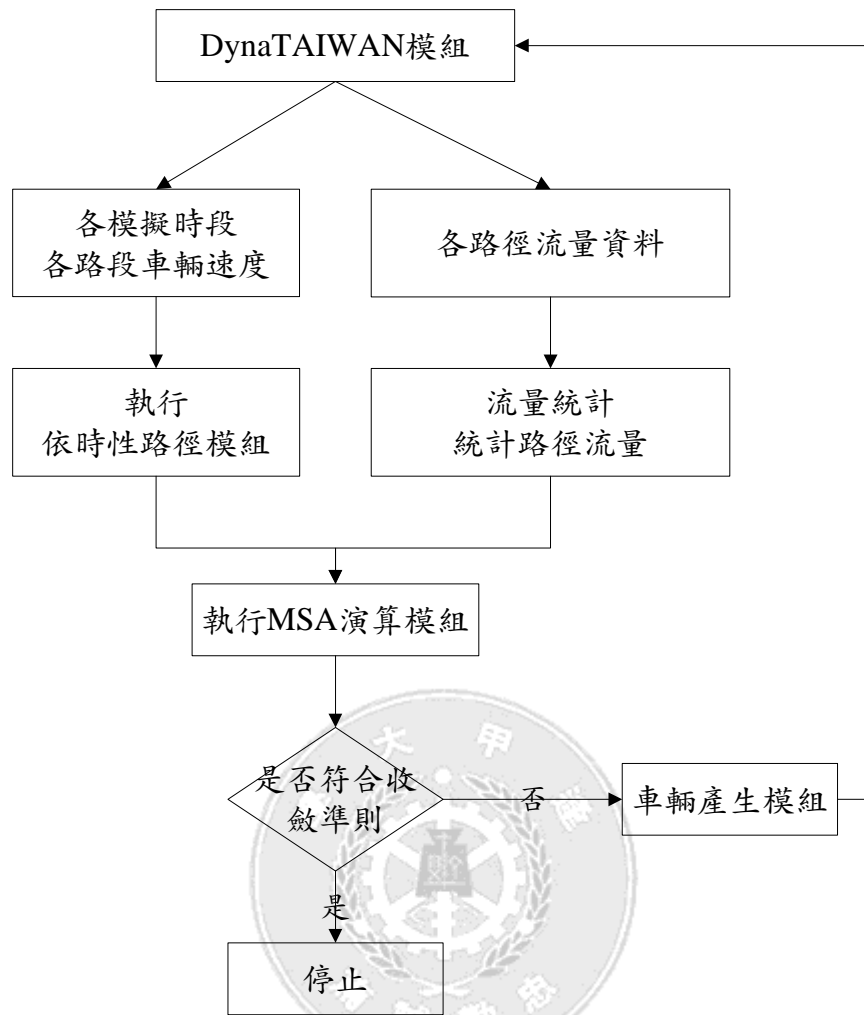


圖 5.1 實驗操作步驟

5.2 單一車種實驗

本節分別針對單一車種進行動態使用者均衡實驗測試，單一車種分別針對均一流量、尖峰流量與常態流量進行測試。

1. 均一流量需求分佈

在均一流量需求分佈下，分別設定增量因子為 0.3、0.5 及 0.7 之情況進行實驗。表 5.1 至表 5.3 為實驗結果。增量因子 0.3 時需 10 次遞迴才可符合收斂條件，增量因子 0.5 時則需要 15 次遞迴，增量因子 0.7 時則需 20 次遞迴符合收斂條件。

表 5.1 均一流量需求分佈(0.3)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	13,251	3.57	3.57	2380.03	1.14	
1	13,251	3.55	3.55	2381.69	1.09	3,832
2	13,251	3.52	3.52	2382.07	1.07	1,998
3	13,251	3.53	3.53	2382.33	1.07	1,021
4	13,251	3.63	3.63	2377.41	1.17	1,383
5	13,251	3.62	3.62	2377.31	1.16	433
6	13,251	3.61	3.61	2377.56	1.14	251
7	13,251	3.62	3.62	2377.56	1.16	160
8	13,251	3.61	3.61	2377.56	1.15	115
9	13,251	3.63	3.63	2377.56	1.16	81
10	13,251	3.61	3.61	2377.56	1.15	37

表 5.2 均一流量需求分佈(0.5)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	22,119	5.19	5.19	2400.59	2.58	
1	22,119	5.23	5.23	2403.96	2.57	5,552
2	22,119	5.27	5.27	2401.07	2.63	4,044
3	22,119	5.29	5.29	2401.68	2.66	2,267
4	22,119	5.27	5.27	2402.00	2.63	1,336
5	22,119	5.24	5.24	2402.03	2.60	802
6	22,119	5.24	5.24	2402.09	2.60	459
7	22,119	5.26	5.26	2402.11	2.62	275
8	22,119	5.24	5.24	2402.14	2.60	187
9	22,119	5.26	5.26	2402.12	2.62	142
10	22,119	5.26	5.26	2402.09	2.62	125
11	22,119	5.26	5.26	2402.09	2.62	112
12	22,119	5.26	5.26	2402.08	2.62	96
13	22,119	5.25	5.25	2402.08	2.61	86
14	22,119	5.28	5.28	2402.09	2.64	74
15	22,119	5.28	5.28	2402.09	2.64	47

表 5.3 均一流量需求分佈(0.7)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	30,969	8.49	8.49	2420.18	5.53	
1	30,969	10.52	10.52	2429.53	7.20	7,096
2	30,969	9.28	9.28	2441.57	6.24	5,850
3	30,969	8.97	8.97	2445.52	5.91	3,621
4	30,969	8.74	8.74	2448.71	5.70	2,266
5	30,969	8.69	8.69	2449.61	5.69	1,365
6	30,969	8.64	8.64	2450.38	5.66	857
7	30,969	8.64	8.64	2450.99	5.64	512
8	30,969	8.65	8.65	2450.89	5.67	346
9	30,969	8.65	8.65	2450.85	5.67	224
10	30,969	8.70	8.70	2450.86	5.71	172
11	30,969	8.68	8.68	2450.85	5.67	142
12	30,969	8.69	8.69	2450.85	5.71	122
13	30,969	8.70	8.70	2450.93	5.71	105
14	30,969	8.69	8.69	2450.95	5.69	92
15	30,969	8.69	8.69	2451.14	5.70	93
16	30,969	8.69	8.69	2451.10	5.68	80
17	30,969	8.67	8.67	2451.10	5.68	73
18	30,969	8.72	8.72	2451.10	5.72	67
19	30,969	8.68	8.68	2451.10	5.66	56
20	30,969	8.81	8.81	2451.10	5.75	49

增量因子 0.3 之實驗結果共產生 13,251 車輛數。此次實驗中車輛旅行時間由 3.57 分鐘改善至 3.52 分鐘後，旅行時間則在 3.61 至 3.63 之間震盪。實驗中因路網未呈現擁擠狀況，指派後車輛旅行時間改善不明顯。違反值趨勢則於第 4 遞子迴前呈現震盪後，慢慢以平滑曲線往收斂準則值 50 接近。

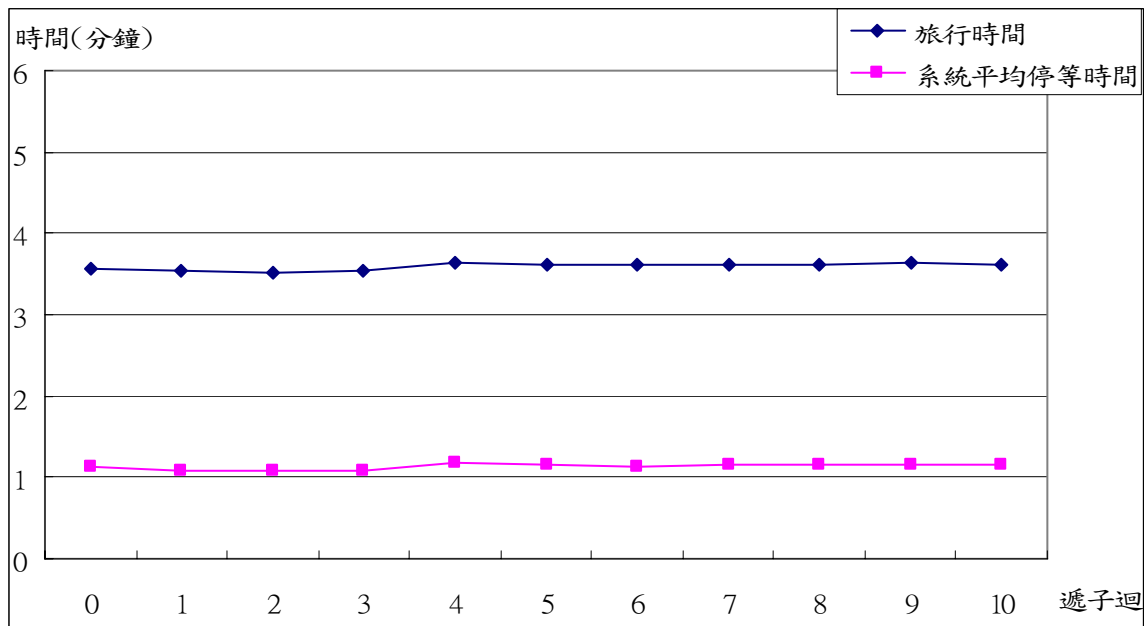


圖 5.2 均一流量需求(0.3)時間趨勢圖

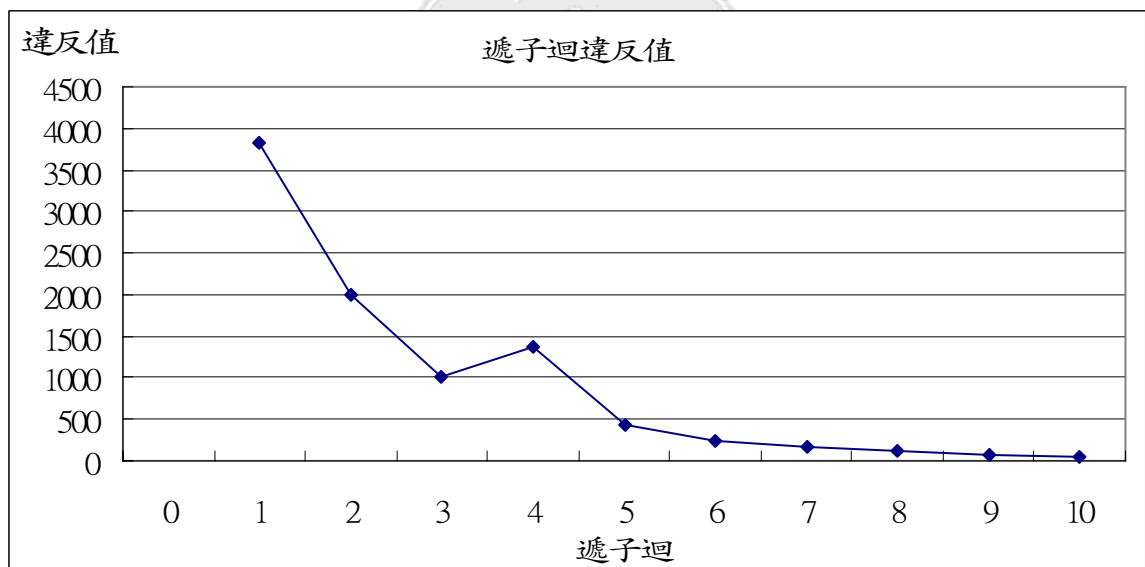


圖 5.3 均一流量需求分佈下(0.3)違反值

增量因子 0.5 之實驗結果共產生 22,119 車輛數總數。實驗中車輛旅行時間由 5.19 分鐘開始增加至 5.29 分鐘時，車輛旅行時間達到最高點後，旅行時間曲線往下移動至 5.26 分鐘附近震盪。小車旅行時間初期會呈現增加情況，主要初始模擬提供較佳路徑，以致於車輛旅行時間會靠近初始旅行時間附近震盪。車輛旅行距離則由 2400 公尺慢

慢接近至 2402 公尺後呈現平穩狀態。

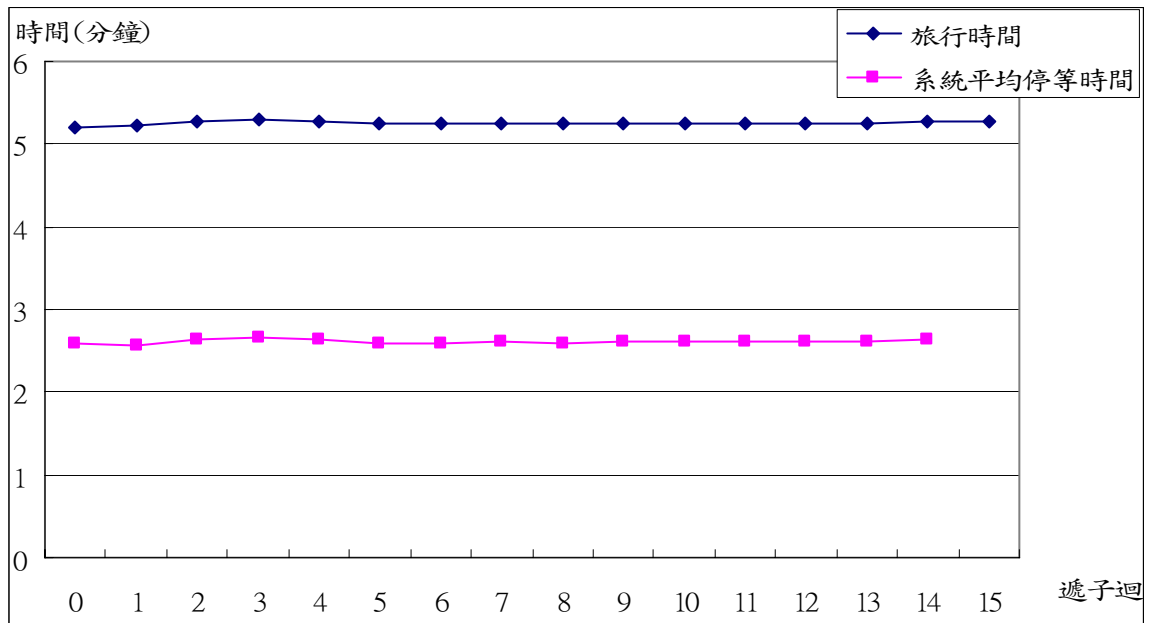


圖 5.4 均一流量需求(0.5)時間趨勢圖

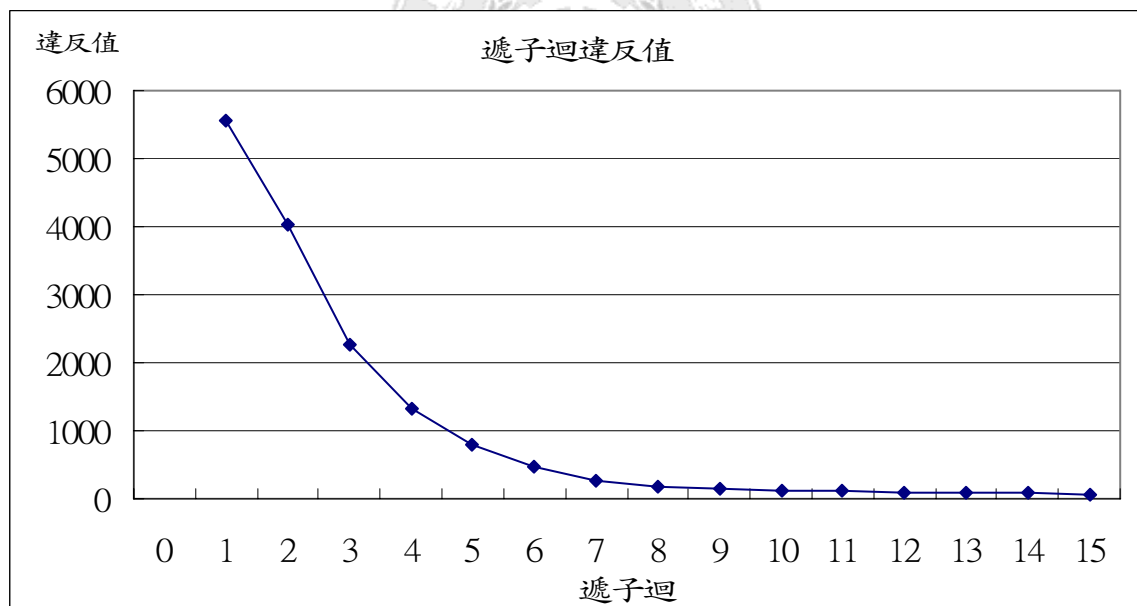


圖 5.5 均一流量需求分佈下(0.5)違反值

增量因子為 0.7 之實驗結果，產生了 30,969 輛車。此實驗中車輛之旅行時間在從 8.49 分鐘增加至 10.52 分鐘後，便開始往 8.6 分鐘移動後，慢慢呈現穩定震盪。車輛之旅行距離則由 2420 公尺慢慢增加

至 2451 公尺後，慢慢呈現平穩的曲線，路網中車輛所使用之路徑距離亦慢慢趨於穩定。違反值則是以平滑曲線往收斂準則值接近。

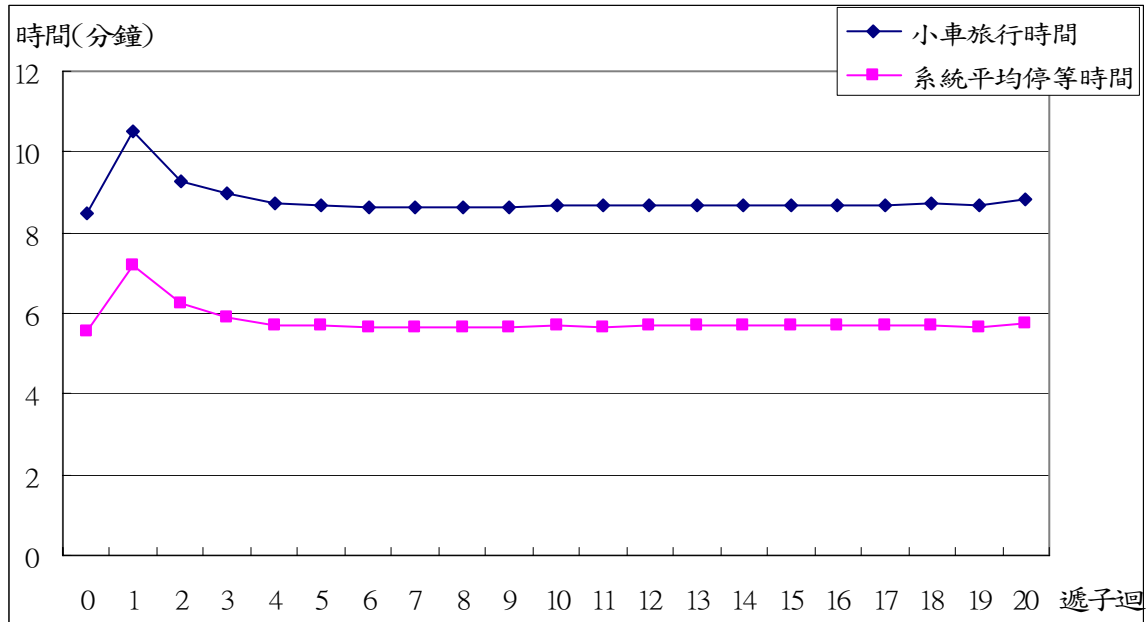


圖 5.6 均一流量需求(0.7)時間趨勢圖

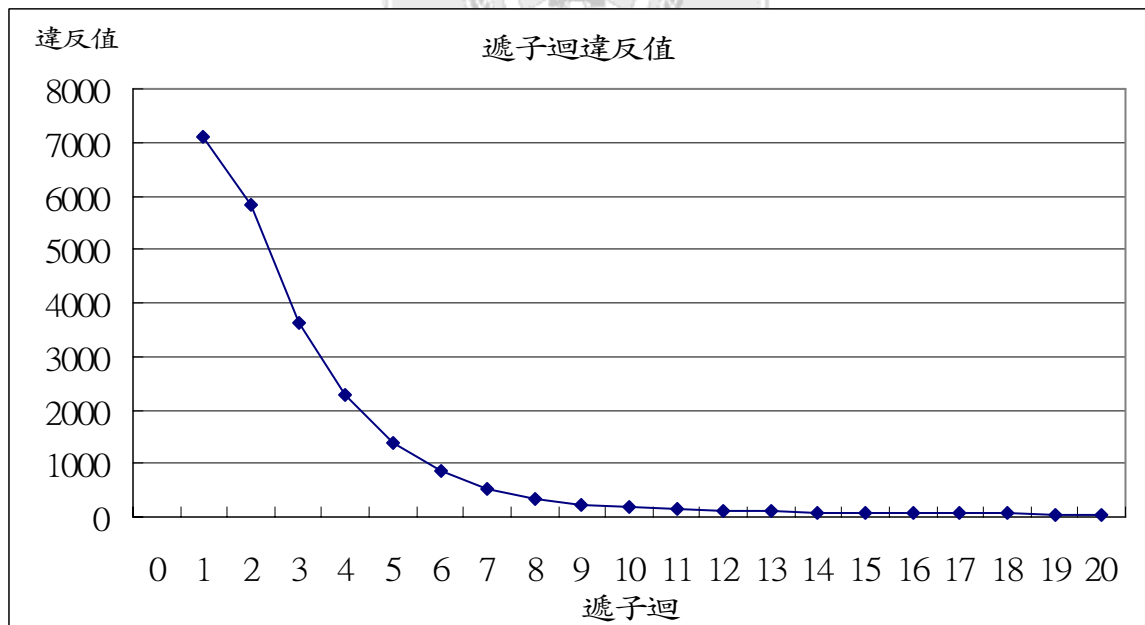


圖 5.7 均一流量需求分佈下(0.7)違反值

2. 尖峰流量需求分佈

此一實驗主要為測試路網中車流量為尖峰流量之反應，假設路網突然三個時段下增加兩倍之車流量，模擬觀察其對動態使用者均衡之影響，其產生總車輛數為 42,750 輛。表 5.4 至表 5.6 為尖峰流量於各增量因子下之實驗結果。當增量因子為 0.3 時需 10 次遞子迴才可達收斂條件，增量因子 0.5 與 0.7 時各需要 12 與 19 次遞子迴。

表 5.4 尖峰流量需求分佈(0.3)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	12,238	4.49	4.49	2387.50	1.94	
1	12,238	4.50	4.50	2387.77	1.94	3,138
2	12,238	4.43	4.43	2388.01	1.85	1,821
3	12,238	4.42	4.42	2387.81	1.84	1,015
4	12,238	4.43	4.43	2388.26	1.86	604
5	12,238	4.46	4.46	2388.44	1.88	382
6	12,238	4.45	4.45	2388.44	1.87	227
7	12,238	4.45	4.45	2388.34	1.86	153
8	12,238	4.43	4.43	2388.28	1.85	98
9	12,238	4.42	4.42	2388.28	1.84	65
10	12,238	4.44	4.44	2388.26	1.86	43

表 5.5 高尖峰流量需求分佈(0.5)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	20,680	7.63	7.63	2391.51	4.87	
1	20,680	7.78	7.78	2400.83	4.99	4,863
2	20,680	7.74	7.74	2406.21	4.92	3,353
3	20,680	7.66	7.66	2410.02	4.79	1,968
4	20,680	7.60	7.60	2411.10	4.76	1,223
5	20,680	7.64	7.64	2411.99	4.80	814
6	20,680	7.61	7.61	2412.61	4.77	574
7	20,680	7.66	7.66	2412.97	4.81	416
8	20,680	7.70	7.70	2413.08	4.80	251
9	20,680	7.62	7.62	2413.30	4.77	147
10	20,680	7.69	7.69	2413.49	4.80	82
11	20,680	7.63	7.63	2413.49	4.79	70
12	20,680	7.65	7.65	2413.52	4.80	48

表 5.6 高尖峰流量需求分佈(0.7)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	29,061	14.00	14.00	2430.34	10.89	
1	29,061	15.73	15.73	2454.46	12.03	6,369
2	29,061	13.79	13.79	2472.43	10.40	4,760
3	29,061	14.60	14.60	2473.37	10.67	6,226
4	29,061	13.48	13.48	2477.34	10.02	3,027
5	29,061	13.15	13.15	2479.03	9.91	1,781
6	29,061	13.46	13.46	2479.55	10.01	1,242
7	29,061	13.12	13.12	2480.22	9.88	891
8	29,061	13.02	13.02	2479.90	9.75	645
9	29,061	13.03	13.03	2479.82	9.81	481
10	29,061	12.96	12.96	2480.10	9.72	360
11	29,061	12.91	12.91	2480.25	9.70	272
12	29,061	13.21	13.21	2480.48	9.80	209
13	29,061	12.95	12.95	2480.61	9.74	159
14	29,061	12.94	12.94	2480.68	9.73	115
15	29,061	12.92	12.92	2480.74	9.72	94
16	29,061	12.89	12.89	2480.76	9.65	71
17	29,061	12.89	12.89	2480.77	9.67	55
18	29,061	12.88	12.88	2480.81	9.65	52
19	29,061	12.95	12.95	2480.82	9.71	44

增量因子 0.3 實驗結果產生車輛數總數為 12,238 輛車。此實驗中車輛旅行時間由 4.49 分鐘增加至 4.5 分鐘後逐漸往下移動，最低旅行時間出現於第 3 個遞子迴 4.42 分鐘，後續旅行時間則於 4.42 至 4.44 分鐘之間震盪。車輛之旅行距離則由 2387 公尺增加至 2388 公尺。

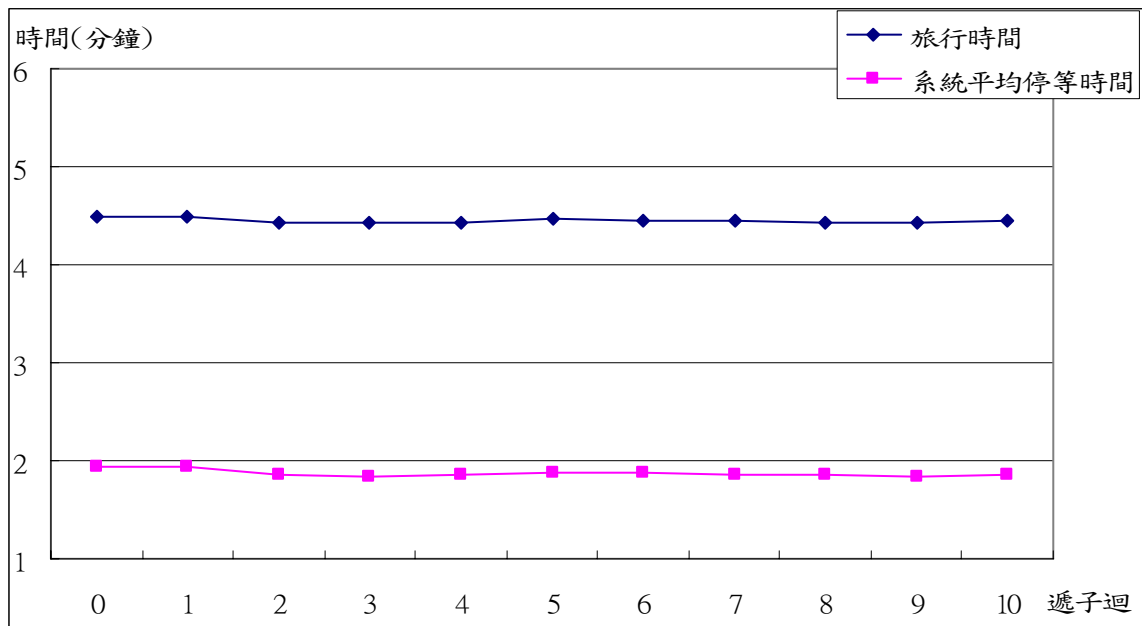


圖 5.8 尖峰流量需求(0.3)時間趨勢圖

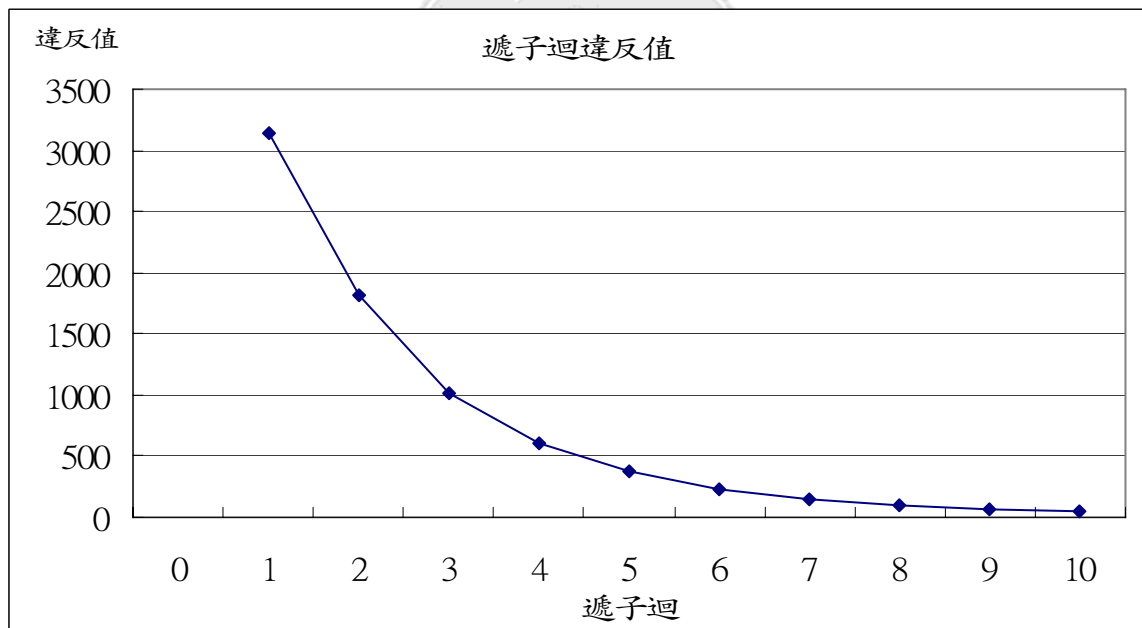


圖 5.9 尖峰流量需求分佈下(0.3)違反值

增量因子為 0.5 之實驗結果產生了 20,680 輛車。此實驗中車輛旅行時間由 7.63 分鐘增加至 7.78 分鐘後，開始往 7.6 分鐘附近移動，而其中最低旅行時間為 7.58 分鐘，在此一實驗中旅行時間會於 7.6 與

7.65 分鐘之間慢慢震盪後趨於平穩狀態。旅行距離則由 2391 公尺慢慢增加至 2413 公尺後車輛旅行距離趨於平穩狀態曲線。違反值則由原 4863 慢慢減緩接近收斂準則值 50。

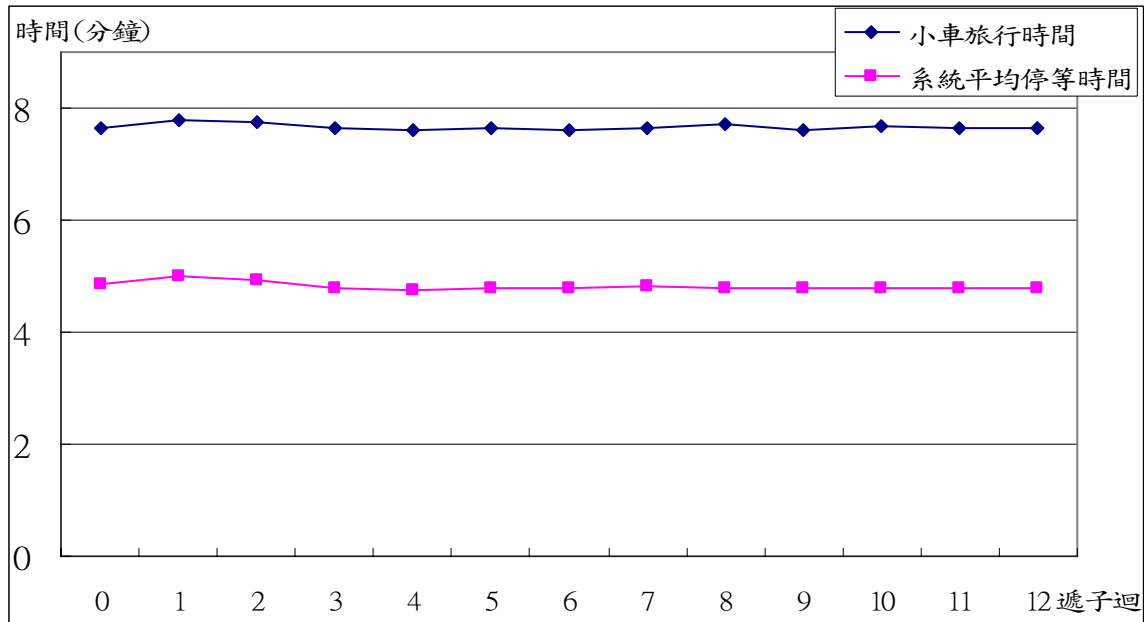


圖 5.10 尖峰流量需求(0.5)時間趨勢圖

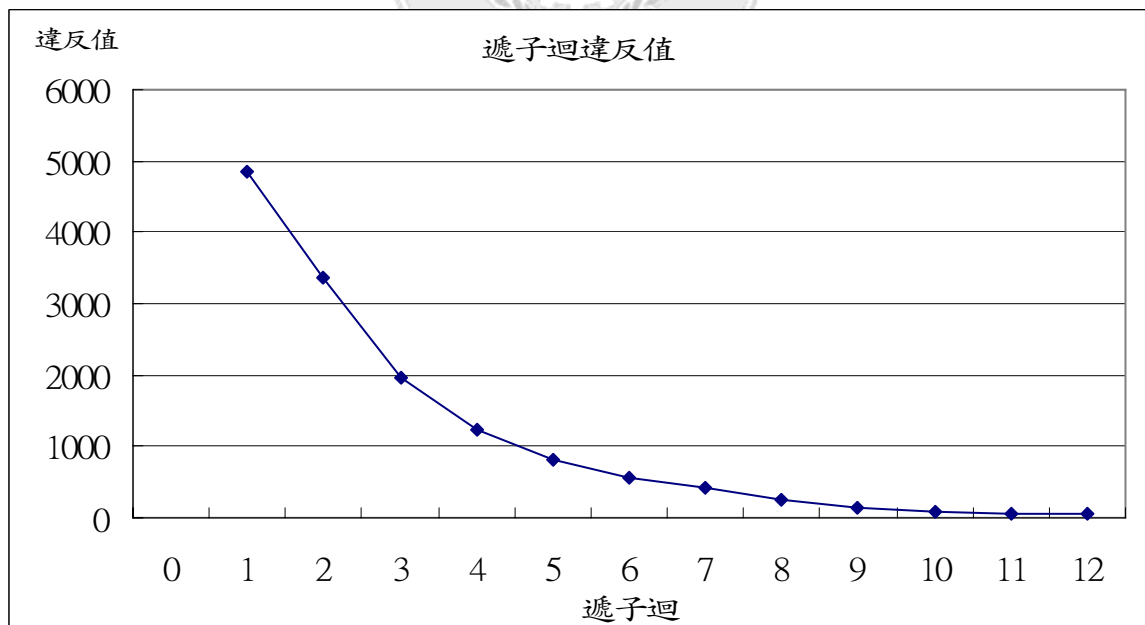


圖 5.11 高尖峰流量需求分佈下(0.5)違反值

增量因子為 0.7 之實驗結果產生車輛數總數為 29,061 輛車。此實驗中車輛旅行時間由 14 分鐘增加至 15.73 分鐘後，便開始往 12 分鐘附近移動，而其中最低旅行時間為 12.91 分鐘，在此一實驗中旅行時間會於 12 與 13 分鐘之間慢慢震盪後趨於平穩狀態。旅行距離則由 2430 公尺慢慢增加至 2480 公尺後車輛旅行距離趨於平緩狀態曲線。違反值由原 6369 慢慢減緩接近收斂準則值 50。

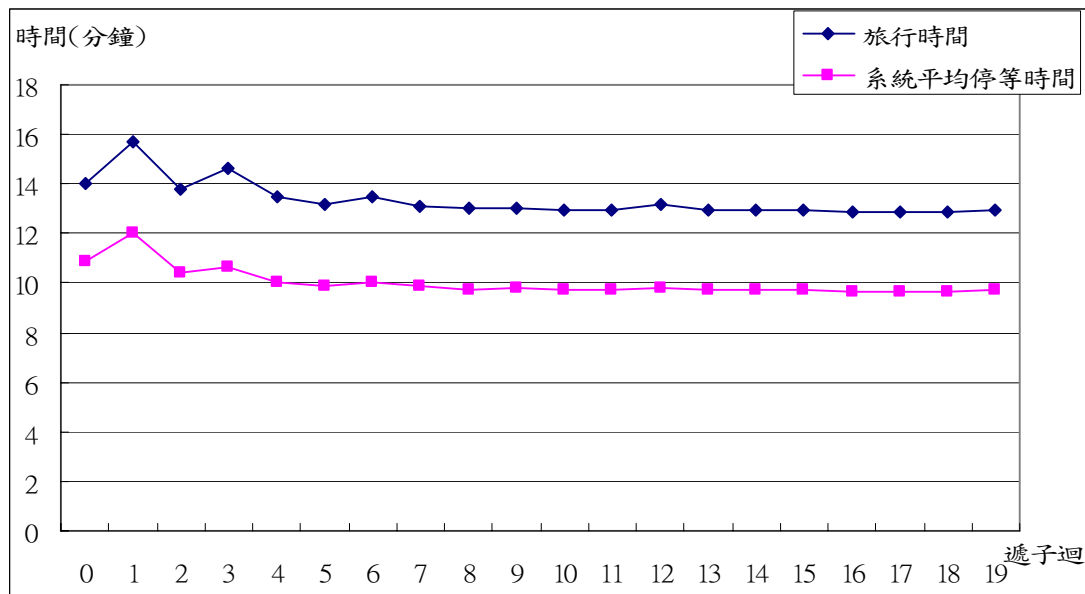


圖 5.12 尖峰流量需求(0.7)時間趨勢圖

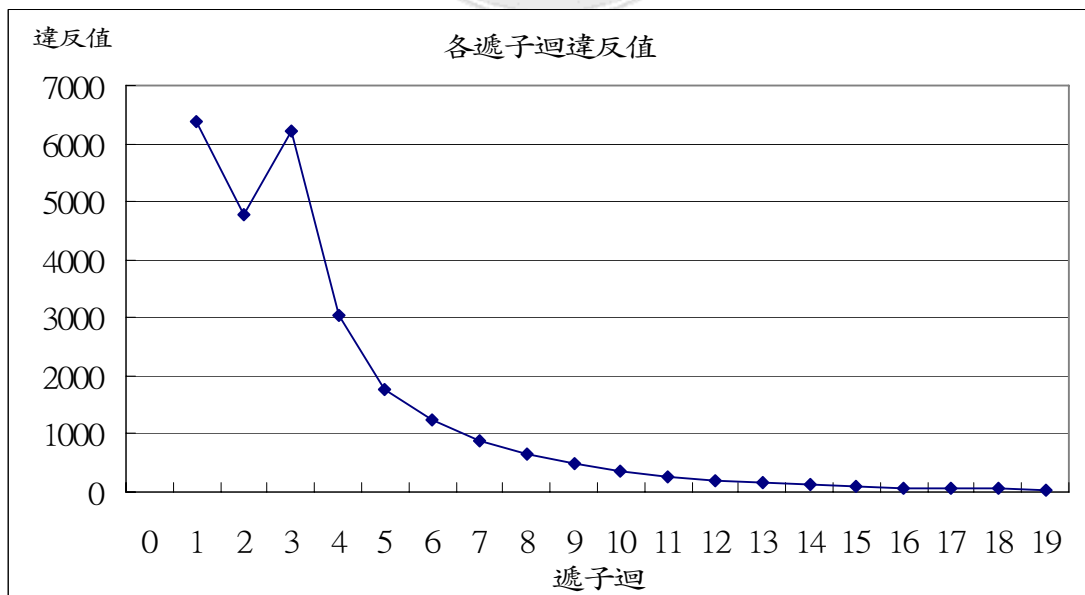


圖 5.13 高尖峰流量需求分佈下(0.7)違反值

3. 常態流量需求分佈

此實驗以常態流量需求分佈之車流量變化觀察動態使用者均衡之反應。表 5.7 至表 5.9 為常態流量於各增量因子下之實驗結果。各增量因子 0.3、0.5 與 0.7 違反值分別於第 10 次、第 15 次及第 20 次達到收斂條件。

表 5.7 常態流量需求分佈(0.3)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	13,069	3.97	3.97	2385.57	1.49	
1	13,069	3.91	3.91	2388.06	1.40	3,469
2	13,069	3.94	3.94	2386.87	1.43	2,072
3	13,069	3.94	3.94	2387.37	1.43	1,160
4	13,069	3.91	3.91	2387.75	1.41	670
5	13,069	3.92	3.92	2387.73	1.42	391
6	13,069	3.92	3.92	2387.65	1.42	238
7	13,069	3.92	3.92	2387.62	1.42	142
8	13,069	3.93	3.93	2387.56	1.42	105
9	13,069	3.93	3.93	2387.56	1.42	68
10	13,069	3.92	3.92	2387.52	1.41	46

表 5.8 常態流量需求分佈(0.5)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間 (分鐘)	系統平均 旅行時間 (分鐘)	系統平均 旅行距離 (公尺)	系統平均 停等時間 (分鐘)	違反值
0	21,908	6.25	6.26	2399.14	3.55	
1	21,908	6.51	6.51	2406.61	3.75	5,310
2	21,908	6.42	6.42	2410.20	3.65	3,765
3	21,908	6.34	6.34	2412.28	3.58	2,261
4	21,908	6.33	6.33	2412.58	3.55	1,323
5	21,908	6.31	6.31	2412.59	3.54	848
6	21,908	6.32	6.32	2412.75	3.54	546
7	21,908	6.34	6.34	2412.87	3.57	348
8	21,908	6.33	6.33	2412.77	3.54	226
9	21,908	6.33	6.33	2412.76	3.57	157
10	21,908	6.31	6.31	2412.74	3.53	112
11	21,908	6.36	6.36	2412.78	3.58	85
12	21,908	6.32	6.32	2412.78	3.54	72
13	21,908	6.33	6.33	2412.76	3.54	66
14	21,908	6.35	6.35	2412.73	3.56	59
15	21,908	6.27	6.27	2412.71	3.51	48

表 5.9 常態流量需求分佈(0.7)實驗結果

遞子迴	車輛數 (輛)	小車旅行時間	系統平均旅行時間 (分鐘)	系統平均旅行距離 (公尺)	系統平均停等時間 (分鐘)	違反值
0	30,727	11.61	11.61	2422.97	8.60	
1	30,727	12.62	12.62	2440.37	9.31	6,779
2	30,727	11.75	11.75	2464.23	8.50	5,625
3	30,727	11.30	11.30	2473.98	8.14	3,544
4	30,727	11.30	11.30	2474.59	8.12	2,166
5	30,727	11.31	11.31	2475.34	8.13	1,421
6	30,727	11.22	11.22	2475.97	8.11	956
7	30,727	11.33	11.33	2476.49	8.15	614
8	30,727	11.29	11.29	2476.53	8.16	428
9	30,727	11.27	11.27	2477.01	8.15	352
10	30,727	11.46	11.46	2477.05	8.28	221
11	30,727	11.36	11.36	2477.06	8.19	175
12	30,727	11.37	11.37	2477.04	8.19	135
13	30,727	11.28	11.28	2477.15	8.20	114
14	30,727	11.35	11.35	2477.16	8.18	100
15	30,727	11.31	11.31	2477.15	8.15	83
16	30,727	11.42	11.42	2477.14	8.26	67
17	30,727	11.35	11.35	2477.18	8.20	68
18	30,727	11.28	11.28	2477.16	8.16	59
19	30,727	11.35	11.35	2477.28	8.22	50
20	30,727	11.32	11.32	2477.26	8.16	48

增量因子為 0.3 之實驗結果產生車輛數總數為 13,069 輛車。此實驗中車輛旅行時間由 3.97 分鐘漸減至 3.91 分鐘，並呈現往下移動趨勢，最低旅行時間為 3.91 分鐘並於 3.91 分鐘左右小幅度震盪後趨於平穩。車輛之旅行距離則由 2385 公尺增加至 2387 公尺。違反值由 3469 漸減至接近至收斂準則值。

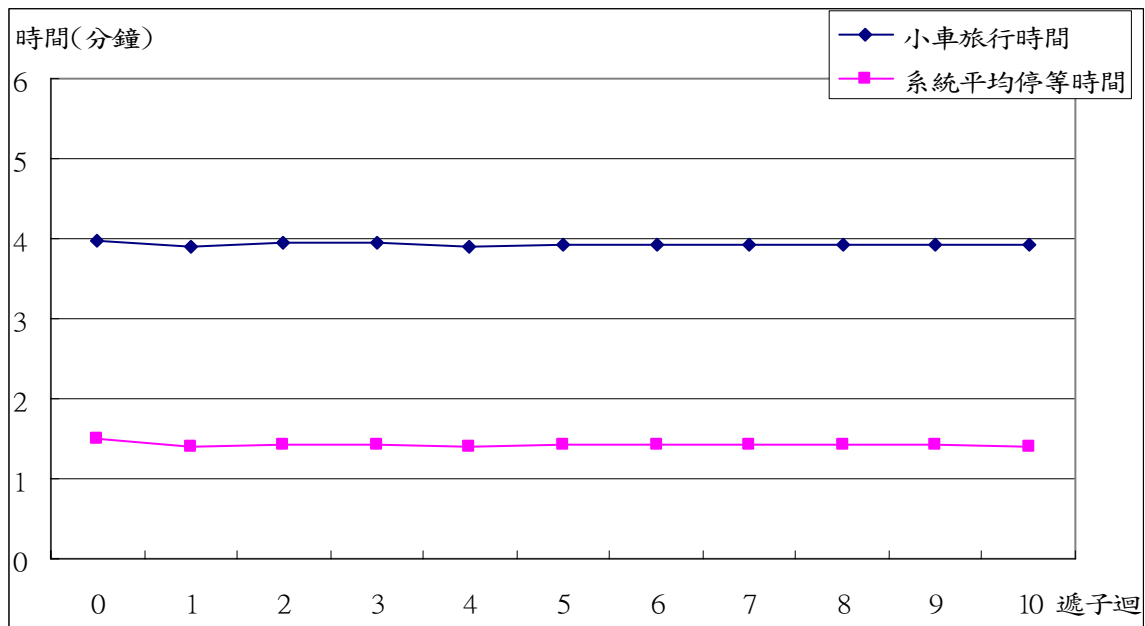


圖 5.14 常態流量需求(0.3)時間趨勢圖

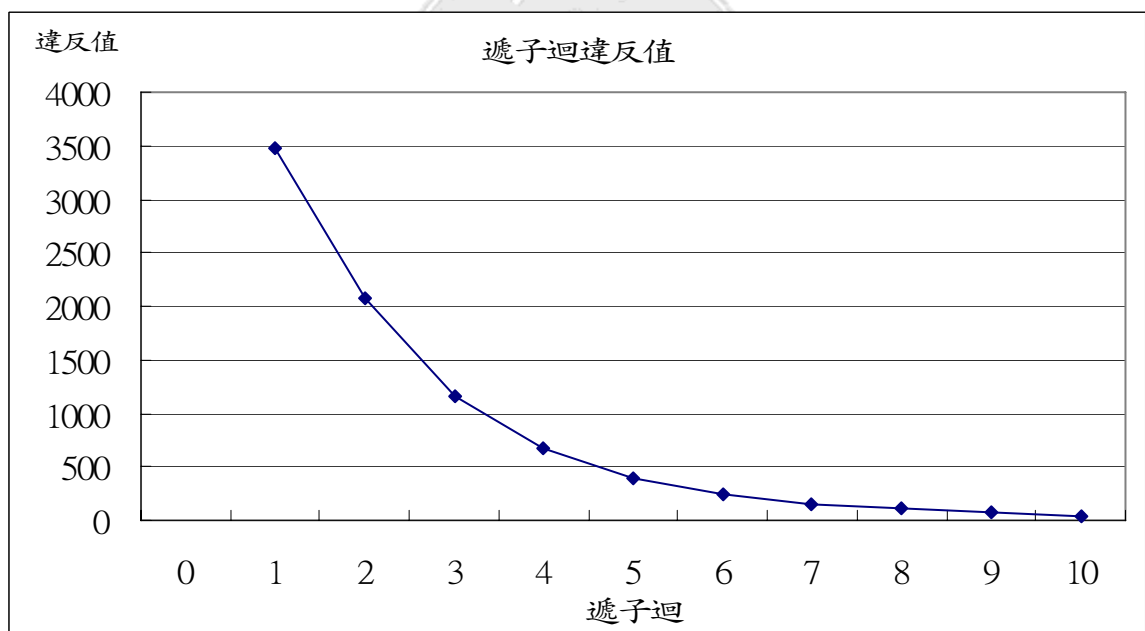


圖 5.15 常態流量需求(0.3)違反值

增量因子為 0.5 之實驗結果共產生 21,908 輛車。此實驗中車輛旅行時間由 5.25 分鐘增加至 5.51 分鐘後，便開始往 5.3 分鐘附近移動，而其中最低旅行時間為第 15 遞迴迴之 5.27 分鐘，在此實驗中旅行時

間會於 5.3 分鐘附近慢慢震盪後趨於平穩狀態。旅行距離則由 2399 公尺慢慢增加至 2412 公尺後車輛旅行距離趨於平穩狀態曲線。違反值由 4863 慢慢接近至收斂值。

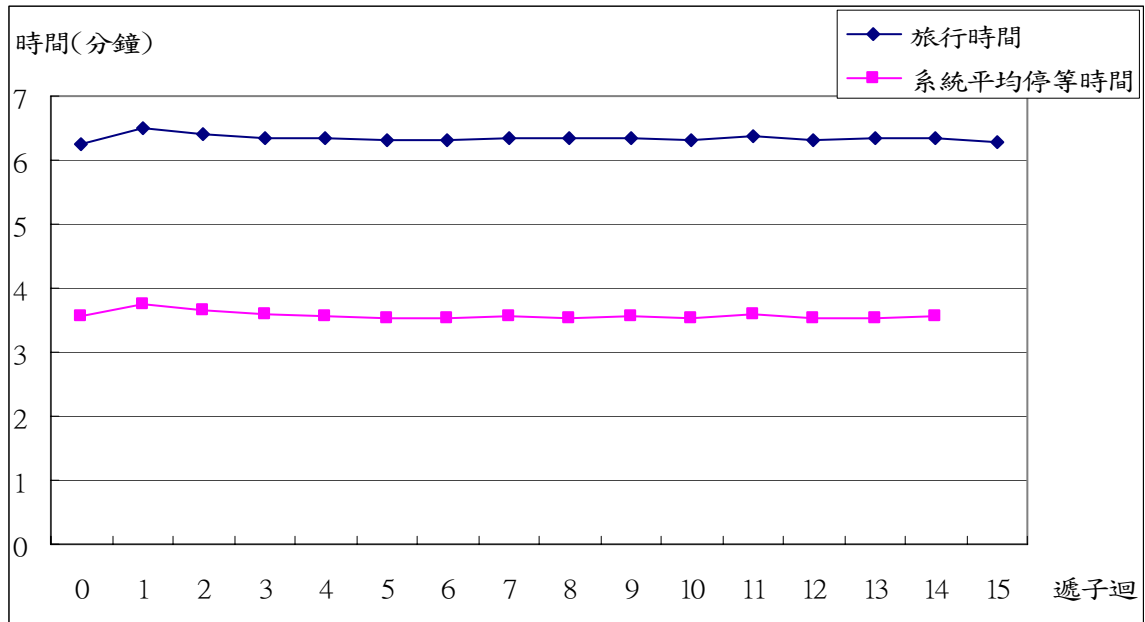


圖 5.16 常態流量需求(0.5)時間趨勢圖

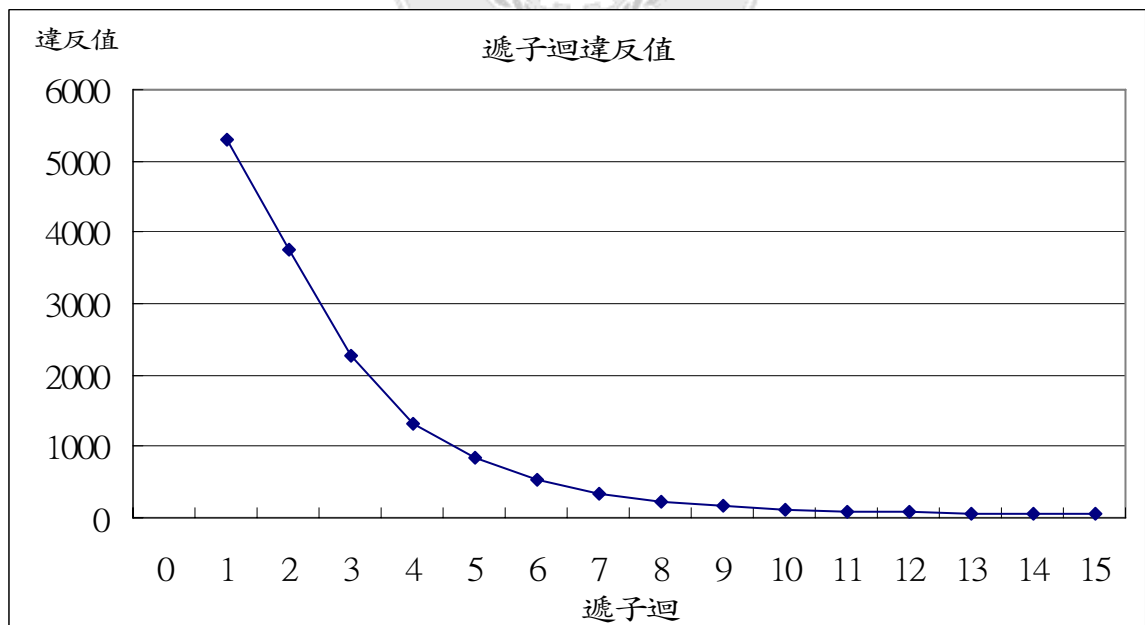


圖 5.17 常態流量需求(0.5)違反值

增量因子為 0.7 之實驗結果則產生了 30,727 輛車。此實驗中車輛旅行時間由 11.61 分鐘增加至 12.62 分鐘後，變開始往 11.3 分鐘附近移動，而其中最低旅行時間為第 6 個遞迴之 11.22 分鐘，實驗中旅行時間會往 12.3 分鐘附近慢慢震盪後趨於平穩狀態。旅行距離則由 2422 公尺慢慢增加至 2477 公尺後車輛旅行距離趨於平穩狀態曲線。違反值由 6779 慢慢減緩接近收斂值。

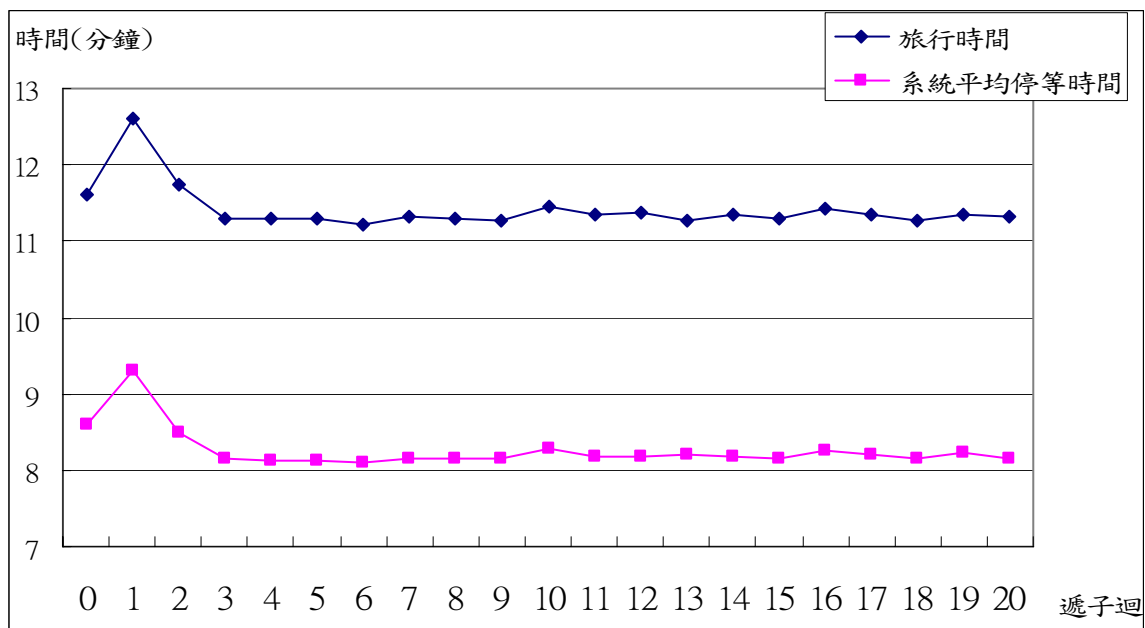


圖 5.18 常態流量需求(0.7)時間趨勢圖

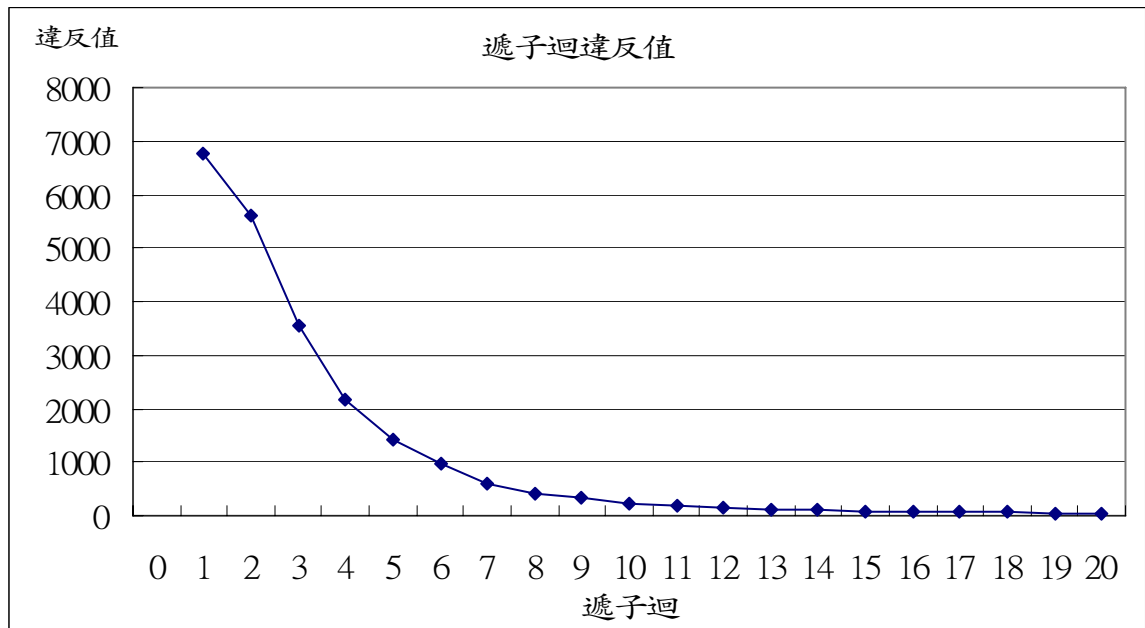


圖 5.19 常態流量需求(0.7)違反值

4. 動態使用者均衡實驗綜合說明

經由三種流量需求分佈與三個不同增量因子假設觀察動態使用這均衡實驗結果以旅行時間與旅行距離進行說明。圖 5.20 至 5.22 為同增量因子下，不同流量分佈之旅行時間趨勢圖。圖 5.23 至 5.25 則為相同情況下之違反值趨勢圖。

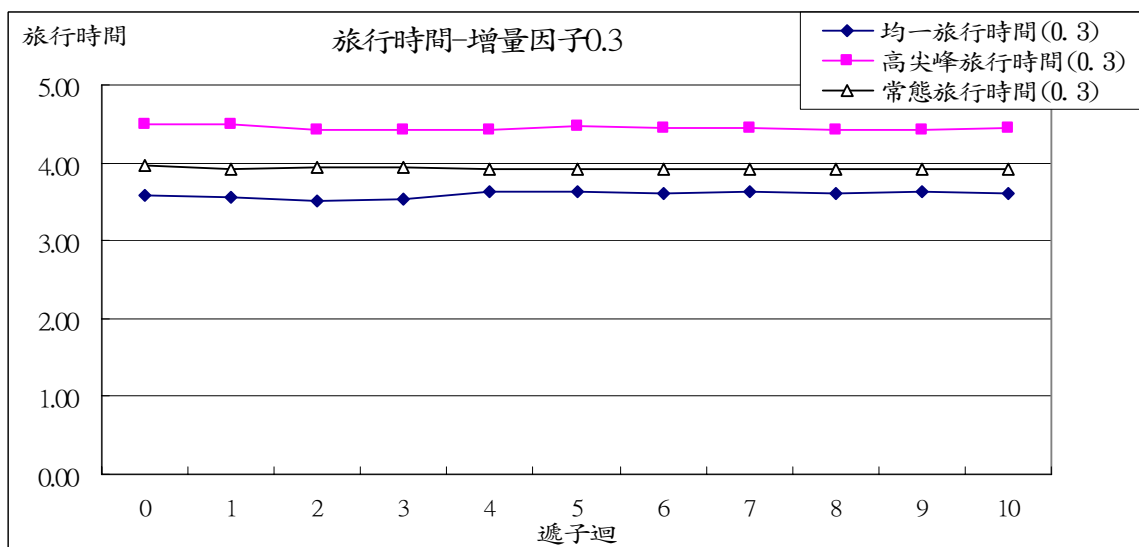


圖 5.20 增量因子 0.3 之旅行時間趨勢圖

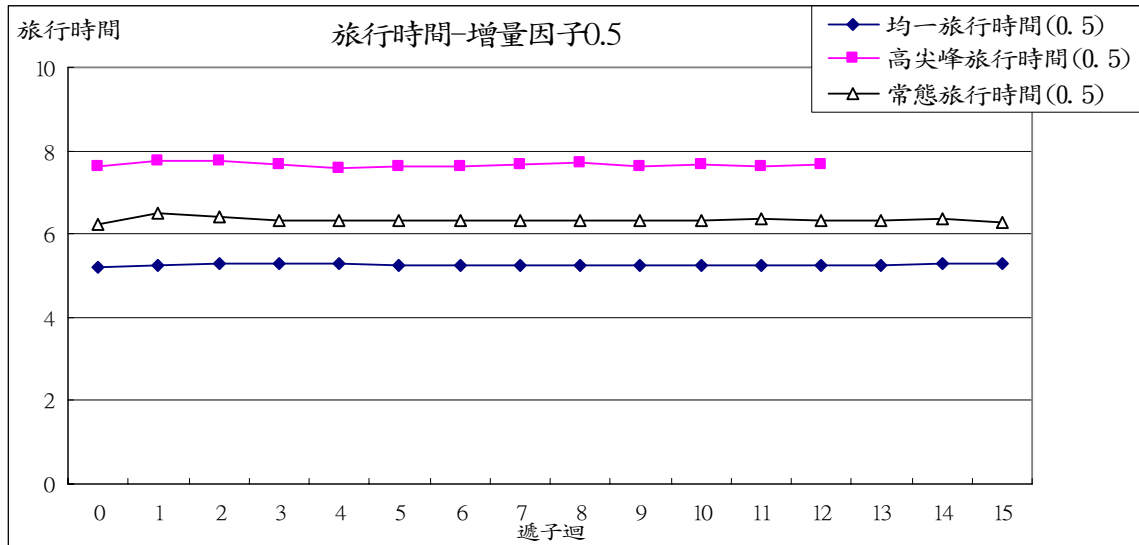


圖 5.21 增量因子 0.5 之旅行時間趨勢圖

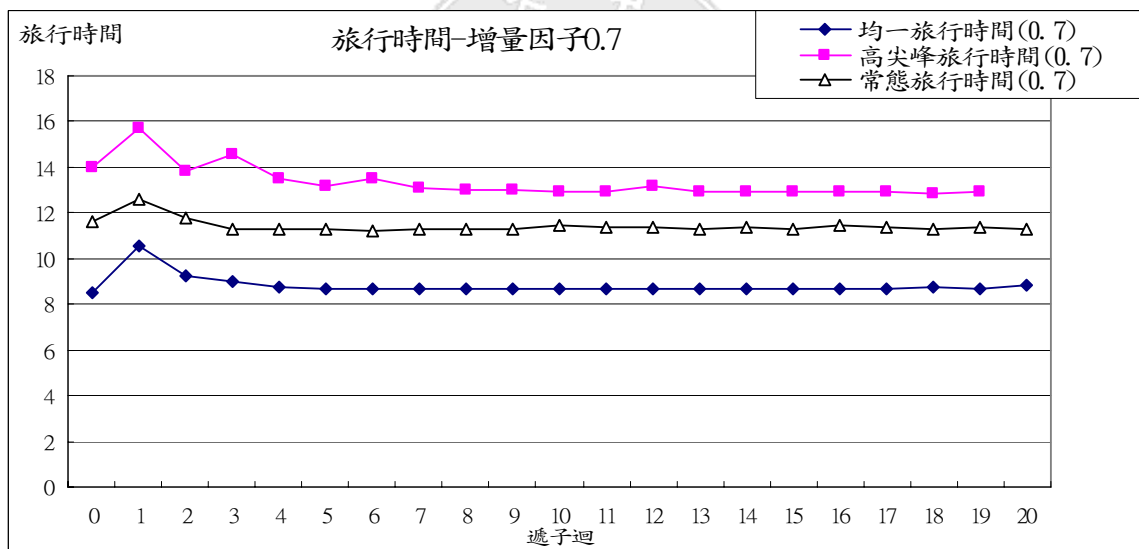


圖 5.22 增量因子 0.7 之旅行時間趨勢圖

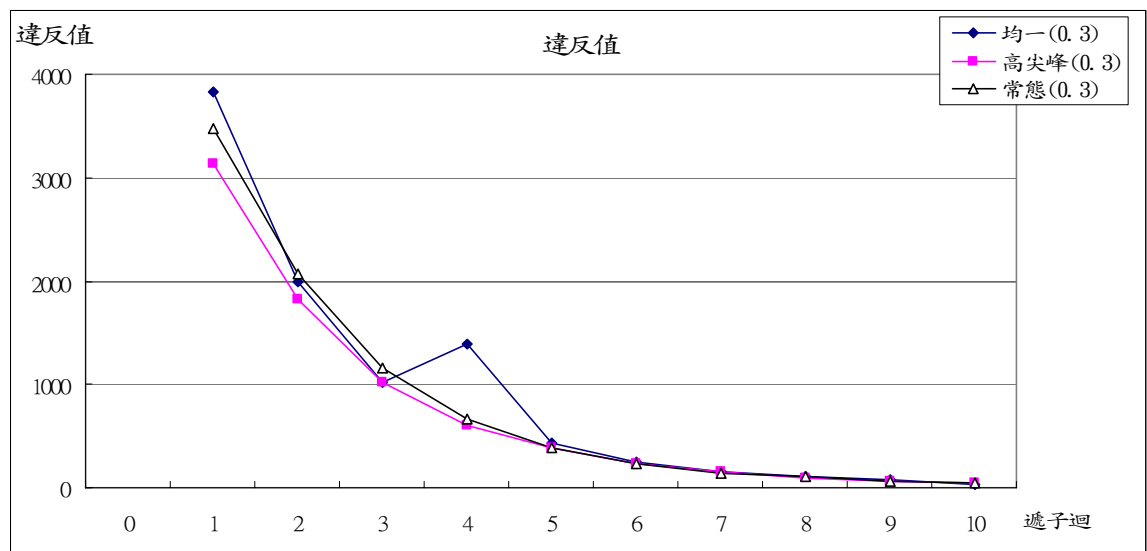


圖 5.23 增量因子 0.3 之違反值趨勢圖

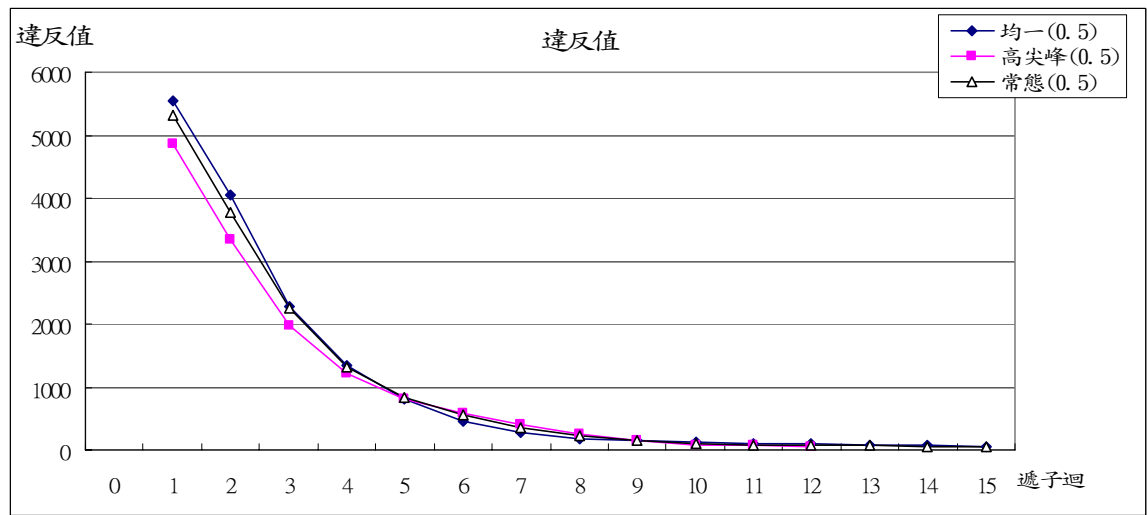


圖 5.24 增量因子 0.5 之違反值趨勢圖

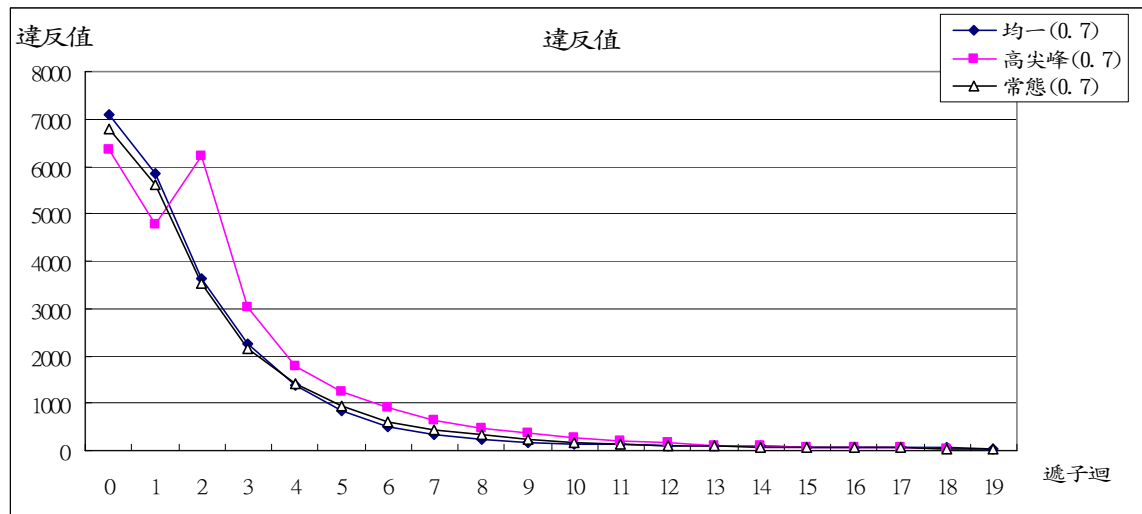


圖 5.25 增量因子 0.7 之違反值趨勢圖

以下針對單一車種實驗結果進行綜合說明：

1. 系統中車輛旅行時間，在增量因子為 0.3 與 0.5 時其曲線變化幅度較為平緩，而在增量因子 0.7 時，會出現部分震盪曲線後再趨於平穩，此主要係因為路網在出現擁擠狀況後，路徑會更換到較遠路徑上造成旅行時間增加。
2. 系統平均旅行距離中，增量因子為 0.3 時，由於路網中未呈現擁擠狀態，因此旅行距離曲線會呈現部分震盪現象後再趨於平緩，而於增量因子 0.5 與 0.7 時，路網中車輛則有擁擠狀態發生，因此會將部分車輛移轉至較不擁擠路段，因此對於車輛旅行距離則會慢慢增加。
3. 就整體實驗結果而言，以高尖峰與常態分佈流量所呈現出 DUE 曲線變化最為明顯，對此 DUE 條件於擁擠狀況下路網會有改善車輛之旅行時間，其改善後時間會比原本第 0 次遞迴迴的旅行時間小。
4. 在違反值觀察上，當增量因子設定為 0.3 時，其流量分佈大致上都會比高尖峰與常態分佈快達到收斂值，主要原因為路網中車輛未呈現壅塞狀況，車輛所使用之路徑變動不大。而在增量因子 0.5 與 0.7 狀況下，路網已呈現出壅塞狀況，車輛會考量避開壅塞路段後，因此，不斷產生不同時段下較好之路徑提供給予車輛使用。

5.3 混合車流實驗

本小節針對混合車流下動態使用者均衡進行實驗，主要以常態流量分佈並針對增量因子 0.5 與 0.7 進行實驗測試。表 5.10 至 5.11 為增量因子 0.5 與 0.7 測試結果，表中資料包含車輛旅行時間、旅行距離以及各遞子迴之違反值。

表 5.10 混合車流下(0.5)實驗數值

遞子迴	總車輛數(輛)	小車車輛數(輛)	小車旅行時間(分鐘)	小車旅行距離(公尺)	違反值	機車車輛數(輛)	機車旅行時間(分鐘)	機車旅行距離(公尺)	違反值
0	43,879	21,892	8.91	2467.96		21,987	5.96	1673.65	
1	43,879	21,892	9.32	2464.75	3,787	21,987	6.01	1674.51	2,401
2	43,879	21,892	9.25	2462.67	2,729	21,987	6.30	1678.17	1,868
3	43,879	21,892	8.88	2464.95	2,016	21,987	5.72	1684.02	1,672
4	43,879	21,892	9.18	2465.17	1,357	21,987	5.72	1685.54	1,251
5	43,879	21,892	9.00	2464.18	947	21,987	5.79	1685.36	916
6	43,879	21,892	9.02	2464.44	667	21,987	5.79	1686.54	800
7	43,879	21,892	9.01	2463.93	433	21,987	5.82	1687.36	669
8	43,879	21,892	9.13	2463.18	323	21,987	5.79	1688.18	573
9	43,879	21,892	9.01	2463.11	227	21,987	5.84	1688.52	424
10	43,879	21,892	9.03	2462.90	171	21,987	5.81	1688.43	298
11	43,879	21,892	8.99	2462.77	132	21,987	5.80	1688.34	222
12	43,879	21,892	9.11	2463.00	107	21,987	5.78	1688.16	152
13	43,879	21,892	9.02	2463.15	100	21,987	5.76	1688.13	98
14	43,879	21,892	9.05	2463.21	87	21,987	5.79	1688.13	57
15	43,879	21,892	9.22	2462.90	85	21,987	6.00	1688.11	41
16	43,879	21,892	9.14	2463.01	84	21,987	5.85	1688.07	42
17	43,879	21,892	9.04	2462.89	75	21,987	5.79	1687.95	32
18	43,879	21,892	9.18	2462.85	64	21,987	5.77	1687.93	30
19	43,879	21,892	9.09	2462.81	49	21,987	5.77	1687.95	30

表 5.11 混合車流下(0.7)實驗數值

遞子迴	總車輛數 (輛)	小車車輛數 (輛)	小車旅行時間 (分鐘)	小車旅行距離 (公尺)	違反值	機車車輛數 (輛)	機車旅行時間 (分鐘)	機車旅行距離 (公尺)	違反值
0	61,400	30,771	25.97	2481.55		30,629	12.86	1733.01	
1	61,400	30,771	26.06	2483.65	6,037	30,629	13.54	1733.77	3,453
2	61,400	30,771	39.10	2530.17	4,986	30,629	19.02	1765.37	3,151
3	61,400	30,771	23.47	2542.54	3,598	30,629	13.12	1785.94	2,964
4	61,400	30,771	21.90	2542.89	2,627	30,629	13.19	1786.36	2,098
5	61,400	30,771	21.27	2546.33	2,023	30,629	12.81	1789.51	1,751
6	61,400	30,771	21.71	2547.76	1,522	30,629	11.95	1792.27	1,541
7	61,400	30,771	21.76	2548.97	1,171	30,629	11.97	1792.35	1,298
8	61,400	30,771	20.29	2549.13	794	30,629	11.59	1793.10	1,090
9	61,400	30,771	21.35	2549.86	568	30,629	11.70	1793.51	933
10	61,400	30,771	20.01	2550.41	416	30,629	11.49	1793.94	839
11	61,400	30,771	19.74	2550.92	298	30,629	11.47	1793.51	702
12	61,400	30,771	19.77	2550.79	248	30,629	11.22	1793.59	902
13	61,400	30,771	20.67	2551.62	263	30,629	11.42	1792.34	481
14	61,400	30,771	20.85	2552.22	183	30,629	11.62	1792.13	394
15	61,400	30,771	20.61	2552.33	160	30,629	11.49	1792.13	305
16	61,400	30,771	21.37	2552.89	144	30,629	11.53	1791.57	228
17	61,400	30,771	19.71	2552.81	126	30,629	11.29	1791.31	157
18	61,400	30,771	19.93	2552.53	124	30,629	11.36	1791.18	123
19	61,400	30,771	19.80	2552.66	117	30,629	11.32	1790.97	89
20	61,400	30,771	20.01	2552.28	105	30,629	11.27	1789.95	74
21	61,400	30,771	19.90	2552.54	95	30,629	11.33	1789.45	70
22	61,400	30,771	19.54	2552.61	85	30,629	11.20	1789.11	53
23	61,400	30,771	19.79	2552.66	79	30,629	11.25	1788.83	41
24	61,400	30,771	19.62	2552.61	67	30,629	11.26	1788.75	30
25	61,400	30,771	19.54	2552.35	59	30,629	11.18	1788.71	23
26	61,400	30,771	19.46	2552.17	49	30,629	11.17	1788.27	17

以下分別針對增量因子 0.5 與 0.7 結果進行說明，並綜合討論實驗結果。

1. 小車與機車增量因子設定為 0.5

此實驗中小車旅行時間由原 8.91 分鐘慢慢改善至 8.88 分鐘後旅行時間曲線於 9 分鐘附近呈現小幅度震盪。機車旅行時間則由 5.96 分鐘改善至 5.7 分鐘後，旅行時間於 5.7 分鐘附近小幅度震盪，如圖 5.26 所示。車輛旅行時間於初期呈現成長曲線，於第 2 個遞子迴時旅行時間達到最高後，各車種旅行時間曲線呈現減緩。在違反值之觀察上，機車於第 15 個遞子迴達到收斂值，小車則需於第 19 個遞子迴才可符合收斂條件，如圖 5.27 所示。

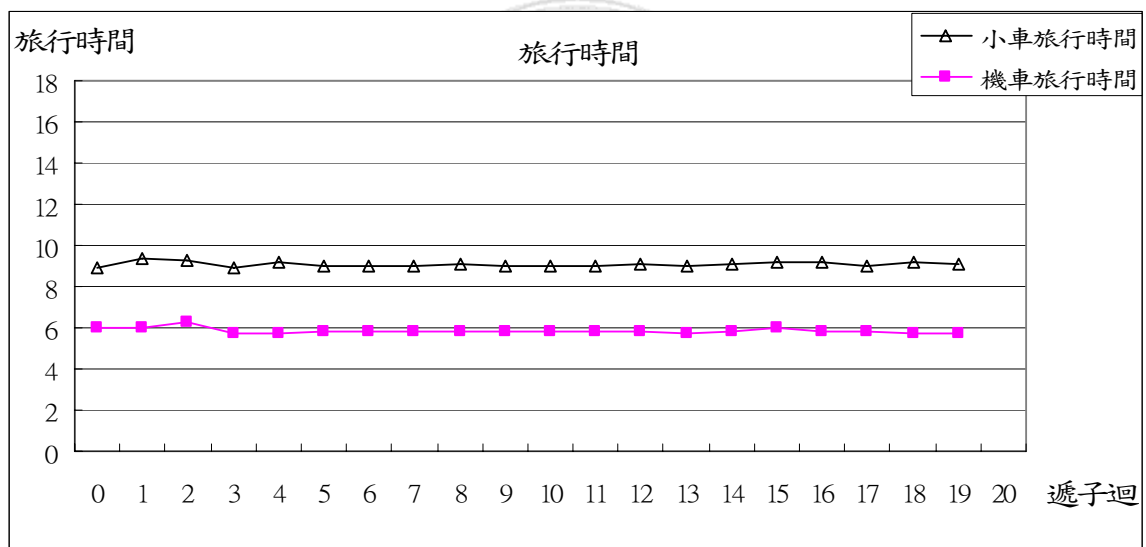


圖 5.26 混合車流下(0.5)旅行時間趨勢圖

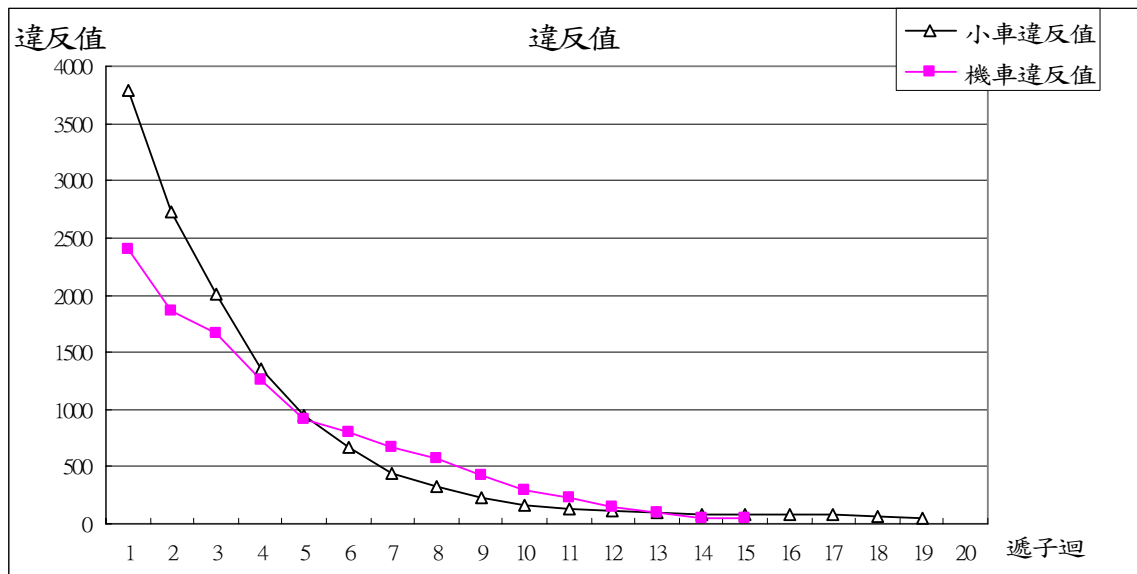


圖 5.27 混合車流下(0.5)違反值趨勢圖

2. 小車與機車增量因子設定為 0.7

實驗中小車旅行時間由原 25.97 分鐘增加至 39.10 分鐘後，小車旅行時間慢慢改善至 19.80 分鐘附近小幅度震盪。機車旅行時間則由原 12.86 分鐘增加至 19.02 分鐘後，旅行時間慢慢改善至 11.20 分鐘左右後逐漸趨於平穩。違反值方面，機車違反值於第 23 遞迴迴時達到收斂值，小車則在第 26 遞迴迴時符合收斂值。旅行時間趨勢圖，如圖 5.28 所示。違反值趨勢圖，如圖 5.29 所示。

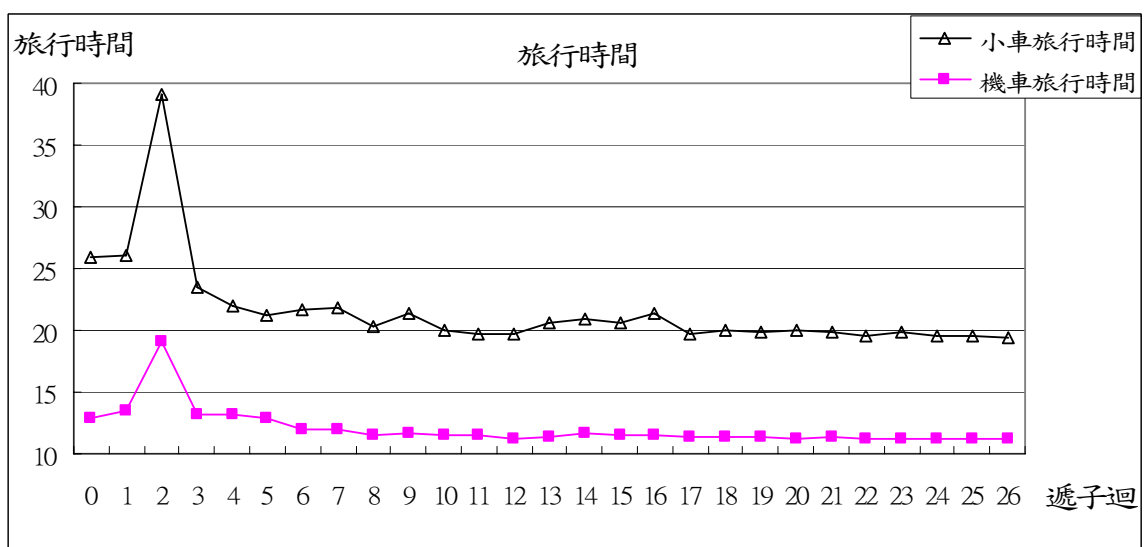


圖 5.28 混合車流下(0.7)旅行時間趨勢圖

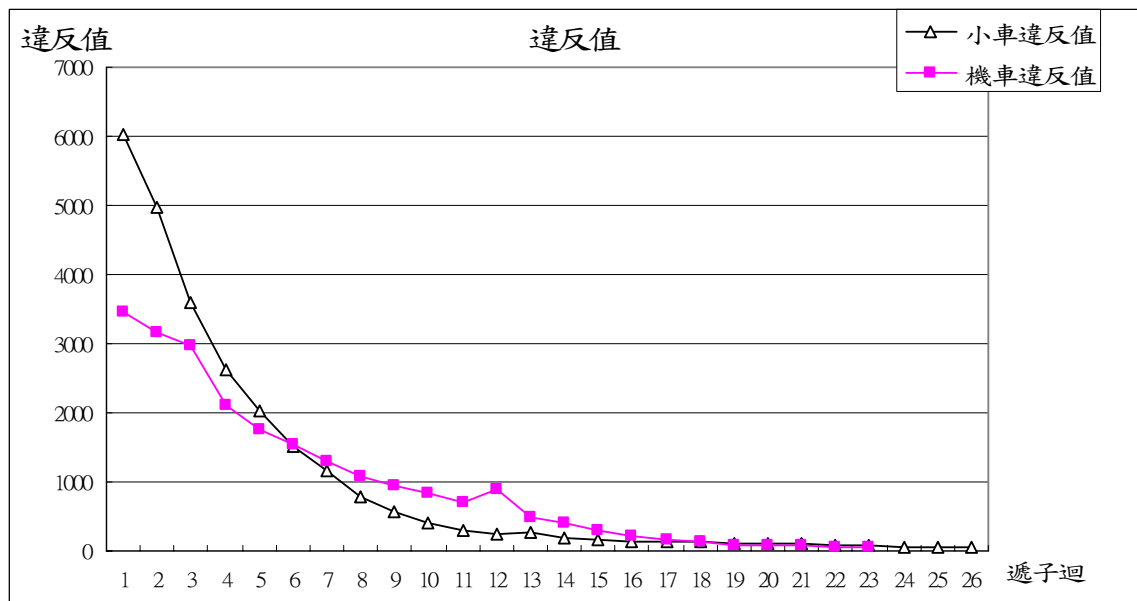


圖 5.29 混合車流下(0.7)違反值趨勢圖

以下針對混合車流實驗結果進行綜合說明：

1. 分別計算兩增量因子的旅行時間改善百分比，其旅行時間改善百分比為第 0 次遞迴與最低旅行時間差值所計算出。於增量因子 0.5 時小車與機車改善百分比分別為 0.34% 與 4.03% ，而當增量因子為 0.7 時小車與機車改善百分比分別為 50.48% 與 42.60% 。在混合車流實驗中，當路網達到擁擠狀況時，對於小車旅行時間與機車旅行時間確實能改善。
2. 實驗測試中小車與機車旅行距離中，其旅行距離曲線皆是呈現成長曲線圖，其曲線會增加至某一距離後趨於平緩，其結果反映出當路網擁擠時，部份車輛會依據指派原則使用較遠路段避開擁擠路段，造成車輛所旅行距離增加。
3. 違反值收斂方面，其機車收斂值較小車早達到收斂值，而小車收斂曲線圖較為平滑，機車收斂曲線則呈現出些許的震盪慢慢接近收斂值。
4. 實驗中機車旅行時間改善幅度不大，其主要原因為研究中將路口延滯時間與兩段式左轉忽略不計，以至當路口有擁擠狀況發生時機車無法將擁擠狀況反應至機車旅行時間。

第六章 結論與建議

本研究主要於探討混合車流下模擬式動態交通指派模式，並於一實驗路網中進行相關實驗測試，以下綜合研究過程及實驗結果，提出下列結論與建議。

6.1 結論

1. 本研究建立混合車流下模擬式動態指派模式架構，依據車輛型態(小車與機車)進行構建模擬式動態指派模式利用，並以 DynaTAIWAN 模擬混合車流下車輛資料，包含路徑及路段速度等。
2. 使用 Visual C++ 建立混合車流下動態交通量指派之使用者均衡演算法，演算法包含了模擬時段路徑與依時性最短路徑比較、MSA 演算及產生下一遞子迴車輛屬性資料，並分別以模組觀念進行構建，包含 DynaTAIWAN、成本計算、依時性最短路徑、流量分配及車輛產生等五個模組，而後續如有不同考量需進行修正時，只需將要調整之模組進行修正或更換即可。
3. 於動態指派模式實驗過程中，依時性最短路徑計算路徑因需將各指派時段之路徑輸出，且此模組運作時需較多記憶體空間進行運算，故其演算過程需較長時間進行運算。
4. 流量重新分配主要是以 MSA 方法進行重新分配，並經由流量分配後計算該路徑流量改變量並觀察其違反值，實驗中小車與機車路徑違反值皆以較平滑曲線方式接近研究設定收斂值。
5. 研究中數值實驗主要測試單一車種(小車)與混合車流(小車與機車)於路網中之動態使用者之反應，以下針對各實驗結果進行綜合說明。

(1) 單一車種

小車旅行時間初期會呈現增加情況，其主要狀況初始模擬提供較佳路徑，以致於車輛旅行時間會靠近初始旅行時間附近

震盪。

(2) 混合車流

- i. 於混合車流實驗中，機車比小車早達到收斂值。而小車收斂曲線圖較為平滑，機車收斂曲線則呈現出些許的震盪慢慢接近收斂值。
- ii. 就實驗中當增量因子 0.5 時車輛旅行時間改善百分比變化不大，但當增量因子 0.7 時，小車與機車旅行時間改善百分比可提升許多，說明當路網車輛達到壅塞時，可透過指派方式將路網中擁擠車輛指派至為擁擠路段，以減緩車輛旅行時間。

6.2 建議

1. 本研究已對一般車流狀況進行測試，後續研究可考慮 VMS 資訊提供與事件發生(如：事故、道路施工等)等不同交通情形進行進一步的研究與測試。
2. 本研究中已建立模擬式動態指派模式架構並針對單一車種(小車)與混合車流(小車與機車)進行測試，後續可加入其他車種(如大車)進行模擬式動態指派模式測試。
3. 研究中所使用之程式目前分為三部份執行，包含第一部份負責模擬初步資料 DynaTAIWAN、第二部份程式為動態使用者均衡(DUE)程式演算過程，第三部份則為 DUE 進行下一個遞迴子迴時所修改之 DynaTAIWAN 程式進行模擬。因此，執行程式時需依序手動執行每一個程式，需耗費部分時間，後續可考量將三部分程式連結成自動模式以節省程序。
4. 測試不同收斂條件，研究中收斂條件主要是以違反值統計方式進行判斷，後續可建立以違反值路徑百分比作為判斷是否達到收斂條件。
5. 對於測試路網為中型路網(50 節點路網)，後續可針對大型路網進行測試，以觀察於大型路網時動態使用者均衡之變化狀況。

參考文獻

1. 交通部(2001),「台灣地區智慧型運輸系統綱要計劃」。
2. 伍靜怡(2002),即時應變事故之動態交通量指派方法論,台灣大學土木工程研究所碩士論文。
3. 李俊賢(1996),在靜態模型中運用傅立葉轉換分析隨機性動態旅行時間之研究,國立台灣大學土木工程學研究所博士論文。
4. 李達權(2002),模擬式動態交通指派模式—系統最佳化原則之研究,逢甲大學交通工程與管理學研究所碩士論文,民國 91 年 6 月。
5. 林蔚明(2004),路口延滯下路徑演算法之研究,逢甲大學交通工程與管理學研究所碩士論文。
6. 邱敏華(2003),高速公路事故路段動態旅行時間模式之研究,國立臺灣大學土木工程學研究所碩士論文。
7. 胡大瀛(2001),「動態路網模擬指派模式之建立」,運輸計劃季刊,第三十卷第一期,pp.1-32。
8. 曾莉莉(1994),高速公路動態路段旅行時間函數研究,國立中央大學土木工程學研究所碩士論文。
9. 運研所(2004),區域智慧型運輸系統示範計畫-核心交通分析與預測系統(第一年期)。
10. Bliemer, M. C.J (2001), Analytical dynamic traffic assignment with interaction user-classes: theoretical advances and applications using a variational inequality approach. Ph.D. thesis, Delft University of Technology, The Netherlands.
11. Bliemer, M. C.J. and Bovy, P. H.L. (2003),” Quasi-variational inequality formulation of the multiclass dynamic traffic assignment problem”, Transportation Research Part B: Methodological, Volume: 37, Issue: 6, pp. 501-519.
12. Carey, M. (1986), “A Constraint Qualification for a Dynamic Traffic Assignment Model”, Transportation Science, Vol. 20, pp. 55-58.
13. Carey, M. (1987), “Optimal Time Varying Flows on Congested Networks”, Operations Research, Vol. 35, No. 1, pp. 58-69.
14. Chen, H. K. (1999), Dynamic Travel Choice Models: A Variational Inequality Approach, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag , pp. 320.
15. Dafermos, S.C. (1971), An extended traffic assignment model with applications to two-way traffic. Transportation Science 5, pp.366–389.

16. Dafermos, S.C. (1972), The traffic assignment problem for multiclass-user transportation networks. *Transportation Science* 6, pp.73–87.
17. Friesz, T. L., Luque, J., Tobin, R. L., and Wie, B. W. (1989), “Dynamic Network Traffic Assignment Considered as a Continuous Time Optimal Control Problem”, *Operations Research*, Vol. 37, No. 6, pp. 893-901.
18. Ghali, M. O. and Smith, H. J. (1992), “Optimal Dynamic Traffic Assignment of a Congested City Network”, *Proceedings of the Second International Capri Seminar on Urban Traffic Network*, Capri, Italy.
19. Ho, J. K. (1980), “A Successive Linear Optimization Approach to the Dynamic Traffic Assignment Problem”, *Transportation Science*, 14, pp. 295-305.
20. Hoogendoorn, S. P. and Bovy P. H.L. (2000), “Continuum modeling of multiclass traffic flow”, *Transportation Research Part B* 34, pp.123-146.
21. Hoogendoorn, S. P. and Bovy P. H.L. (2001), *Platoon-Based Multiclass Modeling of Multilane Traffic Flow, Networks and Spatial Economics*, pp.137-166.
22. Hall, R. W. (1986), “The Fastest Path through a Network with Random Time-Dependent Travel Times,” *Transportation Science*, Vol.20, No.3, pp.182-188.
23. Lo, H.K. (1999), A dynamic Traffic Assignment Formulation That Encapsulates the Cell-Transmission Model. In: Ceder, A. (Ed.), *Transportation and Traffic Theory*. Pergamon, Oxford, pp. 327–350.
24. Lo, H.K., Ran, B. and Hongola, B. (1996), “Multiclass dynamic traffic assignment model: Formulation and computational experiences.”, *Transportation Research Records* 1537, pp.74–82.
25. Merchant, D. K. and Nemhauser, G. L. (1978a), “Model and an Algorithm for the Dynamic Traffic Assignment Problems”, *Transportation Science*, Vol. 12, No. 3, pp. 183-199.
26. Merchant, D. K. and Nemhauser, G. L. (1978b), “Optimality Conditions for a Dynamic Traffic Assignment Model”, *Transportation Science*, Vol. 12, No. 3, pp. 200-207.
27. Peeta, S.(1994), *System Optimal Dynamic Traffic Assignment in Congested Networks with Advanced Information Systems*, Ph.D. dissertation, the University of Texas at Austin.
28. Rickert, M. and Nagel, K. (2001), *Dynamic traffic assignment on parallel computers in TRANSIMS*, *Future Generation Computer Systems* 17,

- pp.637-648.
29. Ran, B. and Boyce, D. E. (1994), "Dynamic Urban Transportation Network Models:Theory and Implications for Intelligent Vehicle Highway Systems,"Lecture Notes in Economics and Mathematical Systems 417, Springer-Verlag, New York.
 30. Ran, B., and Boyce, D.E (1996), Modeling Dynamic Transportation Networks: An Intelligent Transportation System Oriented Approach, Springer-Verlag, Berlin, pp. 356.
 31. Ran, B., Boyce, D. E., and LeBlance, L. J. (1993), "A New Class of Instantaneous Dynamic User-optimal Traffic Assignment Models", Operations Research, Vol. 41, No. 1.
 32. Ran, B., Lo, H.K., and Boyce, D.E . (1996), A formulation and solution algorithm for a multi-class dynamic traffic assignment model. In: Lesort, J.-B. (Ed.), Transportation and Traffic Theory. Pergamon, Oxford, pp. 195–216.
 33. Ran, B., Roupail, N. M., Tarko, A., and Boyce, D. E. (1997), "Toward a Class of Link Travel Time Functions for Dynamic Assignment Models on Signalized Networks," Transportation Research Part B, Vol.31, No.4, pp.277-290.
 34. Sheffi, Y. (1985), Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods, Prentice-Hall, NJ,.
 35. Wardrop, J. G. (1952), "Some Theoretical Aspects of Road Traffic Research", Proceedings, Institution of Civil Engineers II(1), pp. 325-378.
 36. Wynter, L.M. (1995), Advances in the theory and application of the multiclass traffic assignment problem. Ph.D. thesis, Ecole National des Ponts et Chaussées, France.
 37. Ziliaskopoulos, A. K. and Mahmassani, H. S. (1996), "A Note On Least Time Path Computation Considering Delays and Prohibitions for Intersection Movements," Transportation Research Part B, Vol.30, No.5, pp.359-367.
 38. Ziliaskopoulos, A. K. and Mahmassani, H. S. (1996), "Time-Dependent, Shortest-Path Algorithm for Real-Time Intelligent Vehicle Highway System Applications," Transportation Research Record 1408, pp.94-100.
 39. Ziliaskopoulos, A. K., Kotzinos, D. and Mahmassani, H. S. (1997), "Design and Implementation of Parallel Time-Dependent Least Time Path Algorithm for Intelligent Transportation System Applications," Transportation Research PartC, Vol.5, No.2, pp.95-107.

附錄

檔案名稱：MSA.h

```
#ifndef __DTA_MSA_H__  
#define __DTA_MSA_H__
```

```
#include "../DTASim.h"  
#include "../DTADEF.h"  
#include <fstream>  
#include <iomanip>  
#include <vector>  
#include <iostream>  
#include <cstdio>  
using namespace std;
```

// 建立 OLD 路徑資料結構，統計同一起迄點路徑流量

```
struct OLDPath  
{  
    USI          OLDDID;           //車輛編號  
    USI          OLDType;          //車輛屬性  
    USI          OLDBehavior;       // Behavior 行為  
    USI          OLDFamiliarity;    // 熟悉度  
    USI          OLDDAST;           //指派時段  
    USI          OLDOrg;            //起點  
    USI          OLDDes;            //迄點  
    USI          OLDPathflow;       //路徑流量  
    USI          OLDNodenum;        //節點數  
    USI          OLDNodesum;        //節點數和  
    vector<unsigned int> OPath;      //路徑  
};  
typedef vector<OLDPath> OPathlist;
```

// 建立 OLD 路徑資料結構，統計同一起迄點路徑

```
struct OOLDPath  
{  
    USI          OOLDID;  
    USI          OOLDType;  
    USI          OOLDBehavior;  
    USI          OOLDFamiliarity;  
    USI          OOLDDAST;  
    USI          OOLDOrg;  
    USI          OOLDDes;  
    USI          OOLDPathflow;  
    USI          OOLDNodenum;  
    USI          OOLDNodesum;  
    vector<unsigned int> OOPath;  
};  
typedef vector<OOLDPath> OOPathlist;
```

// 建立 TDSP 路徑資料結構

```
struct TDSPPath  
{
```

```

        USI            TDSPAST;           //TDSP 指派時段
        USI            TDSPOrg;          //TDSP 起點
        USI            TDSPDes;          //TDSP 迄點
        USI            TDSPPathflow;     //TDSP 路徑流量
        USI            TDSPNodenum;      //TDSP 節點數
        USI            TDSPNodesum;      //TDSP 節點數和
        std::vector<unsigned int> TPath;  //TDSP 路徑
    };
    typedef vector<TDSPPath> TDsppathlist;

//建立 MSA 路徑資料結構 MSA_2
struct MSA_Trave
{
    USI            MSAPathflow;          //MSA 路徑流量
    USI            MSA_NPathflow;        //MSA 路徑流量
    USI            MSANodenum;           //MSA 節點數
    USI            MSANodesum;           //MSA 節點數和
    double         MSAViolation;          //MSA 違反值
    double         ViolationSum;          //違反值統計
    double         MSAPrcentage;          //路徑百分比
    std::vector<unsigned int> MPath;      //MSA 路徑
};
typedef vector<MSA_Trave> MSATralist;

// 建立 MSA 路徑資料結構 MSA_1
struct MSAPath
{
    USI            MSAAST;               //MSA 指派時段
    USI            MSAOrg;               //MSA 起點
    USI            MSADes;               //MSA 迄點
    MSATralist     Newpathlist;
};
typedef vector<MSAPath> Msapathlist;

struct ODFlow
{
    USI            ODAST;
    USI            ODOrg;
    USI            ODDes;
    USI            ODPFlow;
};
typedef vector<ODFlow> ODflowlist;

class CMSA
{
public:
    CMSA();
    ~CMSA();
    bool SORT();
    bool SUM();
    bool SSUM();
    bool MSA();
    int            ODID;    //ODflow 指標
    int            TDSPID;

```

private:

```
    OPathlist        _OPathlist;
    OLDPath           _OLDPath;
    OOPathlist        _OOPathlist;
    OOLDPath          _OOLDPath;
    TDSPPath          _Tdspath;
    TDsppathlist      _Tdspathlist;
    MSAPath           _Msapath;
    Msapathlist       _Msapathlist;
    MSA_Trave         _MSATrave, _MSATDSP;
    ODFlow            _ODFlow;
    ODflowlist        _ODflowlist;
};
```

#endif

檔案名稱：MSA.cpp

/*MSA 主要負責產生下一各遞迴子迴路徑車輛數產生 車輛數重新分配*/

```
#include "MSA.h"
#include <cmath>
const int Violation_MAX=50; //違反值設定為 50
bool Isexit;
CMSA::CMSA(){ }
CMSA::~CMSA(){ }
bool CMSA::SUM()
{
    bool Isexit;
    int tInterval;

    //車輛統計變數宣告
    int Totalflow=0;
    int Totalnewflow=0;
    cout << endl;
    cout << "統計各指派時段車輛數開始!!" << endl;
    cout << endl;
    cout << "讀取路徑資料.....Carpath.dat!!" << endl;
    // 讀取 MSAinput 基本資料 (Carpath data)
    ifstream ifRead("../MSAInput//Carpath.dat", ios::in);

    // 檢查是否開檔失敗？
    if(ifRead.fail() != 0)
    {
        cout<< "Cannot open file " << "Carpath.dat" << "!\n";
    }
    while(ifRead.peek()!=EOF)
    {
        ifRead >> _OLDPath.OLDID;
        ifRead >> _OLDPath.OLDType;
        ifRead >> _OLDPath.OLDBehavior;
        ifRead >> _OLDPath.OLDFamiliarity;
        ifRead >> _OLDPath.OLDAST;
        ifRead >> _OLDPath.OLDOrg;
        ifRead >> _OLDPath.OLDDes;
        ifRead >> _OLDPath.OLDPathflow;
```



```

ifRead >> _OLDPath.OLDNodenum;
ifRead >> _OLDPath.OLDNodesum;
_OLDPath.OPath.clear();
//讀取路徑
for (USI uTmp = 0; uTmp < _OLDPath.OLDNodenum; uTmp++)
{
    int uOPath=0;
    ifRead >> uOPath;
    _OLDPath.OPath.push_back(uOPath);
}
//讀入歷史資料車輛數進行相加
Totalflow += _OLDPath.OLDPathflow;
Isexit = false;
// 如果資料時間不在範圍內,就跳過不讀,
tInterval = ((int)(_OLDPath.OLDAST / 180)) * 180;
_OLDPath.OLDAST = tInterval;
//如果 vector 是空的
if ( _OPathlist.empty() == true )
{
    //設定 OLDPathflow 為 1
    _OLDPath.OLDPathflow = 1 ;
}
else { //else
    for(int i = 0; i < _OPathlist.size(); i++)
    { //for
        if ( _OPathlist[i].OLDAST == _OLDPath.OLDAST &&
            _OPathlist[i].OLDOrg == _OLDPath.OLDOrg &&
            _OPathlist[i].OLDDes == _OLDPath.OLDDes)
        {
            Isexit = true;
            _OPathlist[i].OLDPathflow = _OPathlist[i].OLDPathflow + 1 ;
            break;
        }
    } //for
} //else
//傳回 vector[i]
if (Isexit == false)
    _OPathlist.push_back(_OLDPath);
} //while 結束

//分各指派時段車輛總數統計
for(int m =0; m < _OPathlist.size(); m++)
{
    Totalnewflow += _OPathlist[m].OLDPathflow;
}

//路徑流量統計資料輸出 ODflow.dat
fstream SPFileOut;
SPFileOut.open("../MSAOut//ODflow.dat", ios::out);

//資料輸出
for(int k =0; k < _OPathlist.size(); k++)
{
    SPFileOut << setw(5) << _OPathlist[k].OLDAST
               << setw(5) << _OPathlist[k].OLDOrg
               << setw(5) << _OPathlist[k].OLDDes

```

```

        << setw(5) << _OPathlist[k].OLDPathflow;
    SPFileOut << endl;
}

SPFileOut.close();
ifRead.close();
//總車輛數輸出
cout << endl;
cout << "模擬車輛數總數：" << Totalflow << " 輛" << "\t";
cout << endl;
cout << "指派時段車輛數總數：" << Totalnewflow << " 輛";
cout << endl;
cout << endl;
cout << "統計各指派時段車輛數結束!!" << endl;
cout << endl;
return true;
}

bool CMSA::SSUM()
{
    bool Isexit;
    int tInterval;
    // 讀取 MSAinput 基本資料 (Carpath data)
    ifstream ifRead("../MSAInput//Carpath.dat", ios::in);

    // 檢查是否開檔失敗？
    if(ifRead.fail() != 0)
    {cout<< "Cannot open file " << "Carpath.dat" << "!\n";}
    cout << "路徑資料寫出.....";
    while(ifRead.peek() != EOF)
    {
        ifRead >> _OOLDPPath.OOLDID;
        ifRead >> _OOLDPPath.OOLDType;
        ifRead >> _OOLDPPath.OOLDBehavior;
        ifRead >> _OOLDPPath.OOLDFamiliarity;
        ifRead >> _OOLDPPath.OOLDAST;
        ifRead >> _OOLDPPath.OOLDOrg;
        ifRead >> _OOLDPPath.OOLDDes;
        ifRead >> _OOLDPPath.OOLDPPathflow;
        ifRead >> _OOLDPPath.OOLDNodenum;
        ifRead >> _OOLDPPath.OOLDNodesum;
        _OOLDPPath.OOPath.clear();
        //讀取路徑
        for (USI uTmp = 0; uTmp < _OOLDPPath.OOLDNodenum; uTmp++)
        {
            int uOOPath=0;
            ifRead >> uOOPath;
            _OOLDPPath.OOPath.push_back(uOOPath);
        }
        Isexit = false;
        // 如果資料時間不在範圍內,就跳過不讀,
        tInterval = ((int)(_OOLDPPath.OOLDAST / 180)) * 180;
        _OOLDPPath.OOLDAST = tInterval;
        for(int i = 0; i < _OOPathlist.size(); i++)
        {
            for
            if ( _OOPathlist[i].OOLDAST == _OOLDPPath.OOLDAST &&

```



```

        _OOPathlist[i].OOLDOrg == _OOLDPPath.OOLDOrg &&
        _OOPathlist[i].OOLDDes == _OOLDPPath.OOLDDes &&
        _OOPathlist[i].OOLDNodenum == _OOLDPPath.OOLDNodenum &&
        _OOPathlist[i].OOLDNodesum == _OOLDPPath.OOLDNodesum)
    {
        Isexit = true;
        _OOPathlist[i].OOLDPPathflow = _OOPathlist[i].OOLDPPathflow + 1 ;
        for (int a = 0; a < _OOPathlist[i].OOLDNodenum; a++)
        {
            int uuOOPath;
            uuOOPath = _OOPathlist[i].OOPath[a];
            _OOLDPPath.OOPath.push_back(uuOOPath);
        }
        break;
    }
}
} //for
//傳回 vector[i]
if (Isexit == false)
    _OOPathlist.push_back(_OOLDPPath);
} //while 結束
ifRead.close();
//路徑資料輸出 path.dat
fstream SPFileOut;
SPFileOut.open("../MSAOut/path.dat", ios::out);
//資料輸出
for(int k =0; k < _OOPathlist.size(); k++)
{
    SPFileOut << setw(5) << _OOPathlist[k].OOLDAST
    << setw(5) << _OOPathlist[k].OOLDOrg
    << setw(5) << _OOPathlist[k].OOLDDes
    << setw(5) << _OOPathlist[k].OOLDPPathflow
    << setw(5) << _OOPathlist[k].OOLDNodenum
    << setw(5) << _OOPathlist[k].OOLDNodesum;
    for (int p =0; p < _OOPathlist[k].OOLDNodenum; p++)
    {
        SPFileOut << setw(5) << _OOPathlist[k].OOPath[p];
    }
    SPFileOut << endl;
}
SPFileOut.close();
cout << "Ok!" << endl;
return true;
}

bool CMSA::MSA()
{
    int tInterval;
    bool check;
    // 讀取 MSAOut 基本資料 (path data)
    ifstream ifRead1("../MSAOut/path.dat", ios::in);
    // 檢查是否開檔失敗?
    if(ifRead1.fail() != 0)
    {
        cout<< "Cannot open file " << "path.dat" << "!\n";
        return false;
    }
    // 讀取 MSAInput 基本資料 (TDSPpath data)
    ifstream ifRead2("../MSAInput/TDSPpath.dat", ios::in);

```

```

// 檢查是否開檔失敗
if(ifRead2.fail() != 0)
{
    cout<< "Cannot open file " << "TDSPPath.dat" << "!\n";
    return false;
}
//讀取 OD 路徑總流量資料
ifstream ifRead3("./MSAOut//ODflow.dat", ios::in);
if(ifRead3.fail() !=0)
{
    cout<< "Cannot open file " << "ODflow.dat" << "!\n";
    return false;
}
cout << "Read ODflow data.....";
while(!ifRead3.eof())
{
    ifRead3 >> _ODFlow.ODAST;    //指派時段
    ifRead3 >> _ODFlow.ODOrg;    //起點
    ifRead3 >> _ODFlow.ODDes;    //迄點
    ifRead3 >> _ODFlow.ODPFlow;  //路徑總流量
    _ODflowlist.push_back(_ODFlow);
}
ifRead3.close();
cout << "Ok!" << endl;
cout << endl;
cout << "Read TDSP data.....";
//讀取 TDSP 資料
while(!ifRead2.eof())
{
    ifRead2 >> _Tdsppath.TDSPAST;
    ifRead2 >> _Tdsppath.TDSPOrg;
    ifRead2 >> _Tdsppath.TDSPDes;
    ifRead2 >> _Tdsppath.TDSPPathflow;
    ifRead2 >> _Tdsppath.TDSPNodenum;
    ifRead2 >> _Tdsppath.TDSPNodesum;
    _Tdsppath.TPath.clear(); //清掉上一條路徑資料
    //讀取路徑
    for (USI uTmp = 0; uTmp < _Tdsppath.TDSPNodenum; uTmp++)
    {
        int TPath=0;
        ifRead2 >> TPath;
        _Tdsppath.TPath.push_back(TPath);
    }
    _Tdsppathlist.push_back(_Tdsppath);
}
ifRead2.close();
cout << "Ok!" << endl;
cout << endl;
cout << "Path data 讀入 MSA struct.....Ok!" << endl ;
cout << endl;
cout << "MSA 演算法開始....." << endl;
int p;
int newflow = 0;    //新流量
int tmpO=0, tmpD=0, tmpT=0;
//遞迴 I

```

```

int I;
cout << endl << endl;
cout << "遞迴 I:";
cin >> I;
cout << endl;
//遞迴 I
double MSA_Violation = 0;
double TDSP_Violation = 0;
check = false;
ifRead1 >> _Msapath.MSAAST;
ifRead1 >> _Msapath.MSAOrg;
ifRead1 >> _Msapath.MSADes;
while(!ifRead1.eof())
{
    ifRead1 >> _MSATrave.MSAPathflow;
    ifRead1 >> _MSATrave.MSANodenum;
    ifRead1 >> _MSATrave.MSANodesum;
    _MSATrave.MPath.clear(); //清掉上一條路徑資料
    _MSATDSP.MPath.clear();
    //讀取路徑
    for (USI uTmp = 0; uTmp < _MSATrave.MSANodenum; uTmp++)
    {
        int MPath=0;
        ifRead1 >> MPath;
        _MSATrave.MPath.push_back(MPath);
    }
    //找到相對應的 OD 總流量
    for(int a=0; a < _ODflowlist.size(); a++)
    { //OD 路徑總量
        if(_Msapath.MSAAST == _ODflowlist[a].ODAST &&
            _Msapath.MSAOrg == _ODflowlist[a].ODOrg &&
            _Msapath.MSADes == _ODflowlist[a].ODDes) {
            ODID = a;
            break;
        }
    }
    //OD 路徑總量
    // 如果資料時間不在範圍內,就跳過不讀,
    tInterval = ((int)(_Msapath.MSAAST / 180)) * 180;
    _Msapath.MSAAST = tInterval;
    for(int m = 0; m < _Tdspathlist.size(); m++)
    { // for m 依時性路徑資料
        if (_Msapath.MSAOrg != _Tdspathlist[m].TDSPOrg ||
            _Msapath.MSADes != _Tdspathlist[m].TDSPDes ||
            _Tdspathlist[m].TDSPAST != _Msapath.MSAAST)
        { continue; }
        TDSPID = m;
        //起迄點一樣且點數一樣且節點數總和相同,原路徑
        if( _MSATrave.MSANodenum == _Tdspathlist[m].TDSPNodenum &&
            _MSATrave.MSANodesum == _Tdspathlist[m].TDSPNodesum )
        {
            //原路徑資料計算 流量重新分配
            _MSATrave.MSA_NPathflow = ceil( ( I * double(_MSATrave.MSAPathflow) ) / (I+1) )
            + ( ( 1 * double(_ODflowlist[ODID].ODPFlow) ) / (I+1) );
            //違反值計算
            MSA_Violation = ( fabs( _MSATrave.MSA_NPathflow - _MSATrave.MSAPathflow ) )

```

```

                                                                    / ( _MSATrave.MSAPathflow ));
if (MSA_Violation >= 0.05)
{
    _MSATrave.ViolationSum = 1; }
else
{
    _MSATrave.ViolationSum = 0; }
    _MSATrave.MSAPrcentage=(double(_MSATrave.MSA_NPathflow)/
        double(_ODflowlist[ODID].ODPFlow));
        check = true;
        break;
    }
    else
    {
//有新路徑，原路徑資料計算，流量重新分配(有新路徑)
_MSATrave.MSA_NPathflow = ceil( ( I * double(_MSATrave.MSAPathflow) ) / (I+1) )
+ ( ( 1 * double(_Tdsppathlist[m].TDSPPathflow) ) / (I+1) );
//違反值計算
MSA_Violation = ( ( fabs( _MSATrave.MSA_NPathflow - _MSATrave.MSAPathflow ) )
        / ( _MSATrave.MSAPathflow ));
        if (MSA_Violation >= 0.05) { _MSATrave.ViolationSum = 1; }
        else { _MSATrave.ViolationSum = 0;}
//機率值
_MSATrave.MSAPrcentage= (double(_MSATrave.MSA_NPathflow) /
        double(_ODflowlist[ODID].ODPFlow));
        break;
    }
} // for m 依時性路徑資料
_Msapath.Newpathlist.push_back(_MSATrave);
//判斷下一筆資料是否存在
if( ifRead1.eof() )
{
if(check == false)
{
//TDSP 流量
_MSATDSP.MSA_NPathflow = ceil( ( I * double(_Tdsppathlist[TDSPID].TDSPPathflow) )
/ (I+1) ) + ( ( 1 * double(_ODflowlist[ODID].ODPFlow) ) / (I+1) );
//TDSP 節點數
_MSATDSP.MSANodenum = _Tdsppathlist[TDSPID].TDSPNodenum;
//TDSP 節點數和
_MSATDSP.MSANodesum = _Tdsppathlist[TDSPID].TDSPNodesum;
//TDSP 違反值計算
TDSP_Violation = ( ( fabs( _MSATDSP.MSA_NPathflow - _MSATrave.MSAPathflow ) )
        / ( _MSATrave.MSAPathflow ));
_MSATDSP.ViolationSum = 1;
for (USI n = 0; n < _Tdsppathlist[TDSPID].TDSPNodenum; n++)
{
    int MPath=0;
    MPath = _Tdsppathlist[TDSPID].TPath[n];
    _MSATDSP.MPath.push_back(MPath);
}
_MSATDSP.MSAPrcentage = double( _MSATDSP.MSA_NPathflow ) /
        double(_ODflowlist[ODID].ODPFlow );
//路徑資料丟回_MSATDSP struct
_Msapath.Newpathlist.push_back(_MSATDSP);
}
//統計路徑流量

```

```

int RouteFlow = 0; //所有路徑流量總值
int M = 0;
double P = 0;
for(int uF=0; uF < _Msapath.Newpathlist.size(); uF++)
{ RouteFlow += _Msapath.Newpathlist[uF].MSA_NPathflow; }
M = _ODflowlist[ODID].ODPFlow - RouteFlow;
if (M < 0) {
for (int uF=0; uF < _Msapath.Newpathlist.size(); uF++) {
if ( _Msapath.Newpathlist[uF].MSA_NPathflow == 0) continue;
else _Msapath.Newpathlist[uF].MSA_NPathflow--;
M++;
_Msapath.Newpathlist[uF].MSAPrcentage =
double(_Msapath.Newpathlist[uF].MSA_NPathflow) /
double(_ODflowlist[ODID].ODPFlow);
//_Msapath.Newpathlist[uF].MSAPrcentage++;
if (M == 0) break;
if (uF == _Msapath.Newpathlist.size() - 1) uF = -1; }
} else if (M > 0) {
for (int uF=0; uF < _Msapath.Newpathlist.size(); uF++) {
_Msapath.Newpathlist[uF].MSA_NPathflow++;
M--;
_Msapath.Newpathlist[uF].MSAPrcentage =
double(_Msapath.Newpathlist[uF].MSA_NPathflow) /
double(_ODflowlist[ODID].ODPFlow);
//_Msapath.Newpathlist[uF].MSAPrcentage++;
if (M == 0) break;
if (uF == _Msapath.Newpathlist.size() - 1) uF = -1;
}
}
//機率值重新計算
for(int up=1; up < _Msapath.Newpathlist.size(); up++)
{
_Msapath.Newpathlist[up].MSAPrcentage = _Msapath.Newpathlist[up].MSAPrcentage +
_Msapath.Newpathlist[up-1].MSAPrcentage; }
_Msapathlist.push_back(_Msapath); //起迄點資料丟回 struct
_Msapathlist[_Msapathlist.size()-1].MSAOrg << "\t" <<
_Msapathlist[_Msapathlist.size()-1].MSADes << "\t" << endl;
break;
}
else
{
//讀取下一筆 MSA 之 AOD 資料
ifRead1 >> tmpT;
ifRead1 >> tmpO;
ifRead1 >> tmpD;
//判斷第一筆與下一筆 ODT 是否是一樣的
if( tmpT != _Msapath.MSAAST || tmpO != _Msapath.MSAOrg || tmpD !=
_Msapath.MSADes )
{
//TDSP 路徑流量重新計算並傳回
if(check == false)
{
//TDSP 流量
_MSATDSP.MSA_NPathflow = ceil( ( I * double(_Tdspathlist[TDSPID].TDSPPathflow) )
/ (I+1) ) + ( ( 1 * double(_ODflowlist[ODID].ODPFlow) ) / (I+1) );
//_MSATDSP.MSA_NPathflow = _ODflowlist[ODID].ODPFlow - newflow;

```

```

//TDSP 節點數
_MSATDSP.MSAnodenum = _Tdsppathlist[TDSPID].TDSPNodenum;
//TDSP 節點數和
_MSATDSP.MSAnodesum = _Tdsppathlist[TDSPID].TDSPNodesum;
//TDSP 違反值計算
TDSP_Violation = ( ( fabs( _MSATDSP.MSA_NPathflow - _MSATrave.MSAPathflow ) )
                    / ( _MSATrave.MSAPathflow ) );
_MSATDSP.ViolationSum = 1;
for (USI n = 0; n < _Tdsppathlist[TDSPID].TDSPNodenum; n++)
{
    int MPath=0;
    MPath = _Tdsppathlist[TDSPID].TPath[n];
    _MSATDSP.MPath.push_back(MPath);
}
int aaa=0;
aaa= _ODflowlist[ODID].ODPFlow;
_MSATDSP.MSAPrcentage = double( _MSATDSP.MSA_NPathflow ) /
                        double(_ODflowlist[ODID].ODPFlow );
//路徑資料丟回 _MSATDSP struct
_Msapath.Newpathlist.push_back(_MSATDSP);
}
else { check = false; }
//統計路徑流量
int RouteFlow = 0; //同一 OD 對下總量
int M = 0;          //OD 總量與路徑和總量差值
double P = 0;
for(int uF=0; uF < _Msapath.Newpathlist.size(); uF++)
{ RouteFlow += _Msapath.Newpathlist[uF].MSA_NPathflow; }
M = _ODflowlist[ODID].ODPFlow - RouteFlow;
if (M < 0) {
    for (int uF=0; uF < _Msapath.Newpathlist.size(); uF++) {
        if (_Msapath.Newpathlist[uF].MSA_NPathflow == 0) continue;
        else _Msapath.Newpathlist[uF].MSA_NPathflow--;
        M++;
        _Msapath.Newpathlist[uF].MSAPrcentage =
            double(_Msapath.Newpathlist[uF].MSA_NPathflow) /
            double(_ODflowlist[ODID].ODPFlow);
        if (M == 0) break;
        if (uF == (_Msapath.Newpathlist.size() - 1)) uF = -1;
    } } else if (M > 0) {
    for (int uF=0; uF < _Msapath.Newpathlist.size(); uF++) {
        _Msapath.Newpathlist[uF].MSA_NPathflow++;
        M--;
        _Msapath.Newpathlist[uF].MSAPrcentage =
            double(_Msapath.Newpathlist[uF].MSA_NPathflow) /
            double(_ODflowlist[ODID].ODPFlow);
        if (M == 0) break;
        if (uF == (_Msapath.Newpathlist.size() - 1)) uF = -1;
    } }
}
//機率值重新計算
for(int up=1; up < _Msapath.Newpathlist.size(); up++) {
    _Msapath.Newpathlist[up].MSAPrcentage = _Msapath.Newpathlist[up].MSAPrcentage +
    _Msapath.Newpathlist[up-1].MSAPrcentage; }
_Msapathlist.push_back(_Msapath); //起迄點資料丟回 struct
_Msapath.Newpathlist.clear();

```

```

//上一筆 ODT 存到暫存變數(O,D,T)中
_Msapath.MSAAST = tmpT;
_Msapath.MSAOrg = tmpO;
_Msapath.MSADes = tmpD;
} } }
    ifRead1.close();
        cout << "MSA 演算法結束....." << endl;
        cout << endl;
        //存檔
fstream SPFileOut;
SPFileOut.open("../MSAOut//MSA.dat", ios::out);
int Totalviolation = 0;
int Counter=0;
/*****MSA 資料輸出開始*****/
for(int k =0; k < _Msapathlist.size(); k++)
{
    for (p = 0; p < _Msapathlist[k].Newpathlist.size(); p++)
    {
        if( _Msapathlist[k].Newpathlist[p].MSA_NPathflow != 0 )
        {
            Counter++;
            SPFileOut<< setw(5) << _Msapathlist[k].MSAAST
                << setw(5) << _Msapathlist[k].MSAOrg
                << setw(5) << _Msapathlist[k].MSADes
                << setw(5) << p
                << setw(5) << _Msapathlist[k].Newpathlist[p].MSA_NPathflow
                << setw(5) << _Msapathlist[k].Newpathlist[p].MSANodenum
                << setw(5) << _Msapathlist[k].Newpathlist[p].MSANodesum
                << setw(10) << setprecision(2) << _Msapathlist[k].Newpathlist[p].MSAPrcentage ;
            Totalviolation += _Msapathlist[k].Newpathlist[p].ViolationSum;
            for (int q =0; q < _Msapathlist[k].Newpathlist[p].MSANodenum; q++)
            {
                SPFileOut << setw(5) << _Msapathlist[k].Newpathlist[p].MPath[q];    }
                SPFileOut << endl;/"    P:" << p << endl;    }    }    }
            SPFileOut.close();
        }
    }
/*****MSA 資料輸出結束*****/
//違反值輸出存檔
fstream SVFileOut;
SVFileOut.open("../MSAOut//SV.txt", ios::out);
SVFileOut << I
    << setw(12) << Totalviolation
    << setw(12) << Counter
    << setw(12) << setprecision(4) << double(Totalviolation) / double(Counter);
SVFileOut.close();
return true;
}

```

檔案名稱：VehGen.h

```
#ifndef __DTA_VEHGEN_H__
#define __DTA_VEHGEN_H__
#include "../DTASim.h"
#include "../DTADEF.h"
#include <fstream>
#include <iomanip>
#include <vector>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <ctime>
using namespace std;
struct B_Path
{
    USI          BRoutenum;          //路徑編號
    USI          BPathflow;          //路徑流量
    USI          BNodenum;           //節點數
    USI          BNodesum;           //節點數和
    std::vector<unsigned int> BPath;  //路徑
    double       BRoutePrcentage;    //路徑百分比
};
typedef vector<B_Path> Bpathlist;
// 建立輸出 MSA 資料結構
struct A_Path
{
    USI          AAST;               //指派時段
    USI          AOrg;               //起點
    USI          ADes;               //迄點
    Bpathlist    NewBpathlist;
};
typedef vector<A_Path> Apathlist;
//建立模擬資料結構
struct CarPath
{
    USI          CarID;               //車輛編號
    USI          CarType;             //車輛屬性
    USI          CarBehavior;         // Behavior 行為
    USI          CarFamiliarity;      //熟悉度
    USI          CarAST;              //指派時段
    USI          CarOrg;              //起點
    USI          CarDes;              //迄點
    USI          CarPathflow;         //路徑流量
    USI          CarNodenum;          //節點數
    USI          CarNodesum;          //節點數和
    vector<unsigned int> CPath;       //路徑
    double       CarPrcentage;        //機率值
};
typedef vector<CarPath> CPathlist;
class CVehGen
{
public:
    bool VehGen();
private:
```




```
A_Path          _Apath;
Apathlist       _Apathlist;
B_Path          _Bpath;
//Bpathlist     _Bpathlist;
CarPath         _CarPath;
CPathlist       _CPathlist; };
#endif
```

檔案名稱：VehGen.cpp

```
#include "VehGen.h"
```

```
//Rand()產生 0~1 之間的亂數
```

```
double Rand() { return double(rand()) / RAND_MAX; }
```

```
bool CVehGen::VehGen()
```

```
{
    //依時間產生亂數
    srand((unsigned)time(NULL));
    cout << "隨機將車流量重新分配至 NewCarpath.....";
    // 讀取基本資料 (MSA data)
    ifstream ifRead1("./MSAOut/MSA.dat", ios::in);
    // 檢查是否開檔失敗？
    if(ifRead1.fail() != 0)
    {
        cout<< "Cannot open file " << "MSA.dat" << "!\n";
        return false;
    }
    int tmpO, tmpD;
    while(!ifRead1.eof())
    {
        //Read1
        ifRead1 >> _Apath.AAST
            >> _Apath.AOrg
            >> _Apath.ADes
            >> _Bpath.BRoutenum
            >> _Bpath.BPathflow
            >> _Bpath.BNodenum
            >> _Bpath.BNodesum
            >> _Bpath.BRoutePrcentage;
        _Apath.NewBpathlist.clear();
        _Bpath.BPath.clear();
        for (USI uTmp = 0; uTmp < _Bpath.BNodenum; uTmp++)
        {
            int BPath=0;
            ifRead1 >> BPath;
            _Bpath.BPath.push_back(BPath);
        }
        if (_Apathlist.size() <= 0) {
            tmpO = 0;
            tmpD = 0;
        } else {
            tmpO = _Apathlist[_Apathlist.size()-1].AOrg;
            tmpD = _Apathlist[_Apathlist.size()-1].ADes;
        }
        if (_Apath.AOrg == tmpO && _Apath.ADes == tmpD) {
            _Apathlist[_Apathlist.size()-1].NewBpathlist.push_back(_Bpath);
        } else {
            //B 部分路徑傳回 vector
            _Apath.NewBpathlist.push_back(_Bpath);
        }
    }
}
```

```

        _Apathlist.push_back(_Apath);
    }
} //Read1
ifRead1.close();
// 讀取基本資料 (Carpath data)
ifstream ifRead2("../MSAInput//Carpath.dat", ios::in);
// 檢查是否開檔失敗？
if(ifRead2.fail() != 0)
{
    cout<< "Cannot open file " << "Carpath.dat" << "!\n";
    return false;
}
while(!ifRead2.eof())
{ //Read2
    ifRead2 >> _CarPath.CarID
        >> _CarPath.CarType
        >> _CarPath.CarBehavior
        >> _CarPath.CarFamiliarity
        >> _CarPath.CarAST
        >> _CarPath.CarOrg
        >> _CarPath.CarDes
        >> _CarPath.CarPathflow
        >> _CarPath.CarNodenum
        >> _CarPath.CarNodesum;
    _CarPath.CPath.clear();
    for (USI uTm = 0; uTm < _CarPath.CarNodenum; uTm++)
    {
        int CPath=0;
        ifRead2 >> CPath;
        _CarPath.CPath.push_back(CPath);
    }
    int VehRouteID = 0;
    for(int m = 0; m < _Apathlist.size(); m++)
    { //for m
        if( _CarPath.CarAST >= _Apathlist[m].AAST &&
            _CarPath.CarAST < (_Apathlist[m].AAST+180)&&
            _CarPath.CarOrg == _Apathlist[m].AOrg &&
            _CarPath.CarDes == _Apathlist[m].ADes )
        {
            do
            {
                _CarPath.CarPrcentage = Rand();
                //判斷有多少路徑
                for(int r=0; r < _Apathlist[m].NewBpathlist.size(); r++)
                { //for r
                    //判斷各路徑機率值，決定路徑
                    if( _CarPath.CarPrcentage < _Apathlist[m].NewBpathlist[r].BRoutePrcentage)
                    {
                        VehRouteID = r;
                        break;
                    }
                } //for r
                //判斷路徑流量不為 0，才進行路徑更換
                if( _Apathlist[m].NewBpathlist[VehRouteID].BPathflow <= 0)
                { continue; }
                //路徑更換
            }
        }
    }
}

```

```

        _CarPath.CPath.clear();
for(int uK=0; uK < _Apathlist[m].NewBpathlist[VehRouteID].BNodenum; uK++)
{
    int CPath=0;
    CPath = _Apathlist[m].NewBpathlist[VehRouteID].BPath[uK];
    _CarPath.CPath.push_back(CPath);
}
_CarPath.CarNodenum = _Apathlist[m].NewBpathlist[VehRouteID].BNodenum;
_CarPath.CarNodesum = _Apathlist[m].NewBpathlist[VehRouteID].BNodesum;
//路徑流量減 1
_Apathlist[m].NewBpathlist[VehRouteID].BPathflow--;
break;
} while(true);
break;
}
else
{ continue; } }
//將 Car 路徑資料放置 vector
_CPathlist.push_back(_CarPath);
} //Read2
ifRead2.close();
cout << "Ok!";
cout << endl;
////////NewCarpath 資料輸出開始////////
fstream NCPFileOut;
NCPFileOut.open("../MSAOut/NewCarpath.dat", ios::out);
for(int p =0; p < _CPathlist.size(); p++)
{
    NCPFileOut << setw(6) << _CPathlist[p].CarID
    << setw(6) << _CPathlist[p].CarType
    << setw(6) << _CPathlist[p].CarBehavior
    << setw(6) << _CPathlist[p].CarFamiliarity
    << setw(8) << _CPathlist[p].CarAST
    << setw(6) << _CPathlist[p].CarOrg
    << setw(6) << _CPathlist[p].CarDes
    << setw(6) << _CPathlist[p].CarPathflow
    << setw(6) << _CPathlist[p].CarNodenum
    << setw(6) << _CPathlist[p].CarNodesum;
    for (int q =0; q < _CPathlist[p].CarNodenum; q++)
    {
        NCPFileOut << setw(5) << _CPathlist[p].CPath[q];
    }
    NCPFileOut << endl;/"    P:" << p << endl;
}
NCPFileOut.close();
////////NewCarpath 資料輸出結束////////
return true;
}

```