2023 年臺灣國際科學展覽會
優勝作品專輯

作品編號　**190013**

參展科別　電腦科學與資訊工程

作品名稱　以最佳化演算法進行鐵路時刻表排點

得獎獎項

就讀學校　臺北市立建國高級中學

指導教師　蔣宗哲、王鼎中

作者姓名　吳亞倫

關鍵詞　鐵路時刻表、最佳化演算法、登山演算法

作者簡介

大家好，我是建國中學數理資優班二年級的吳亞倫。我從小便對鐵路有著濃厚的興趣，時常會去各地搭乘與拍攝不同的火車。高一時有機會進入建中數資班就讀，也接觸到了程式設計，便一頭栽進了寫程式的領域，目前擔任建中電子計算機研習社的學術長。自從開始做專題研究之後，我便希望能將兩個興趣結合，做出一個可以有所實際應用的研究。很高興能夠參與這一次的國際科展，希望能讓我得到許多新的經驗。

另外，我要在此感謝蔣宗哲教授和王鼎中老師一路上對我的指導，他們的研究經驗給予我巨大的幫助，使我可以完成這項研究。我對他們的教導和關愛表示最誠摯的感謝。也要感謝一路上支持與鼓勵我的同學和家人，還有特教組老師、台鐵局與交通部同仁對於我各方面的協助。

壹、 摘要

鐵路時刻表排點直到目前為止仍十分仰賴人工作業，且排班優劣對於乘客服務品質有顯著的影響。本研究採用啟發式最佳化演算法以及模擬器進行旅客列車鐵路時刻表排點，希望能夠找出一份針對旅客需求，能夠提升旅客運輸成功率且降低旅途時間的時刻表。我們提出一種班表編碼機制，可依此機制產生班表草稿。我們研發的模擬器可將班表草稿轉換為合法無衝突之班表。最後，透過登山演算法來搜尋班表草稿，並以模擬器評估班表優劣，我們實現了一個自動化排班系統。實驗結果指出我們的模擬器能夠有效率地產生無衝突之班表，且所提出之演算法操作有助於提升運輸成功率和降低旅途時間。

Abstract

Up to now, railway timetabling still relies heavily on manual work, but the quality of its scheduling significantly impacts the service quality felt by passengers. This study uses a heuristic optimization algorithm and simulator to schedule passenger railway timetables, hoping to find a timetable that can improve the success rate of passengers and reduce travel time. We propose a timetable encoding method to generate a “draft timetable,” and our simulator can turn the “draft timetable” into conflict-free timetables. At last, by searching for the “draft timetable” through Hill climbing algorithms and evaluating the quality of the timetable with simulators, we implemented an automated timetabling system. The experimental results indicate that our simulator can efficiently generate conflict-free timetables, and the proposed algorithm operation helps to improve the passenger success rate and reduce travel time.

貳、 前言

一、 研究動機

鐵路時刻表排點一向都是一份曠日費時以及需要大量決策性思考的工作。有鑒於所有鐵路列車之運行全部需按照鐵路時刻表進行，時刻表的安排優劣與否即代表了鐵路系統的路線使用效率。由於時刻表關係到行車安全、系統收益、旅客滿意度甚至是政策導向，因此往往只能透過大量的人力手動進行排點。但是，隨著鐵路系統複雜化以及班次密度增加，人工排點便逐漸無法負荷。人工排點以經驗作為進行班次安排與衝突排除的基準，意味著排點時無法將太多細節納入考量，也難以預測時刻表的優劣度；所需的時間極長更是其致命傷。據臺鐵調度員表示，目前每一次改點，從安排新班次、化解衝突到與乘務、機務人員的排班協調至少需要六個月以上的作業時間，若我們可以自動化解決前半部分，可以減少非常多人工作業與時間。

另外，在我們使用鐵路通勤時，由於現有的列車停靠站安排與車輛運用並非完全按照乘客之需求與喜好進行，因此常常會遇到客滿或是等不到車的狀況。這些狀況大致可以分為有車但是客滿、只有慢車、或著是完全沒有列車可以讓我們在所希望的時間抵達目的地。而這些狀況的產生不但會增加乘客的旅運時間，更會因此降低乘客搭乘意願。

因此，我希望可以研究出一種自動安排列車停靠站模式的方法，能夠針對乘客等車以及搭乘時間進行最佳化，並且排出一個相對應不會產生列車衝突的時刻表。

二、 研究目的

（一） 透過搜尋演算法尋找一個能夠最佳化旅客需求，並且最小化旅客乘車時間之鐵路時刻表

（二） 建立一個模擬系統，能夠在短時間內排解時刻表當中的衝突，並且依照乘客資料評估一時刻表的優劣度

（三） 使用臺鐵真實的路線資料以及旅客數據產生最佳化之時刻表

三、 問題定義

本研究所要探討的問題如下：在給定乘客歷史資料、線路與車站、各型列車基本資料的情況下，產生包含**列車數量**、各列車**停靠站規則**與**首站抵達時間**的時刻表。並且透過搜尋演算法找出轉為正式且滿足**運轉安全限制**的鐵路時刻表之後，能夠**滿足載客成功率門檻**並**最小化乘客總旅程**時間的班表。以下將針對上述所提到的進行詳細說明。

（一） 本研究給定的資料

1. 乘客歷史搭乘資料（OD 表）：

旅客起訖資料 (Origin-destination (OD) information) 是一種在進行交通相關決策常常用到的資料。OD 表描述在一個時段當中，有多少的乘客從某車站進入系統，以及他們的旅途終點。透過分析此資料可以得知在此鐵路系統當中人流的偏好移動方式，以進行時刻表的安排。在本研究中，我們取得臺鐵透過票證資料統計出的 2019 年每日分時 OD 表[1]，並計劃利用此歷史資料作為計算各站間旅運需求的準則。

欄位名稱	欄位型別	欄位說明
TripDate	DATE	旅次日期
TripHour	INTEGER	旅次時段(Ex: 12表示為12時00分至12時59分)
TicketClass	STRING	票證分類 (N-IC: 非電子票證; IC: 電子票證)
IDType	STRING	電子票證卡種
HolderType	STRING	持卡身分
OriginStationID	STRING	起站代碼
OriginStationName	STRING	起站中文名稱
DestinationStationID	STRING	迄站代碼
DestinationStationName	STRING	迄站中文名稱
Volume	INTEGER	旅運量

圖 1：交通數據匯流平臺 臺鐵每日各站分時 OD 資料(O) 資料項目

[1] 本資料來源為交通部管理資訊中心。本研究透過公文向交通部臺灣鐵路管理局運務處請求准予以研究名義使用，並且利用交通部數據匯流平台下載。

2. 線路、車站資料，此類資料包含以下各項：

(1) 車站基本資料與順序，例如：站名、車站編號、上下行的隔站編號[2]
(2) 車站可供旅客列車使用之股道數量
(3) 各站標準停車時間[3]
(4) 各站間各車種標準行車時間[3]
(5) 進站最低間隔時間

3. 各型列車基本資料[3]，包含：

(1) 列車數量限制。
(2) 各型的列車只有此數量的列車可供運用，排點時所使用的列車數不可超過此限制。
(3) 列車最大載客量。

（二） 最佳化目標

在本研究旨在針對乘客的需求進行最佳化。因此，所進行的最佳化目標皆以乘客作為本體。而我們所訂定的最佳化目標有下列兩項：

1. 最大化乘客成功率 SuccessRate

我們認為每一位乘客在選擇進入鐵道系統時，會期望能夠找到一個乘車方式，使他能在特定時間之前抵達他的目的地，否則他會離開鐵路系統，轉而使用其他交通工具。另外，若某班次客滿而使得乘客無法上車，乘客也會無法抵達目的地。而所謂的成功率，指的便是所有乘客能夠透過本系統抵達目的地的比例。

2. 最小化乘客平均旅運時間

對於所有成功抵達目的地的旅客，我們希望我們的班表可以盡可能減少其候車與乘車時間。而我們定義每位乘客的旅途時間定義為（出站時間） – （進站時間），而我們會對此數值的平均進行最小化。

[2] 此資料透過政府資料開放平臺取得。（資料集 ID：33425）
[3] 向臺鐵局運務處綜合調度所取得。

（三） 運轉限制

在列車進行運轉時，必須要遵守特定的運轉規則，包括遵守閉塞區間、軌道容量限制與維持最小時隔等等。這些限制會增加排點的困難度。而若排出來的時刻表不遵守這些限制，我們稱之為此時刻表內有衝突發生，必須進行排解。詳細的衝突定義與請看文獻探討之列車衝突，而排解方式則請看第一模擬器 --- 排點中相關部分。

（四） 班表草稿

由於真實時刻表上面待決定的資料過多，如果我們最佳化演算法直接搜尋時刻表，則解空間會過大，反而不利於求解。因此在本研究當中，我們會對列車時刻表進行編碼，即為此處所提的班表草稿。在進行搜尋時，最佳化演算法會先產生出此班表草稿，其中包含多班次的列車。每列車分別記載其首末站、中途個站是否停靠、以及首站預計的發車時間。由於透過上述資料，我們有辦法透過模擬器進行「解碼」，產生出一組真正的時刻表。

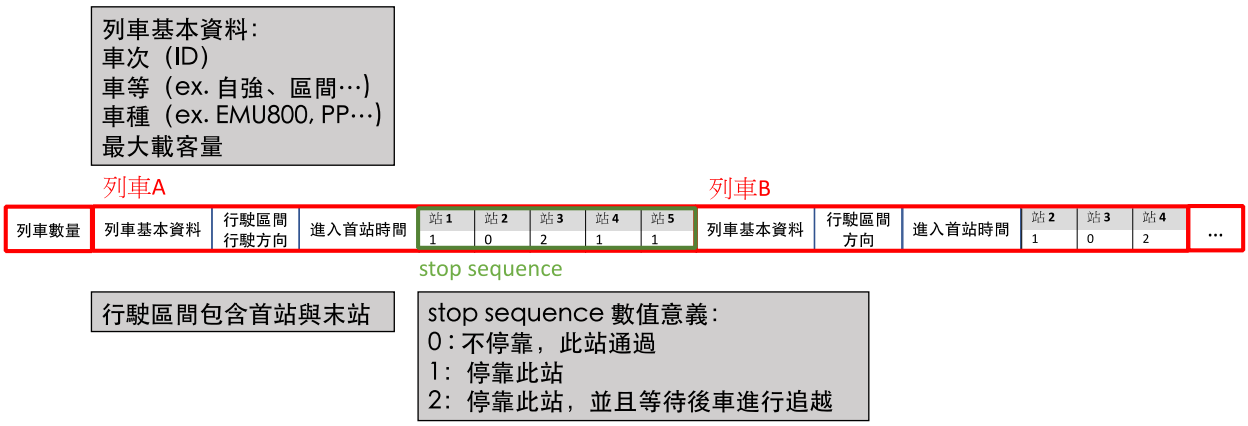
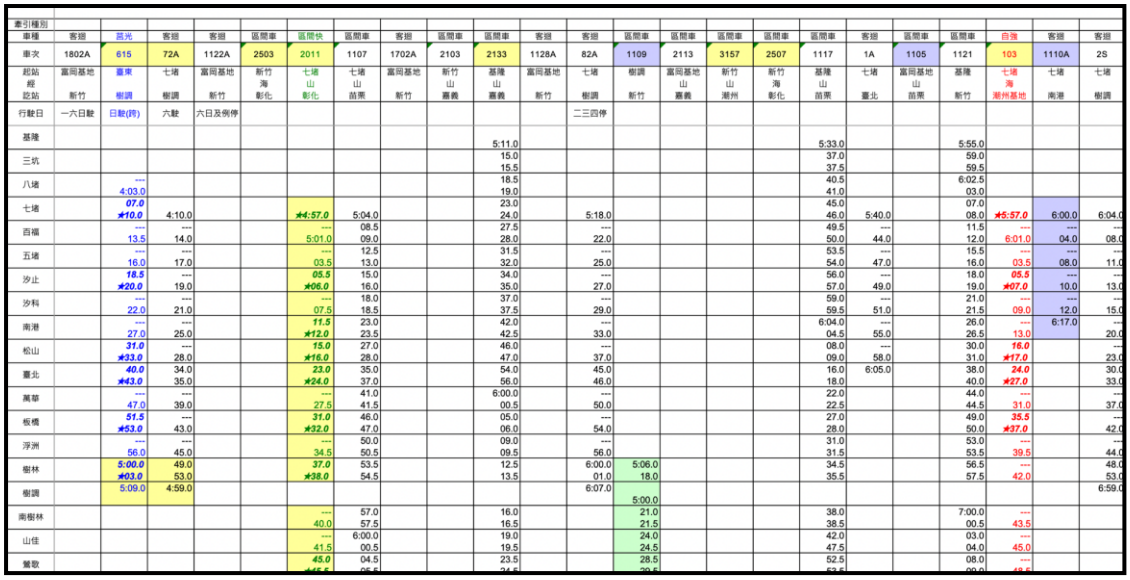


圖 2：「班表草稿」編碼方式

四、 文獻探討

（一） 鐵路時刻表簡介

在一個現代鐵路運輸系統當中，時刻表即是一切行車的依據。舉凡發車、停靠、通過等運行事件，或是股道、月台的分配，全部都必須按照時刻表來運行。如此一來，才有辦法避免掉列車衝突。



車種	客迴	莒光	客迴	客迴	區間車	區間快	區間車	客迴	區間車	區間車	客迴	客迴	區間車	區間車	區間車	區間車	區間車	客迴	區間車	區間車	自強	客迴	客迴
車次	1802A	615	72A	1122A	2503	2011	1107	1702A	2103	2133	1128A	82A	1109	2113	3157	2507	1117	1A	1105	1121	103	1110A	2S
起站	富岡基地	臺東	七堵	富岡基地	新竹	七堵	七堵	富岡基地	新竹	基隆	富岡基地	七堵	樹調	富岡基地	新竹	新竹	基隆	七堵	富岡基地	基隆	七堵	七堵	七堵
經					海	山	山		山	山				山	山	海	山		山		海		
站	新竹	樹調	樹調	新竹	彰化	彰化	苗栗	新竹	嘉義	嘉義	新竹	樹調	新竹	嘉義	潮州	彰化	苗栗	臺北	苗栗	新竹	潮州基地	南港	樹調
行駛日	一六日駛	日駛(同)	六駛	六日及例停								二三四停											
基隆										5:11.0							5:33.0			5:55.0			
三坑										15.0							37.0			59.0			
										15.5							37.5			59.5			
八堵		---								18.5							40.5			6:02.5			
		4:03.0								19.0							41.0			03.0			
七堵		07.0								23.0							45.0			07.0			
		★10.0	4:10.0			★4:57.0	5:04.0			24.0		5:18.0					46.0	5:40.0		08.0	★5:57.0	6:00.0	6:04.0
百福		---	---			---	08.5			27.5		---					49.5	---		11.5	---	---	---
		13.5	14.0			5:01.0	09.0			28.0		22.0					50.0	44.0		12.0	6:01.0	04.0	08.0
五堵		---	---			---	12.5			31.5		---					53.5	---		15.5	---	---	---
		16.0	17.0			03.5	13.0			32.0		25.0					54.0	47.0		16.0	03.5	08.0	11.0
汐止		18.5	---			05.5	15.0			34.0		---					56.0	---		18.0	05.5	---	---
		★20.0	19.0			★06.0	16.0			35.0		27.0					57.0	49.0		19.0	★07.0	10.0	13.0
汐科		---	---			---	18.0			37.0		---					59.0	---		21.0	---	---	---
		22.0	21.0			07.5	18.5			37.5		29.0					59.5	51.0		21.5	09.0	12.0	15.0
南港		---	---			11.5	23.0			42.0		---					6:04.0	---		26.0	---	6:17.0	---
		27.0	25.0			★12.0	23.5			42.5		33.0					04.5	55.0		26.5	13.0		20.0
松山		31.0	---			15.0	27.0			46.0		---					08.0	---		30.0	16.0		---
		★33.0	28.0			★16.0	28.0			47.0		37.0					09.0	58.0		31.0	★17.0		23.0
臺北		40.0	34.0			23.0	35.0			54.0		45.0					16.0	6:05.0		38.0	24.0		30.0
		★43.0	35.0			★24.0	37.0			56.0		46.0					18.0			40.0	★27.0		33.0
萬華		---	---			---	41.0			6:00.0		---					22.0			44.0	---		---
		47.0	39.0			27.5	41.5			00.5		50.0					22.5			44.5	31.0		37.0
板橋		51.5	---			31.0	46.0			05.0		---					27.0			49.0	35.5		---
		★53.0	43.0			★32.0	47.0			06.0		54.0					28.0			50.0	★37.0		42.0
浮洲		---	---			---	50.0			09.0		---					31.0			53.0	---		---
		56.0	45.0			34.5	50.5			09.5		56.0					31.5			53.5	39.5		44.0
樹林		5:00.0	49.0			37.0	53.5			12.5		6:00.0	5:06.0				34.5			56.5	---		48.0
		★03.0	53.0			★38.0	54.5			13.5		01.0	18.0				35.5			57.5	42.0		53.0
樹調		5:09.0	4:59.0									6:07.0											6:59.0
													5:00.0										
南樹林						---	57.0			16.0			21.0				38.0			7:00.0	---		
						40.0	57.5			16.5			21.5				38.5			00.5	43.5		
山佳						---	6:00.0			19.0			24.0				42.0			03.0	---		
						41.5	00.5			19.5			24.5				47.5			04.0	45.0		
鶯歌						45.0	04.5			23.5			28.5				52.5			08.0	---		

圖 3：鐵路時刻表（檔案來源：臺鐵提供）

圖 3 為一份臺鐵旅客列車時刻表的截圖。可以看到在圖中每一直行為一班列車，行最上方有可以得知列車基本資訊，如車等及班次。而在其下方各站的位置則依序為各站的進入與出站時間。而通過之列車則只標示出站時間（同進站時間）。

以上的時刻表有著旅客以及方便司機員查看的特性，但是，在進行時刻表的安排或是車輛的行控調度時，上方的時刻表呈現方式並沒有辦法方便的協助我們識別出衝突，因此鐵路公司會使用另一種時刻表表現方式，也就是運行圖。

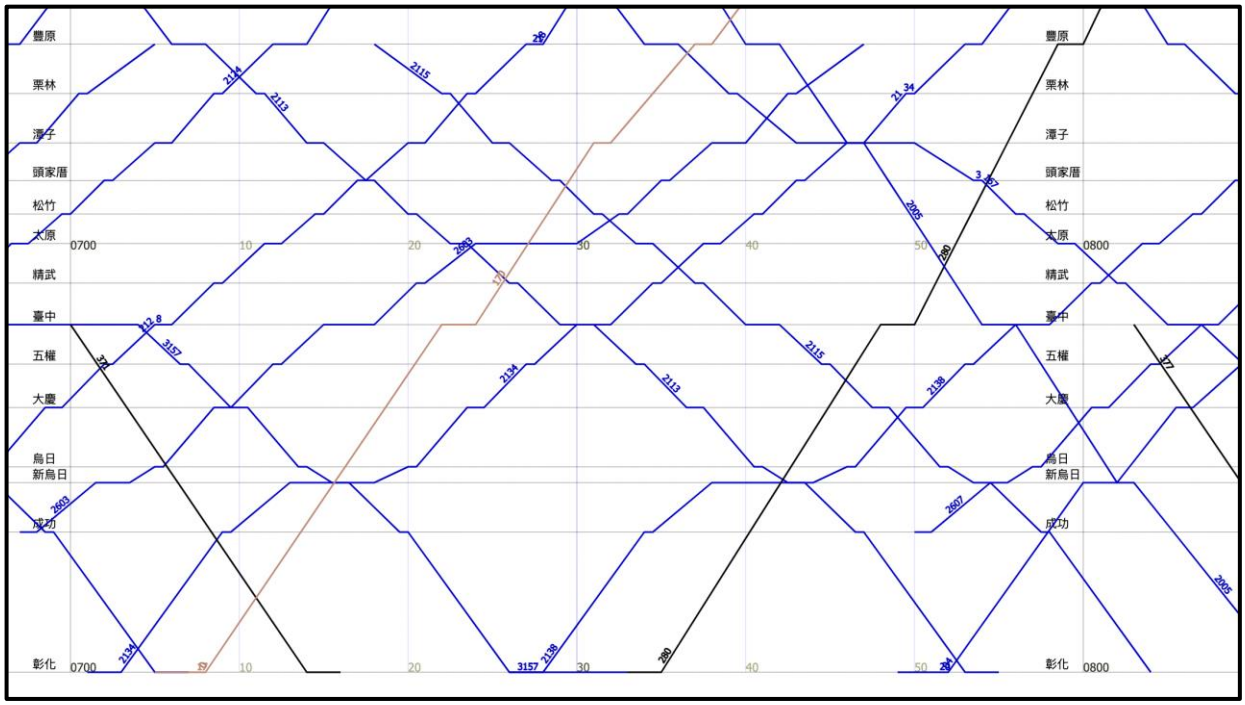


圖 4：運行圖（擷取自 台灣鐵路運行圖網站）

圖 4 為一張經典的運行圖，圖中的縱軸代表地理位置，橫軸代表時間，而圖中的每條線個代表一台列車。從此圖中，我們可以很方便地看出每輛列車在每個時刻會出現在何站之間。而因為運行圖的線條交叉即代表列車有追越或是會車，因此，運行圖可以方便作業人員識別出班表中潛在的衝突。

由於運行圖具有訊息密度大以及能夠明顯的識別出衝突等優點，因此一般來說，在以人工進行排點時，會以運行圖作為主要的輔助工具。不過對於電腦來說圖像並非最好的運算方式，因此在本研究當中，運行圖只會作為最後產出的結果，以及作為此報告書當中對於衝突狀況的描述。

（二） 排點問題文獻探討

時刻表排點問題（Train Timetabling/Scheduling Problem, 以下簡稱 TTP），旨在安排出一個可行的時刻表，使得列車在運行的時候可以滿足一些特定營運限制，像是軌道容量等等，使得實際列車在運行時不會發生衝突。此類問題常常會因為針對不同的目標進行最佳化排點，而有不同的變種。根據 Caprara 等人的研究 (2007) 表示，鐵路排點的

相關問題可以依照時間尺度以及決策單位分為多個不同的子問題，其相關順序如下圖所示。而在本研究中，我們則主要著手在 Train Timetabling 的部分。

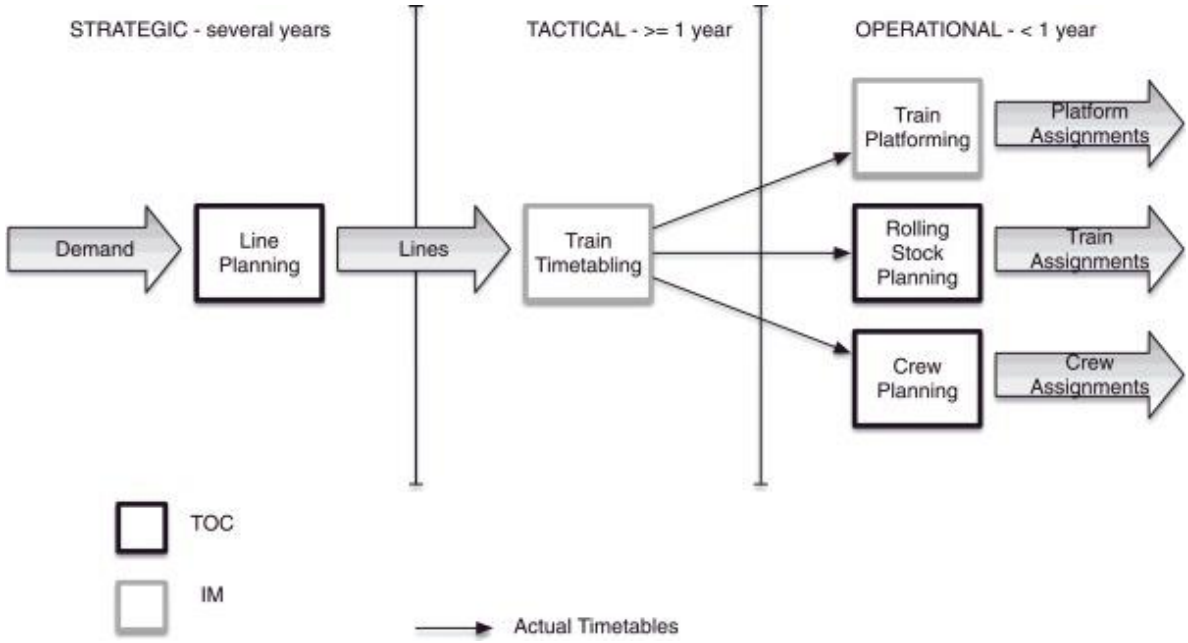


圖 5：鐵路營運相關規劃問題（Robenek, 2016）

大多數的研究在處理 TTP 問題時，是以 Mixed Integer Linear Programming (MILP) 或是 Integer Linear Programming (ILP) 方式解決。早期目標函數多半為營運方希望最大化利潤（Caprara et al. , 2002），近期則也出現以乘客角度進行研究的文章（Niu, 2015）。同時也有研究嘗試以啟發式演算法中的基因演算法試圖進行排點（Tormos, 2008）。臺灣方面，交通部運輸研究所也多次進行過相關的研究。

而在本研究當中，我們主要希望能夠以滿足旅客的角度進行每班列車停靠站與首站發車時間的安排，並且根據所設計出來的停靠站規則安排出一份可行的時刻表。我們會先透過最佳化演算法來產生班表草稿，並且將此班表草稿放入模擬器當中，透過模擬方式化解衝突並產生最後的時刻表，然後放入乘客到模擬器當中去以計算旅客的搭車成功率與搭乘、等候時間，再將此結果回饋到最佳化演算法當中來進行以進行下一步的搜尋。透過重複以上步驟多次來得到一個最佳化之後的班表。

（三） 列車衝突

以上內容中有提到，我們時刻表安排時必須要可以避免衝突。而這邊將針對列車衝突進行介紹。鐵路運輸時為了避免列車相撞，因此與路面交通一樣皆需要號誌系統的輔助。但在同一段鐵軌上面，無論是追撞或是對撞，皆不會有閃避的空間，也就是說，我們必須要想辦法確保一段鐵軌區間當中只能有一輛列車，其餘列車皆不可進入此區間，以免發生對撞。而此段區間則稱為一「閉塞區間」。圖 6 為閉塞區間的示意圖。在此號誌系統下，進入前車所在的閉塞區間的號誌皆會顯示為紅燈，代表必須要停車，而再往前則會顯示黃燈，告訴駕駛需要將速度調降到 60km/h。

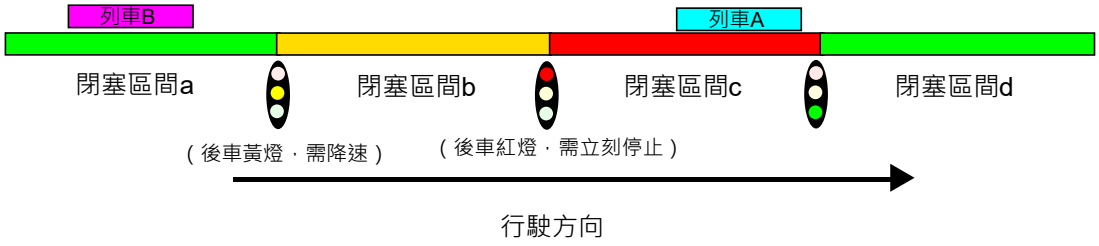


圖 6：閉塞區間示意圖（圖中色塊是以 B 車視角所繪製）

我們在排點時，其中一項限制就是要讓列車能夠在遵守此號誌的情況下運行。但是，閉塞區間若以號誌機作為排點基準，會使系統過於複雜，無法有效的進行排點。在現行的人工排點方式當中，我們會事先計算出車站間的行駛時間與某些特定的間隔時間（Headway Time），並且以此為基準進行排點。

這邊定義排點會遇到的衝突如下：[4]

1. 無法進站

假設有一列車 T 想要進入車站 S，無論列車 T 是否有要停靠此車站，S 當下狀況必須要符合下列條件才可以使列車T進站

(1) 車站S必須至少有一空股道K可以使列車 T 進行通過或是停靠。

[4] 本段落的所有列車運行限制及衝突簡介，皆是在 2022 年 8 月初，臺鐵局運務處綜合調度所計劃組（目前臺鐵負責進行排點之單位）的調度員同仁與研究人員進行訪談時，由對方所說明的臺鐵現行排點時的相關定義。

(2) 且若當前時刻為 t_c，空股道K上一列車離站時刻為 t_l，則 t_c 與 t_l間必須要有一最低時隔 (headway time) HW_{plat}，即 $t_c - t_l \geq HW_{plat}$須成立。

(3) 對於當前時刻 t_c 與同向之上一進站列車進站之時刻 t_b，也必須要滿足一最低時隔 HW_{STA}。

根據向調度員詢問，得知臺鐵 HW_{STA} 目前各車站皆以 3 分鐘為準，而 HW_{plat} 會與股道配置方式以及前一班車之停靠方向有關。一般來說，同向的月台會設定為 4 分鐘，而若該列車要停靠一反向之月台，則必須要設為 5 分鐘。

2. 站間被卡住

定義數組 $\left(ts_i^A, te_i^B\right)$ 為列車 i 從 A 站發車到 B 站進站的時間對，換句話說，為佔據軌道$\overrightarrow{AB}$的起訖時間。現在有相鄰兩站：起站O 、訖站D。設列車i, j為相鄰兩車，在正常情況（不發生追撞或站間追越）下，若$ts_i^O < ts_j^O$（i 較 j 早離開站O），$te_i^D < te_j^D$ 應成立。反之，若 $ts_i^O < ts_j^O$ 但是 $te_i^D > te_j^D$ ，則稱之為列車 i 卡到列車 j 。其運行圖如下所示：

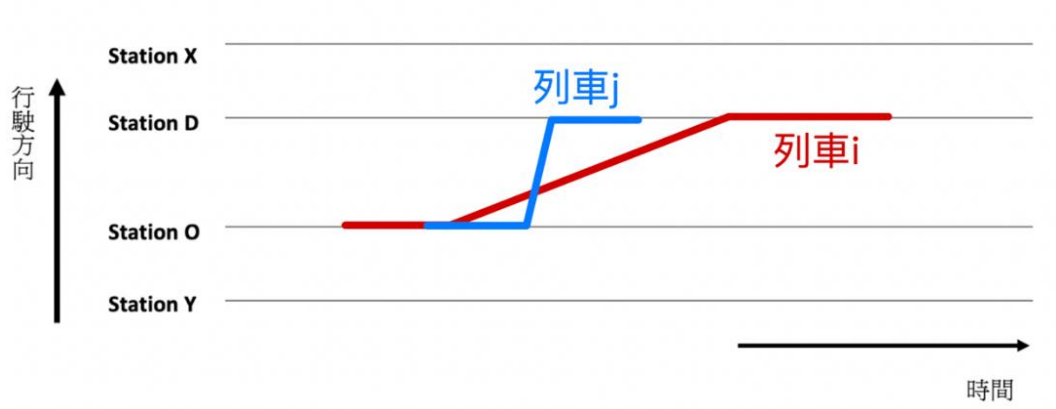
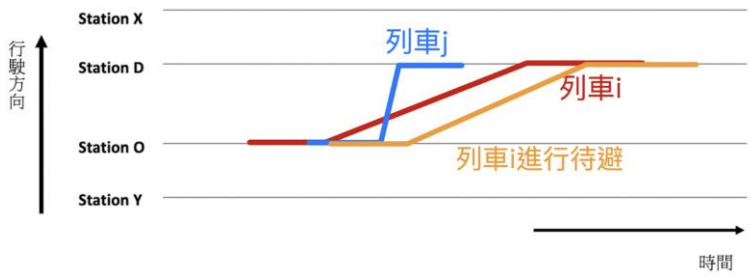


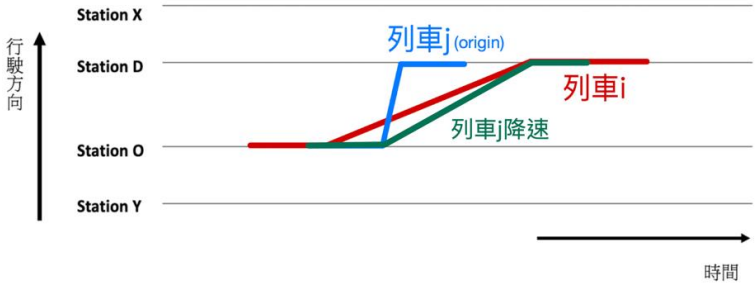
圖 7：i 卡到 j 的運行圖

可以發覺的是，對於以上兩種衝突，若要進行化解，勢必有至少一方必須要犧牲其行車時間使得另一方得以通過。我們將可行的三種做法整理如下：



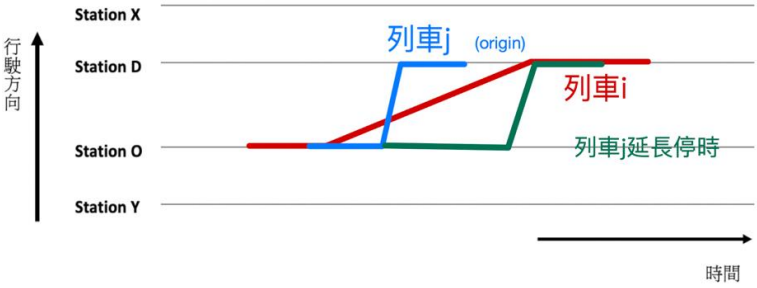
1. i 待避：

延後ts_i^O使得$ts_i^O > ts_j^O$，即讓列車i等到列車j 發車後再發車。此情況稱為列車i 在站O 進行待避。



2. j 降速：

列車j出站時間ts_j^O不變，但是延後te_j^D，即延長列車j在$\overrightarrow{\mathrm{OD}}$的行駛時間，此稱之為列車$j$降速。



3. j 延長停時：

列車j在$\overrightarrow{\mathrm{OD}}$的行駛時間不變，同步延後$\left(ts_j^O, te_j^D\right)$，即代表延長了列車$j$在車站停靠時間，稱之為$j$在$O$延長停時。

其中我們可以看到，對於i、j 兩台列車，以待避的方式會使前車（i車）行駛時間延長，反之，降速以及延長停時會造成後車 （j車）行駛時間延長。且由於這幾種衝突化解模式的抉擇會影響列車離開站D的順序，進而影響其他站是否會發生衝突，因此，此部分的抉擇十分重要。而本研究的處理方式將於後續段落進行說明。

參、 研究過程與方法

一、 研究流程圖

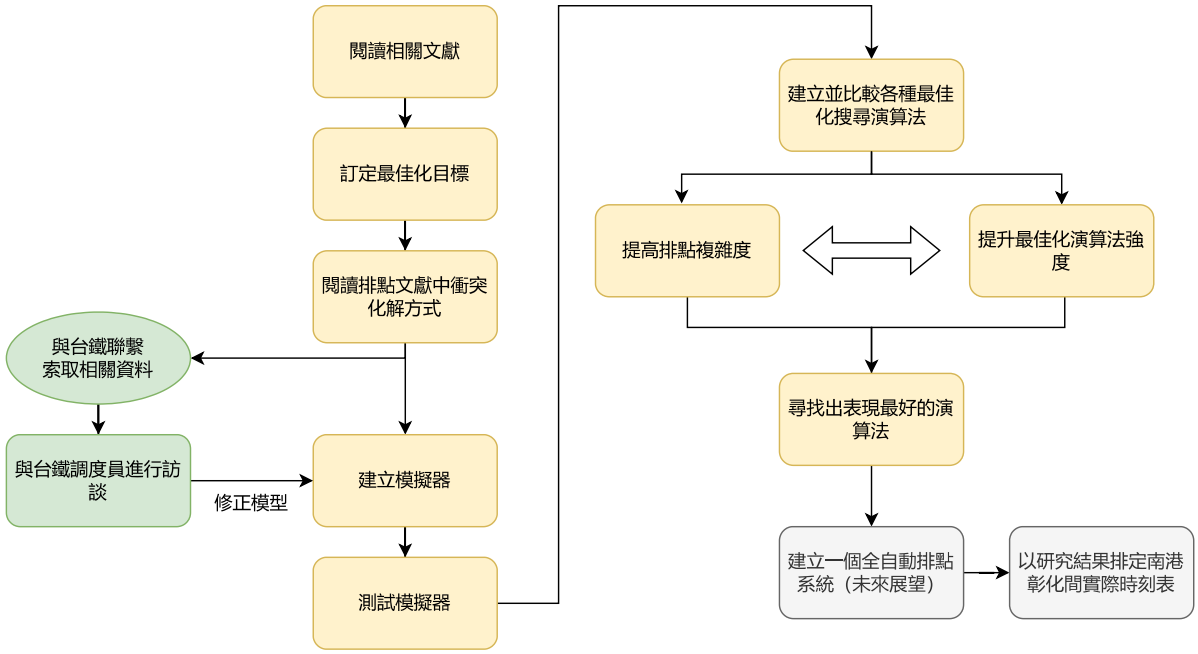


圖 8：研究流程圖

二、 研究架構與求解流程

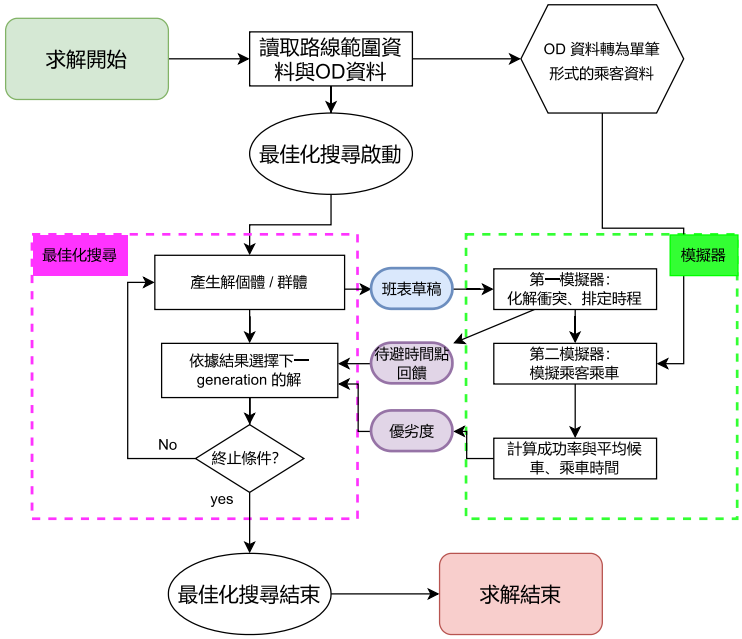


圖 9：求解流程圖

本研究的演算法大約可以分為兩個部分（圖 9）。第一部分為搜尋演算法，而第二部分為排點模擬器。我們透過搜尋演算法生出多組不同的「班表草稿」，當啟發式演算法生出一個班表草稿時，我們會將班表草稿傳入模擬器當中，以產生真正的班表以及計算出班表的優劣度。

而模擬器主要也可以分為兩大步驟。第一步驟為透過所得的班表草稿模擬出一個真實且沒有衝突的時刻表。第二步驟則透過我們生出的時刻表，配合 OD 資料，模擬在此時刻表下乘客的移動以及他們所花費的總時間，作為班表優劣度依據。

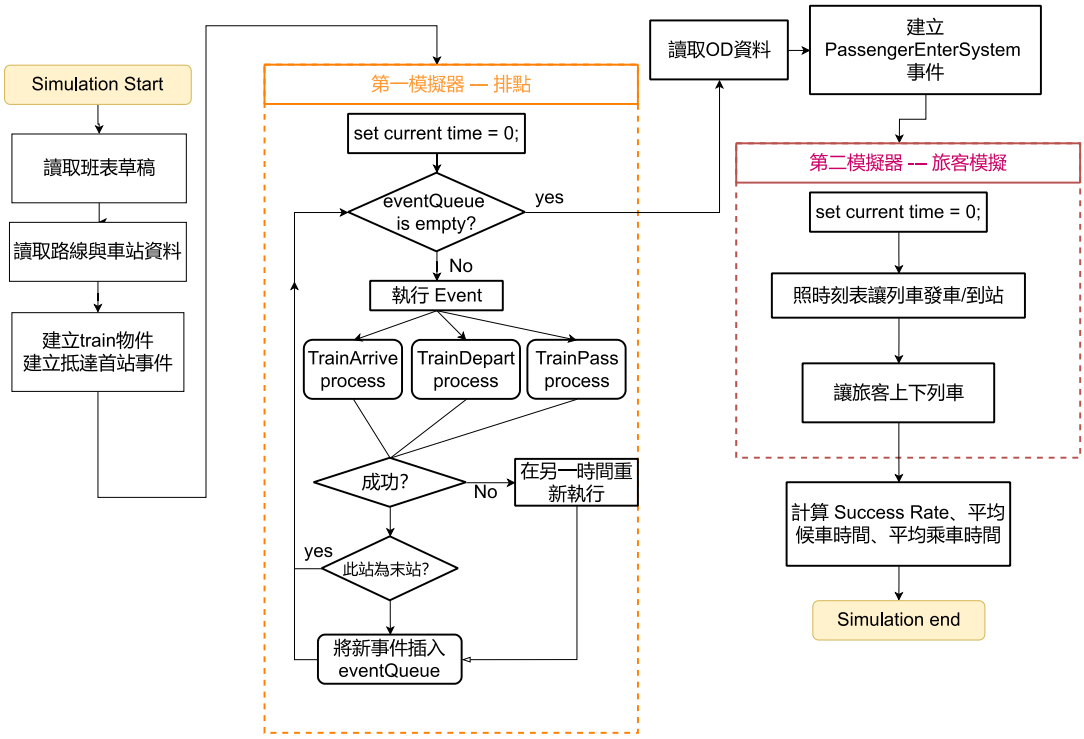


圖 10：模擬器流程圖

三、 第一模擬器 --- 排點

在本研究當中，模擬器為安排班表與評估解優劣的其中一個重要部分。而在我們的第一模擬器旨在透過模擬列車的運行，藉此化解列車衝突事件的發生，調整班表草稿使其**符合運轉限制**。

在本研究中模擬器是使用 Discrete-event simulation (DES) 的方式進行模擬，亦即將列車的發車、到站、通過由連續的動作轉為離散的一個個的事件 (Event)，並且按照時間順序依序執行各個事件。在程式實作的部分我們以物件導向的方式，定義多種事件物件（Event Object），包括①列車抵達事件（TrainArrive）、②列車發車事件（TrainDepart）、③列車通過事件（TrainPass）與

④乘客進入系統事件（PassengerEnterSystem，僅第二模擬器中使用到）。在模擬過程中我們會以一個照時間序排序集合「EventQueue」來維護所有的 Event Object。

於模擬過程當中，我們一開始先假設班表草稿當中的所有列車皆可以按照最佳時分運行，即在車站間皆以全速運行，使得運行時間恰等於各車種的標準行車時間，且各站的停靠時間皆為各站標準停車時間。在最佳時分的運行狀態下，我們可以保證此列車上的乘客的乘車時間一定為最小。

我們以此假設為前提，依序按照首站抵達時間為每輛列車建立第一個 TrainArrive 事件物件，並且按照時間順序執行 EventQueue 中的事件。在每次執行完成後再依序情況新增下一個 Event Object。但若以最佳時分運行時會發生衝突，則會有相對應的事件處理方式。

以下將解釋各種不同事件的執行方式：

（一） 列車抵達事件（TrainArrive）

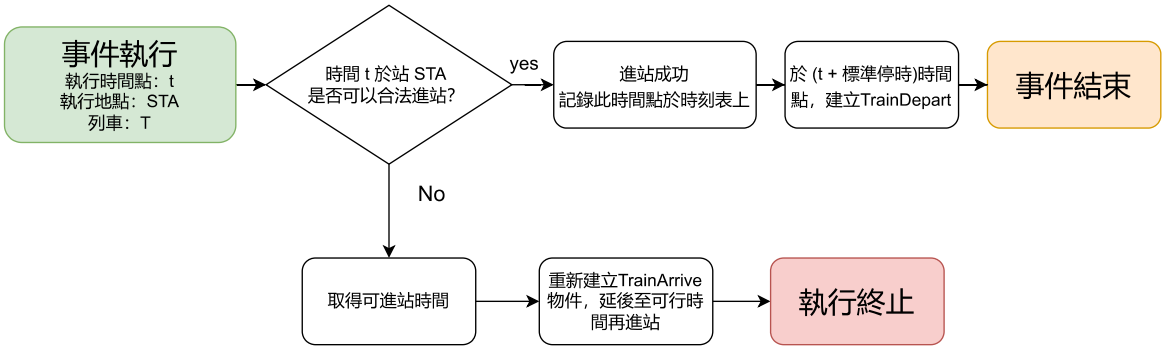


圖 11：：TrainArrive 執行流程圖

TrainArrive 事件表示一台列車 T 想要於時刻 t 進入車站 STA。模擬此事件旨在於化解掉**無法進站**的衝突。

當此事件被執行時，模擬器會先去檢查列車是否可以進站，包含 STA 是否有空月台以及滿足最小間隔時間等上述所列的條件限制。若此衝突發生，則模擬器會找出最近的可行時間，並且重新於該時間建立一新的 TrainArrive 事件，代表列車延後至該時間進站。此操作亦意味著 T 從上站往這站的行駛時間拉長，速度降低。此外，在執行 TrainArrive 時，我們會確保最先抵達STA的列車先進入可用的月台。

而若 T 可以不會發生無法進站的衝突，則我們會記錄此時間至時刻表上，然後於建立一個於標準停車時間點後執行的 TrainDepart 事件。

（二） 列車發車事件（TrainDepart）

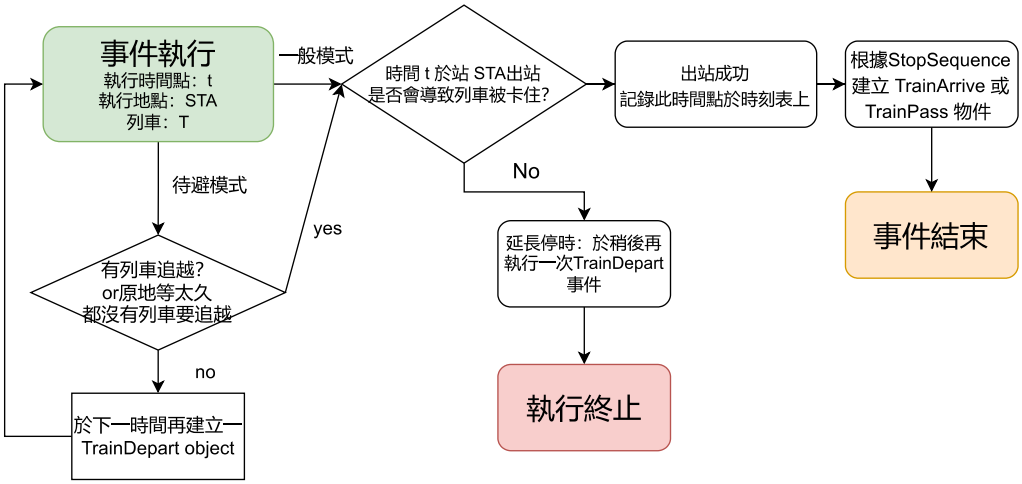


圖 12：TrainDepart 執行流程圖

列車發車事件指的是一台列車要從一個車站發車。此事件旨在化解從此站發車到下一站之間**列車被卡住**的衝突。

在本研究中，TrainDepart 事件有不同的模式 — ①一般模式 及 ②待避模式。一般模式表示此列車在標準停靠時間一過之後，會盡量越早發車越好，不會刻意等待列車追越。而待避模式則會刻意在月台繼續等待，直到後一班車在此站追越該車為止。而是否爲待避模式是透過最佳化演算法在班表草稿中決定的。

在 TrainDepart 當中，當出站時發覺以最佳時分運行會導致後續路段出現被卡住的現象，我們會讓其進行延長停時[5]。唯當被卡住列車之等級較前車高時，模擬器會進行紀錄，並讓後續最佳化演算法決定前車是否要切換停靠模式至待避模式。

[5] 此作法為臺鐵局調度員表示之現有作法。

（三） 列車通過事件（TrainPass）

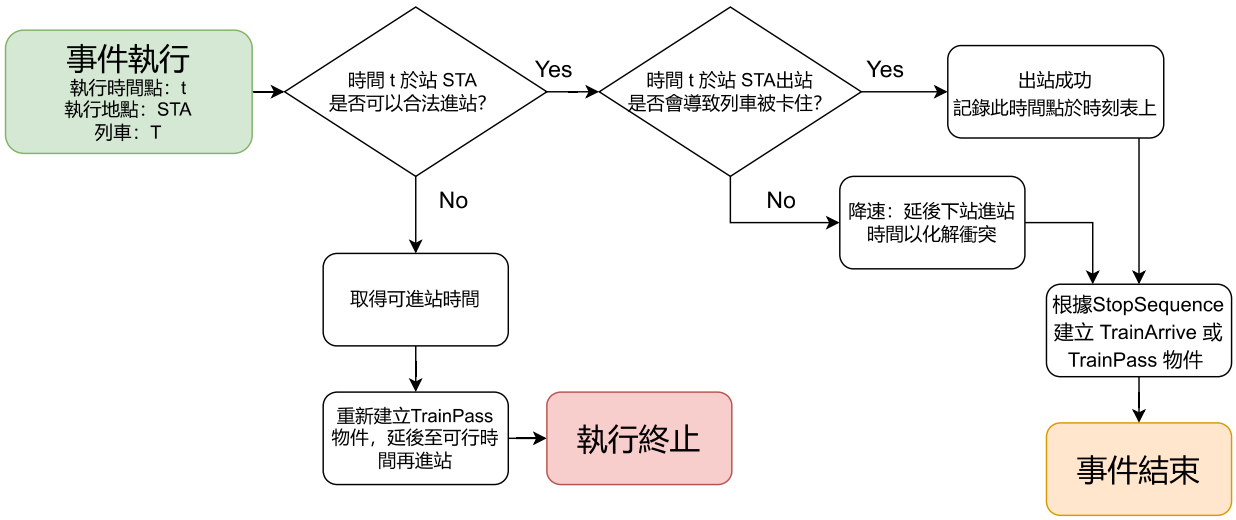


圖 13：TrainPass 執行流程圖

列車通過事件代表一列車不停靠該站，將直接通過。此事件進站時衝突處理方式與 TrainArrive 大致相同，但出站處因為該列車並不會停靠，因此**無法如同 TrainDepart 一樣直接延長停時**，必須另外以特殊方式處理。

在臺鐵局現行的排點方式當中，如果通過列車中途被卡住，會直接延長上一個停靠站的停靠時間，以調整使得兩相鄰的車站間的路段列車皆得以全速運行。但是在程式當中此作法會**導致模擬器要不斷回朔**，且一處調整完過後可能造成其他處必須重新安排，導致新的衝突發生，因此並非完全可行的做法。

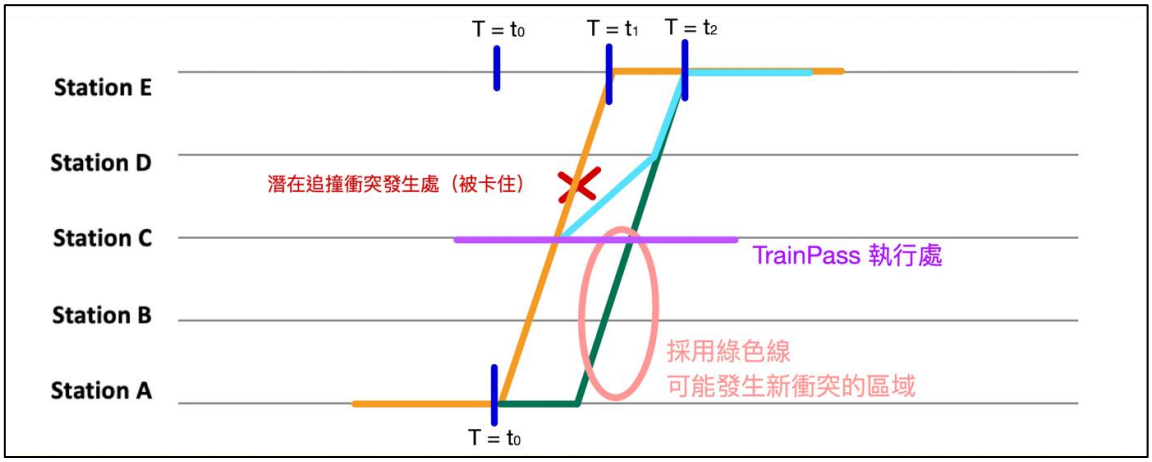


圖 14：通過事件衝突解決方式抉擇

（橘線：原始路徑；綠線：臺鐵局作法/延長停時；淺藍線：本研究中的折衷解決方案）

如圖 14 中所表示，若該列車 A 站的下一停靠站為 E，而在 CD 站間會被卡住。若以人工排點的作法會把模擬器執行處返回到 A 站、$T = t_0$之處。但此作法仍有可能於粉色區域中發生新的衝突，如此連鎖反應下會使的模擬器無法運行。但因為本研究主要最佳化目標為乘客乘車時間，因此若選擇直接在站 C 以降速方式處理（淺藍線），則可以在對於乘客來說，AE 間運行時間與綠線相同的情況下，以有效率的方式處理衝突。

四、 第二模擬器 --- 旅客模擬

第二模擬器旨在透過模擬乘客的移動方式來求出乘客的旅途時間與成功率。在本研究中，我們假設一個乘客(群)會在一個時間點內進入其起站 O，並且希望成功在$t_{deadline}$之前抵達其目標車站 D。在乘客進入系統之後，其會從時刻表中等待一最快抵達目的地，且轉乘k次以內的搭車方式。而我們假設乘客都是自私且不會顧慮其他乘客的 (Kroon, Maróti & Nielsen, 2015)，因此乘客不會因為某班車可能擠不上而是先改搭其他列車。

在前面的問題定義中有提到，在本研究當中我們是使用 OD 資料作為評估乘客需求與運量的準則。但是 OD 資料的時間是以小時作為單位，且其人數可能到上百人，並無法直接將此資料放入模擬器。因此，我們會先將 OD 資料轉成供模擬器使用之格式。其中，我們會將乘客隨機切分成 5 人以內的小團體，並且隨機設定其可以接受的轉乘次數k，並且於 OD 資料中所記錄的該小時內選擇隨機選擇一時間作為該乘客進入系統的時間。另外，由於 OD 資料中並不包含$t_{deadline}$，因此我們會按照乘客的搭乘距離為其設定一個合理的$t_{deadline}$值。

第二模擬器也是由第一模擬器中的各種 event object 串起來。不同的是在 TrainArrive 時，我們進行的動作是讓列車中的乘客下車，而 TrainDepart 時，則是讓月台上的乘客上車。要注意的是，如果列車已經客滿，則乘客便無法上車，必須要重新規劃一路徑。若無其他路徑可以讓乘客在$t_{deadline}$之前抵達目的地，則乘客便會離開系統，放棄搭乘，使此解的成功率下降。

另外在乘客搜尋最短路徑時，我們採用了圖論演算法中廣為人知的 Dijkstra's 演算法(Dijkstra , 1959)進行優化。我們將每一個可行的事件建為圖上的點，而一個車站中的眾多不同時間的進站、出站事件則構成一個點集，並且建造兩種邊。第一種為連接同站相鄰兩事件的邊，此類邊的邊權為 0。第二種邊則代表一組〔上車 (depart), 下車 (arrive)〕配對。此類邊的邊權為 1。如此一來，在此圖上的最短路徑的邊權若為y，則代表乘客可以在搭乘y輛車（轉乘$y-1$次）以內抵達。

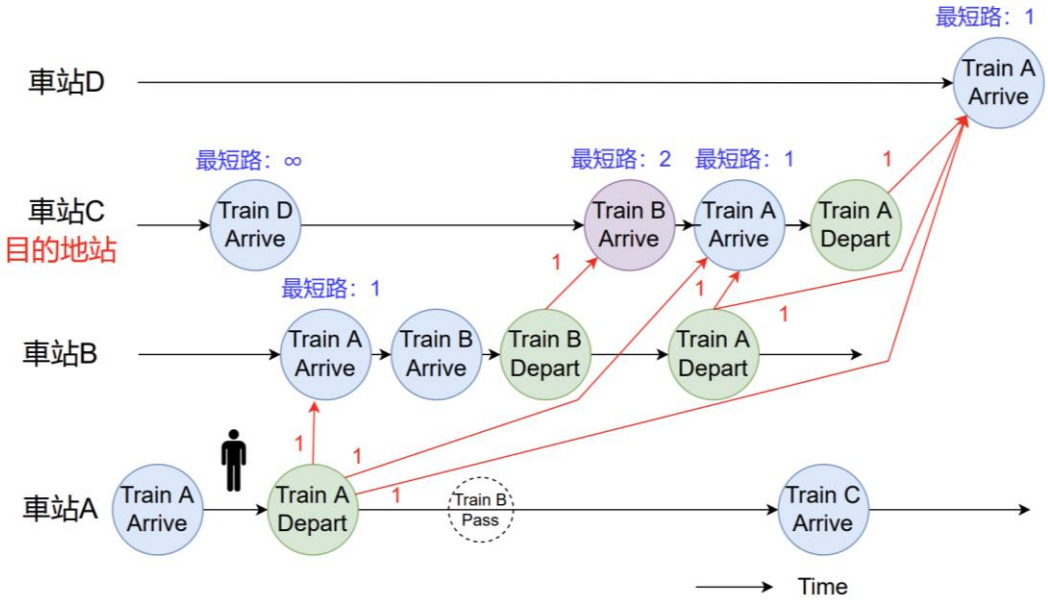


圖 15：搜尋最短路徑建圖方式

（箭頭代表有向邊，其中紅色邊的邊權為 1，黑色邊邊權為 0。點/事件已按照時間順序由左往右排列）

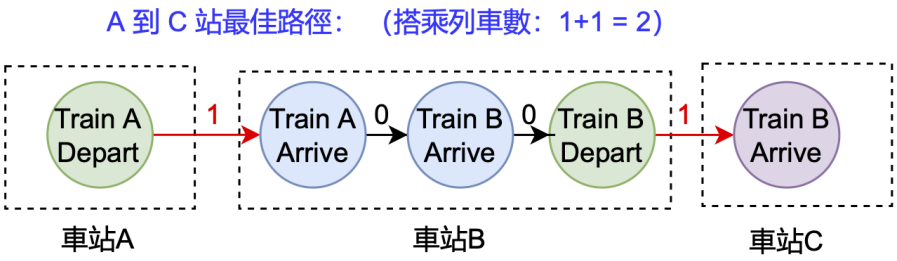


圖 16：按照圖 15 建圖後，從站 A 到站 C 乘客選擇的路徑

如此一來，乘客在決定路徑時邊十分容易。假設乘客在t_s時要從 A 站出發往 B 站，則我只要求 A 站點集當中時間≥ t_s的第一個點，到 B 站點集中所有時間介於$(t_s, t_{deadline})$之間，第一個邊權 $\leq k$ 的最短路徑，就會是此乘客的最佳搭乘路徑。

五、 最佳化搜尋演算法

本研究中旨在透過最佳化演算法以找出一份能夠最佳化成功率以及旅途時間的時刻表。能夠處理此類問題的演算法包含基因演算法、禁忌搜尋法、模擬退火法等等。而在本研究中，我們選擇以登山演算法以及其相關優化來進行最佳化搜尋。

（一） 登山演算法（Hill climbing algorithm）

Hill climbing 為最佳化演算法當中一種基於貪婪的作法，是眾多搜尋演算法當中十分基礎的一項。在多目標最佳化問題當中，我們可以將每個解對應到的優劣程度視為一個空間，稱之為搜尋空間，而此空間中每一點都會對應到一高度，即為此解優劣度。從圖 17 我們可以看到在一個解空間當中，會有所謂區域最佳解（local optima）與全域最佳解（global optima），而 Hill Climbing 演算法主要就可以用來搜尋區域最佳解。

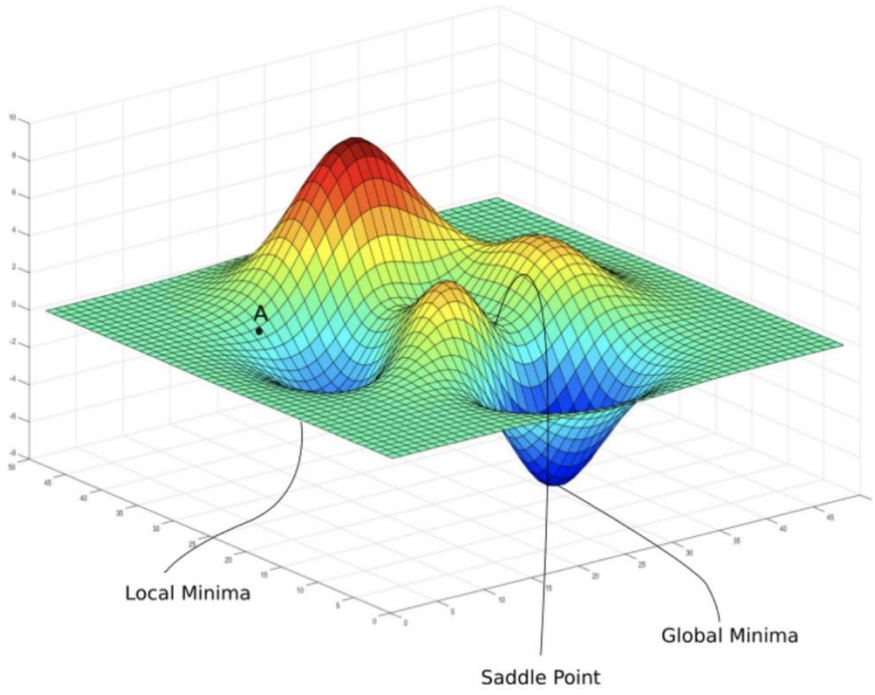


圖 17：搜尋空間示意圖（source: Yadav）

Hill Climbing 演算法的想法十分淺顯易懂。假設我們有一個多目標最佳化問題，其解優劣度可以用一函數$f(x,y)$來表示，而我們希望找到一組(x,y)使得$f(x,y)$ 有最小值，即求 $\min_{\forall x,y} f(x,y)$。在 Hill climbing 的一開始我們會先隨機選擇空間中的一點$u(x_0,y_0)$作為起點。在每一次的迭代當中，我們會對此解進行擾動，產生一個鄰近的新點，並且檢查此點的函數值是否較原本的小。若

有能找到一點 $v(x,y) \mid f(v) < f(u)$，則移動往該處，繼續搜尋。如此的過程像是我們在登山時不停往鄰近較低處前進一樣，函數值會非嚴格單調遞減，最後能夠找到一區域最低點，故稱之為登山法（Hill Climbing）。

圖 18：hill climbing 流程圖

在 Hill Climbing 演算法當中，影響是否容易找到好的解的因素在於我們所給予的擾動，若擾動過小，則演算法難以離開搜尋空間較小的低窪處，而非較好的解。而若擾動過大，則此演算法為與隨機搜尋無異，無法有效率的進行搜尋。以下圖為例，擾動過小會使演算法錯過較佳的解，但若有適中大小的擾動則可以使找到更好的解。

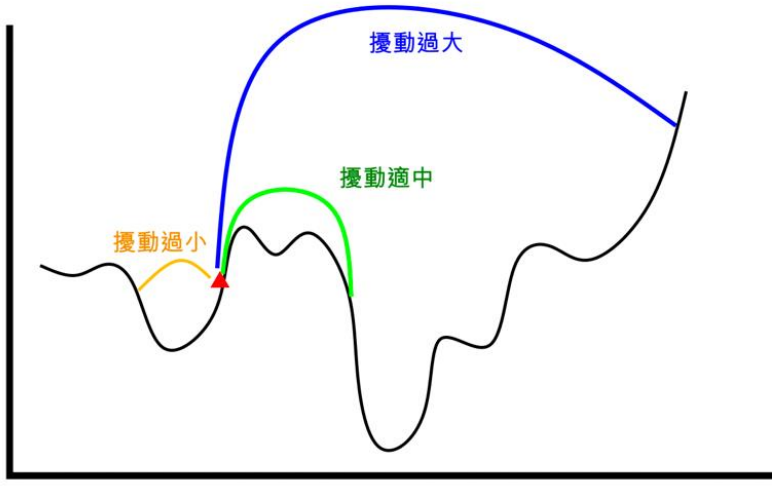


圖 19：擾動大小對求解之影響示意圖

Hill climbing 的結束條件通常是我們事先決定好一個足夠的迭代次數，做完該次數即結束。但是此作法當迭代次數過多時，會導致浪費運算時間與資源，而太少時則會無法確保已經找到最佳解。因此有一種作法稱之為 multi start 的優化方式，即為當演算法連續多次皆無法找到周圍有較低點時，我們會判斷現在所處位置已為低點當中，因此主動停下演算法，並且重新搜尋一次。如此一來我們便可以在相同的時間內進行多次搜尋，可以提升搜尋效率。

登山法的優勢在於它的實作簡單，而且可以有效率地找到區域最佳解，因此在本研究當中我們將利用他作為我們較小規模排點測試當中的演算法。但其缺點也十分明顯，如果我們過程當中所給予的擾動不當，或是初始解太爛，此演算法便很容易陷入不夠好的區域最小解當中。因此我以下我們將以下列不同方式想辦法對其優化。

（二） 利用登山演算法求解

在本研究中當中，我們會利用 Hill Climbing 來搜尋班表草稿。由於登山演算法大致可以分為「初始化」、「擾動」與「判斷是否接受」等三大部分，因此以下將對一一進行說明。

1. 初始化

在求解一開始時，我們先隨機產生擁有固定列車數量的一組班表草稿，其中的每一輛列車的停靠站、起始時間皆是隨機產生的，並且用其進行擾動與搜尋。之所以採用隨機的方式是因為在登山法的過程當中，如果初始解極為相似的話，產生出來的結果也會相似，無法幫我們有效找到最佳解。

2. 擾動

我們採用的擾動包含以下所列出：

(1) 將一列車起始站或結尾站向前/向後移動一站。（執行機率：20%）

(2) 更改一列車首站的時間。（執行機率：30%）

(3) 調整一列車各站的停靠站規律。（執行機率：30%）

(4) 更改一列車的行駛方向。（執行機率：10%）

(5) 將一列車整個重新更換為一隨機出來之列車：此做法是為了使擾動的程度可以更大。（執行機率：10%）

另外，我們希望可以比較是否允許更改列車數量對於此最佳化問題造成顯著的影響。因此，我們多設計了一種擾動方式如下：

(6) 增加或減少一列車（增：5%，減：10%）：

若當前班表中列車數量介於給定的最多與最少列車數範圍內，在一定的機率下，演算法會對此時刻表進行增/減車。在減車時我們會從所有列車當中找出載客數最少的一班列車，並

且將其刪除。此擾動有助於使班表更貼合乘客需求，並且刪除不合適之列車；但是過度增開列車可能會使路線受到更多佔用使衝突變多，以及提升營運成本；太快刪除列車則可能使所求的解在尚未演化到可載客之前即遭到刪除。因此，我們將進行刪除列車的機率調整至只有 10%。

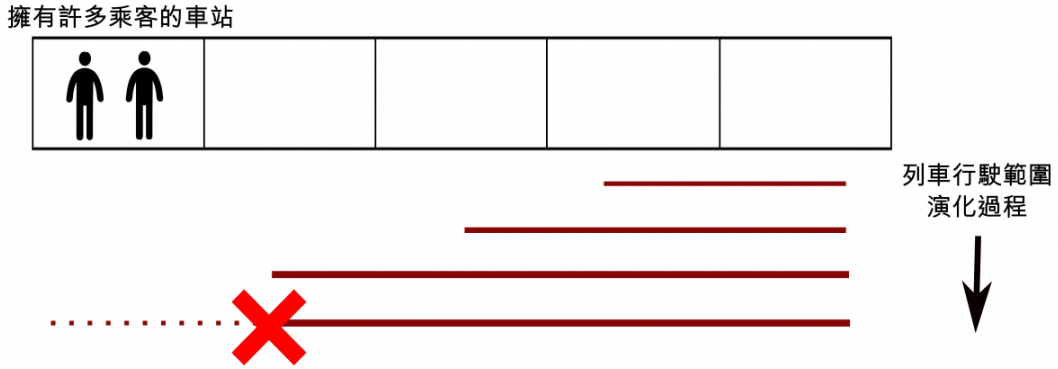


圖 20：過早刪除列車

如圖所示，過早可刪除列車可能使列車在演化至乘客可以搭乘前便已經被刪除。

每一次的迭代當中，我們都會選擇一輛列車進行，從(1)到(5)等五項之間隨機選擇一項進行擾動，並且以新產生的班表草稿放入模擬器當中進行排點。而(6)的擾動放入於否則將於稍後實驗中作為討論項目。

3. 解的優劣評估

本研究的最佳化目標包含最大化成功率以及最小化乘客旅途時間。但是由於我們有兩個目標同時在進行，如果直接採用其中一項做優化，勢必會犧牲掉另外一項。因此，本研究採用了一種特殊的解決方式。

對於每一組乘客，我們會決定一個 Success Rate 的我們所允許的限度值Θ_{SR}，如果當前的成功率已經大於Θ_{SR}，則代表我們的成功率已經夠大了，不需要再對他做最佳化，此時便只要對時間做最佳化。

Θ_{SR}值的具體用法如下：假設有兩個不同的解a, b，其成功率與旅途時間分別是SR_a, SR_b和t_a, t_b。則我們會以下列方式比較其優劣度：

$$\begin{cases} IF\ min\left(SR_a, \Theta_{SR}\right) == min\left(SR_b, \Theta_{SR}\right) : compare\ t_a, t_b \\ ELSE : compare\ SR_a, SR_b \end{cases}$$

此比較方式的意義在於當$SR_a = SR_b$或 $SR_a > \Theta_{SR}$&&$SR_b > \Theta_{SR}$時，我們便只比較旅途時間，否則便只比較成功率。這個做法在可以讓我們搜尋時會變成先優化成功率至足夠大之後，在滿足$SR_a > \Theta_{SR}$的情況下改為優化時間，以協助我們對多目標進行最佳化。

但是，由於登山法有其本質上容易陷入區域低點的限制，只有上述步驟可以預期其搜尋效果有限，因此我們決定再設計更多的優化方式。

（三） 初始解以 OD 資料優化

由於初始解的選擇往往會影響後續的搜尋過程很大，因此，我們希望能有一種做法使我們初始解直接較貼近乘客的分佈情形。因此我們設計了這種方式去修改原本的初始解。

在 Hill Climbing 開始之前，我們先將 OD 資料讀入，並且分析每一站的多寡。而之後我們會針對人數的分佈情形給每個車站一個機率值，作為決定停靠站規律的參考。此值越大則代表初始解當中選擇停靠此站的機率越大。

我們認為此做法將有利於在搜尋過程的一開始便找到成功率較高的解。

（四） Threshold-Accepting 選擇接受機制

Threshold-Accepting 是用來解決登山法容易陷入區域最佳解的問題所提出的優化。此演算法最早在 1990 年由 Dueck 等人提出，其作法有些類似於模擬退火演算法。

在原本的登山法當中，我們只允許解一路往低谷下降。但是，有時候我們會發現全域的最佳解只與當前所卡住的位置以一山之隔。如下圖所示，當我們位於紅色三角形處，如果此時演算法允許我們往上爬一點點，便有機會找到更好的解。

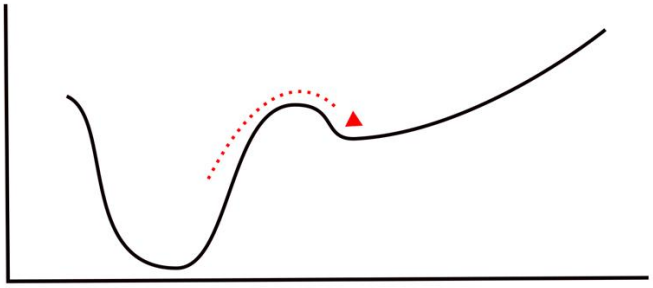


圖 21

而 Threshold-Accepting 就是在允許解偶爾可以變差一點點，以增加我們找到更低點的機會。在進行操作時會有一個門檻值φ，代表目前的解相較於最好的解可以變差多少。換句話說，如果v代表我要最小化的目標，而$v_{cur} - v_{best} < \varphi$，則演算法會接受$v_{cur}$作為下一次迭代的基準。

另外，門檻值φ會隨著迭代次數的增加線性的下降，到了一定次數之後便不允許解變差，以利我們收斂到最佳解。以本研究中為例，我們一開始將φ_{SR}設為 0.1，φ_{time}設定為 100 個單位時間。而在進行迭代的過程中，以總共 3000 次迭代為例，我們設定 Threshold-Accepting 機制會在第 500 次迭代之後啟動，而φ值會開始下降，直到第 2500 次迭代時φ值會降為 0，代表此時已經不接受稍微變差的解了。另外，由於本實驗的雙最佳化目標，因此在進行 accepting 時和原本評估解的優劣時相似，一樣會先比較成功率再比較旅途時間。

肆、 研究結果與討論

一、 研究設備與器材

（一） 程式語言：C++

C++ 在使用上有著速度優勢。由於最佳化的過程當中需要重複執行模擬器多次，因此程式執行速度十分重要。另外，對於其語法的熟悉程度以及 C++ 擁有好用的 STL 模板庫都是選擇其作為主要語言的原因。

（二） 輔助套件：

- nlohmann json：一個在 C++ 當中處理 JSON 檔案的開源函式庫。
- spdlog：一個進行運行紀錄輸出（logging）的開源函式庫。

（三） 運行環境：

1. 電腦：MacBook Air (M1, 2020)
2. C++執行環境：
    (1). Xcode (version 13.4.1)
    (2). Apple clang (version 13.1.6) (clang-1316.0.21.2.5)

二、 列車模擬結果

在本實驗當中，我們將臺鐵現行的時刻表轉換為班表草稿，並且嘗試使用本研究所建立的模擬器重新進行排點，並且放入乘客資料計算臺鐵目前班表的載客成功率以及旅途時間。

本模擬的資料範圍如下：

1. 車站：南港至彰化，共 49 站。
2. 原始時刻表：2022 年 9 月的時刻表[6]。我們將其各站的到達以及出發時間刪除，只留下停靠站規律以及進入首站的時間，並且將其轉為班表草稿的格式。
3. 乘客：2019 年 9 月的 OD 資料[7]，我們將其平日的搭乘人數進行統計後取平均，並且轉為模擬器的輸入格式。

我們透過上述資料進行模擬，並且將產生出來的時刻表以運行圖方式呈現。另外，由於臺鐵現行排點最小單位為 30 秒，因此我們模擬器產生的時間也是以 30 秒作為單位。

- 實驗結果：

1. 模擬結果：

執行時間	成功率	乘客平均 候車時間	乘客平均 乘車時間	乘客旅途時間
61.7321 秒	77.1621%	22 單位（11 min）	74 單位（37 min）	96 單位（48 min）

上表為本次模擬的執行結果。可以看到的是由於進行模擬的乘客量十分龐大（約 100 萬筆），因此在執行時間需要 61 秒鐘。但是整體而言本模擬器的執行速度仍十分迅速。在千人以內的模擬基本上皆可以在 0.3 秒之內執行完成。

2. 時刻表 JSON 檔以及運行圖：

我們的模擬器有辦法將所排出的時刻表轉為臺鐵公開資料的 JSON 格式，並且透過其他的軟體/平台將此檔案轉為其他樣式。在本實驗當中，我們將模擬產生出來的時刻表放入運行圖轉換軟體[8]當中，產生出如圖 22 的運行圖以利我們觀察所產生的時刻表。可以發現的是本研究模擬器所產生的時刻表皆可以成功化解衝突，所產生的時刻表也與實際的相似。

[6] 資料來源：政府資料公開平臺（資料集 ID: 6138）
[7] 本資料來源為交通部管理資訊中心。本研究透過公文向交通部臺灣鐵路管理局運務處請求准予以研究名義使用，並且利用交通部數據匯流平台提供下載。資料集名稱：臺鐵每日各站分時 OD 資料(O)。
[8] 本程式是透過開源程式碼開放平台 GitHub 取得，並按照 MIT License 規範進行使用。

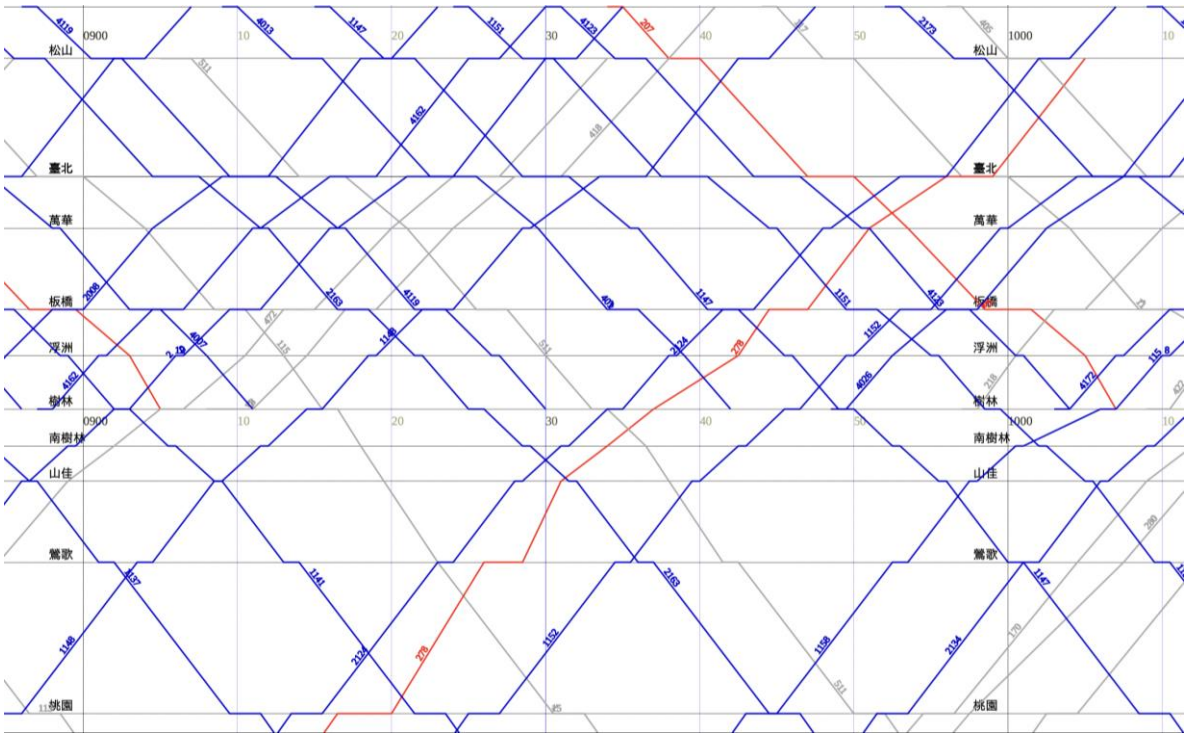


圖 22：模擬產出之運行圖（截取部分，完整檔連結請參見柒、參考資料與其他）

總結本實驗，我們確認了我們的模擬器能夠在極短時間內進行實際的列車排點、化解安全衝突，並且放入乘客進行時刻表優劣度的計算。另外，我們還有辦法將模擬器產生的結果轉為制式的 JSON 檔以方便其餘的工作使用，並且還可以將 JSON 檔進一步轉為調度員平常所使用的運行圖。

三、 最佳化演算法

（一） 實驗一：Hill climbing 演算法比較

在本實驗當中，我們將比較有無「初始解以 OD 優化」以及有無加入「更動列車數」這項擾動，對於求解的影響。由於 Hill Climbing Algorithm 無法處理過大的模型，因此我們排點的範圍如下：

1. 車站數：南港至桃園，共 11 站
2. 時間尺度：600 個半分鐘（5 小時）
3. 列車數目：10 列以內
4. 列車種類：區間車（EMU800 型, 定員 1262 人）、自強號（定員 566 人）、傾斜式自強號（定員 376 人）

5. 採用 multi start 方式優化，連續 500 次迭代無更佳解即停止，且每筆測試最多執行 3000 次迭代

在上述的範圍內，為了檢測此演算法是否可以針對各種情況進行排點，因此我們針對不同的乘客需求情況設計了四組輸入的乘客資料，每一組測試資料分別會有三筆測資，每筆測資將會被執行十次。另外由於每一組的測試資料由於乘客數量以及求解難易度不同，擁有不同的成功率門檻，分別如下表示：

		乘客在空間中離散程度	
		1組： 乘客的起迄站集中，多於同一處上/下車，人流方式較一致。	2組： 乘客多於不同處上/下車，可能為起站分散、迄站分散或是起迄站皆分散。
乘客在時間中離散程度	α組：乘客數量少且到站時間集中	$\alpha 1$：人少集中型 Success Rate Threshold (Θ_{SR})= 0.8 Train Amount = 5	$\alpha 2$：人少分散型 Success Rate Threshold(Θ_{SR}) = 0.8 Train Amount = 5
	β組：乘客數量多且到站時間分散	$\beta 1$：人多集中型 Success Rate Threshold(Θ_{SR}) = 0.7 Train Amount = 10	$\beta 2$：人多分散型 Success Rate Threshold(Θ_{SR}) = 0.7 Train Amount = 10

針對以上測資，我們有三種演算法要進行測試：

- HC 演算法：最基本的 Hill Climbing，無初始解優化，不搜尋列車數量
- INIT-HC 演算法：包含初始解優化之 Hill Climbing 演算法
- HC-TN：包含「更動列車數」擾動之 Hill Climbing 演算法。其中，α組的列車數量最大為 6 列，β組為 10 列。

另外，在本實驗中，由於本實驗範圍列車數較少，較不易發生衝突，且測資相對簡單，暫時不加入待避模式。

- 實驗結果：

表 1：Hill climbing 演算法比較結果統計（該項目中最佳者以粗體字表示）

		$\alpha1$（人少集中型）			$\alpha2$（人少分散型）			
測資	Search algorithm	HC	INIT-HC	HC-TN	HC	INIT-HC	HC-TN	測資
1	SuccessRate 平均	60.4%	59.2%	**73.7%**	55.0%	43.8%	**70.7%**	4
	SuccessRate 中位數	55.1%	55.1%	**81.6%**	54.0%	40.7%	**80.8%**	
	SuccessRate 標準差	0.1863	0.1174	0.2131	0.1579	0.1175	0.1930	
	SuccessRate 高於Θ_{SR}數量	3	1	**6**	1	0	7	
	旅途時間平均（30 sec）	**42**	42	45	68	---	**49.2857**	
	旅途時間中位數	**41**	42	**41**	68	---	**48**	
	旅途時間標準差	2.6458	---	10.9909	---	---	7.6095	
2	SuccessRate 平均	**60.2%**	61.1%	58.9%	56.1%	48.0%	**77.1%**	5
	SuccessRate 中位數	50.0%	**53.2%**	50.0%	55.1%	42.6%	**81.5%**	
	SuccessRate 標準差	0.2218	0.1767	0.1973	0.1916	0.1607	0.1294	
	SuccessRate 高於Θ_{SR}數量	2	**3**	**3**	2	1	**9**	
	旅途時間平均（30 sec）	**55**	72	57.3333	54.5	75	**53.8889**	
	旅途時間中位數	55	71	**51**	**54.5**	75	56	
	旅途時間標準差	21.2132	13.5277	16.4418	7.7782	---	8.6955	
3	SuccessRate 平均	58.4%	55.4%	**67.6%**	64.4%	60.7%	**81.1%**	6
	SuccessRate 中位數	51.6%	56.9%	**78.9%**	63.6%	65.9%	**81.8%**	
	SuccessRate 標準差	0.1756	0.1665	0.1954	0.1482	0.1942	0.0429	
	SuccessRate 高於Θ_{SR}數量	1	0	**3**	3	2	**9**	
	旅途時間平均（30 sec）	67	---	**50.6667**	60.6667	77	**47.3333**	
	旅途時間中位數	67	---	**36**	62	77	**47**	
	旅途時間標準差	---	---	27.1539	4.1633	15.5563	11.8533	
		$\beta1$（人多集中型）			$\beta2$（人多分散型）			
測資	Search algorithm	HC	INIT-HC	HC-TN	HC	INIT-HC	HC-TN	測資
7	SuccessRate 平均	60.8%	**62.3%**	59.1%	68.7%	57.0%	**71.1%**	10
	SuccessRate 中位數	51.1%	**52.9%**	51.1%	70.3%	56.7%	**70.6%**	
	SuccessRate 標準差	0.1299	0.1517	0.1027	0.0379	0.1288	0.0163	
	SuccessRate 高於Θ_{SR}數量	**4**	3	**4**	6	3	**10**	
	旅途時間平均（30 sec）	68	89.6667	**51.5**	68.6667	85	**64.4**	
	旅途時間中位數	57	102	**50**	69	82	**62**	
	旅途時間標準差	27.1784	24.9065	5.9722	13.1707	27.6225	10.2654	
8	SuccessRate 平均	57.2%	**63.9%**	59.2%	64.6%	54.0%	**70.5%**	11
	SuccessRate 中位數	67.4%	**69.2%**	**69.2%**	68.3%	51.5%	**70.3%**	
	SuccessRate 標準差	0.1487	0.1373	0.1542	0.0832	0.1134	0.0035	
	SuccessRate 高於Θ_{SR}數量	2	**5**	**5**	5	2	**10**	
	旅途時間平均（30 sec）	66	84.4	**59.4**	66.8	86	**61.9**	
	旅途時間中位數	66	77	**57**	69	86	**60.5**	
	旅途時間標準差	4.2426	24.7548	10.5262	4.4944	18.3848	8.3858	
9	SuccessRate 平均	64.5%	50.7%	**67.5%**	69.1%	67.5%	**71.0%**	12
	SuccessRate 中位數	65.4%	44.2%	**71.3%**	70.3%	69.8%	**70.8%**	
	SuccessRate 標準差	0.0899	0.1535	0.1118	0.0703	0.0776	0.0084	
	SuccessRate 高於Θ_{SR}數量	4	1	**6**	8	5	**10**	
	旅途時間平均（30 sec）	**64.25**	97	76.3333	53.625	80	**51.3**	
	旅途時間中位數	66	97	**64**	53	76	**51**	
	旅途時間標準差	4.9244	---	29.8440	9.7678	10.2225	7.8323	

各組測資表現：$\beta2 > \beta1 > \alpha1 > \alpha2$

演算法整體表現：$C > A > B$

- 結果討論

從上表中整體來看，在大部分的測資當中，HC-TN 對於旅客成功率的最佳化效果大於另外兩個演算法，而其旅途時間上的最佳化上，HC-TN 的中位數幾乎皆優於 HC，但平均值上則不一定。另外，INIT-HC 幾乎在所有的情況當中都表現最差。另外，由於β組（人多型）我們給予 10 列車，且Θ_{SR}值較低，因此各演算法達到門檻的數量皆較α組（人少型）多。

1. HC vs HC-TN

從表中我們發現，在$\alpha 2$（人少分散型）的測資當中，HC 與 INIT-HC 的成功率皆表現得十分不理想，但 HC-TN 卻有極高的成功率以及較低的旅 g;途時間(α2-1、α2-3)。於是我們決定觀察 HC-TN 在每一次求解當中迭代次數與時間、成功率、列車數之關係，如圖 23 所示，其中圖中的紅線代表成功率、藍線為旅途時間、綠線則為列車數量。

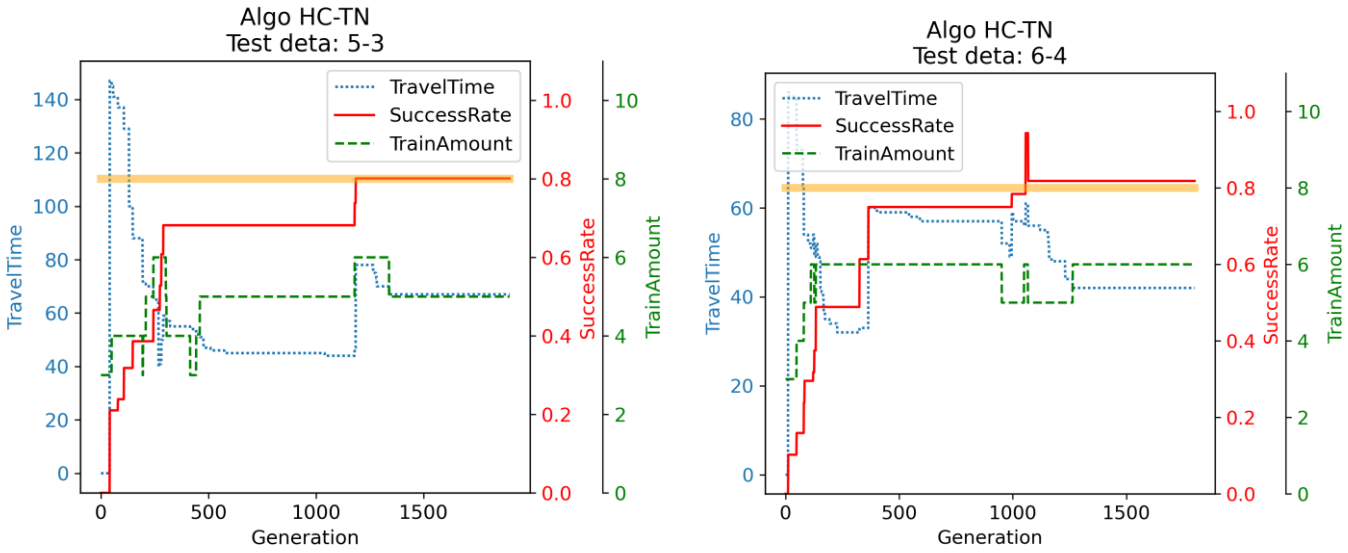


圖 23：HC-TN 迭代次數與時間、成功率、列車數之關係圖

（左：測資 α2-2, 右：測資 α2-3, 黃色線為Θ_{SR}位置；Travel Time 單位：30 秒）

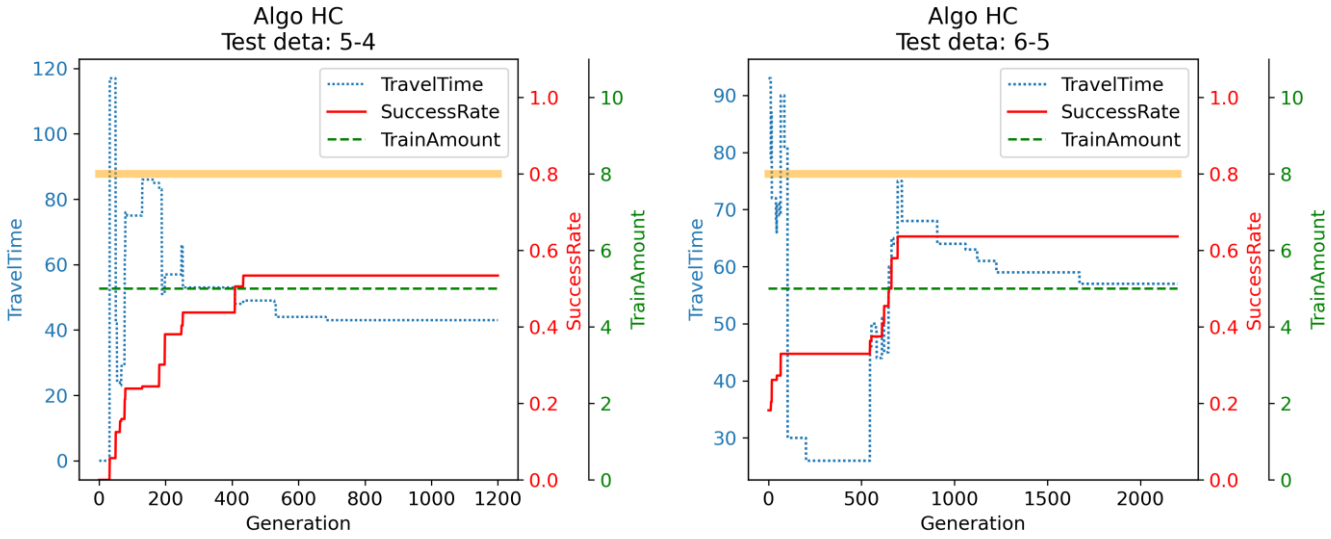


圖 24：HC 迭代次數與時間、成功率、列車數之關係圖（左：測資 α2-2, 右：測資 α2-3）

比較 HC 成功率的成長曲線，我們可以發覺在此測資組中，HC 的成功率曲線從開始的 20%以下往上成長，但是在 500 個 generation 之後達到約 50%~60%時便停止成長，整體成長幅度極緩。但是 HC-TN 的成功率曲線在 500 個 generation 便已經達到 70%以上，且隨後仍持續增長到 Θ_{SR}(80%)以上。另外，我們可以發現 HC-TN 成功率（紅線）之上升位置多和列車數（綠虛線）的變動位置一致，但其所使用的列車數不一定較多。由此可以推知其表現較佳與列車數可變動有一定的關聯。

因此我們推論在 $\alpha 2$（人少分散型）當中 HC-TN 明顯較佳的原因是因為該組人數少且分散，HC 較難僅透過調整時間、運行範圍及停靠站規律產生夠大擾動，使列車調整至旅客在時空中主要的分佈位置。但加入「淘汰不適應列車」以及「增加列車數量」功能的 HC-TN 則可以透過淘汰不適應列車之後所空出的列車數，再度隨機一列新的列車，使找到乘客的機率變大，因而提高找到較好結果的機率。而另外兩演算法則因為擾動不足，無法產生較好的解。而由於α2、β2的乘客空間中分佈較散，人流方向不明顯，因此此差異效果遠較α1和β1（集中型）明顯。

順帶一提的是，HC-TN 在β2（人多分散型）組當中時常可以產生高達 90% 以上甚至 100%的成功率，但是後續多伴隨著旅途時間的下降而使成功率降回Θ_{SR}附近，由此可知 β 組（人多型）的Θ_{SR}值應該可以再設更高，使各演算法差距更明顯。我們可以發覺 HC-TN 在許多測資當中時間皆較低，推測也是因為其 SuccessRate 攀升得快使得旅途時間有足夠迭代次數可以進行優化。

2. HC vs INIT-HC

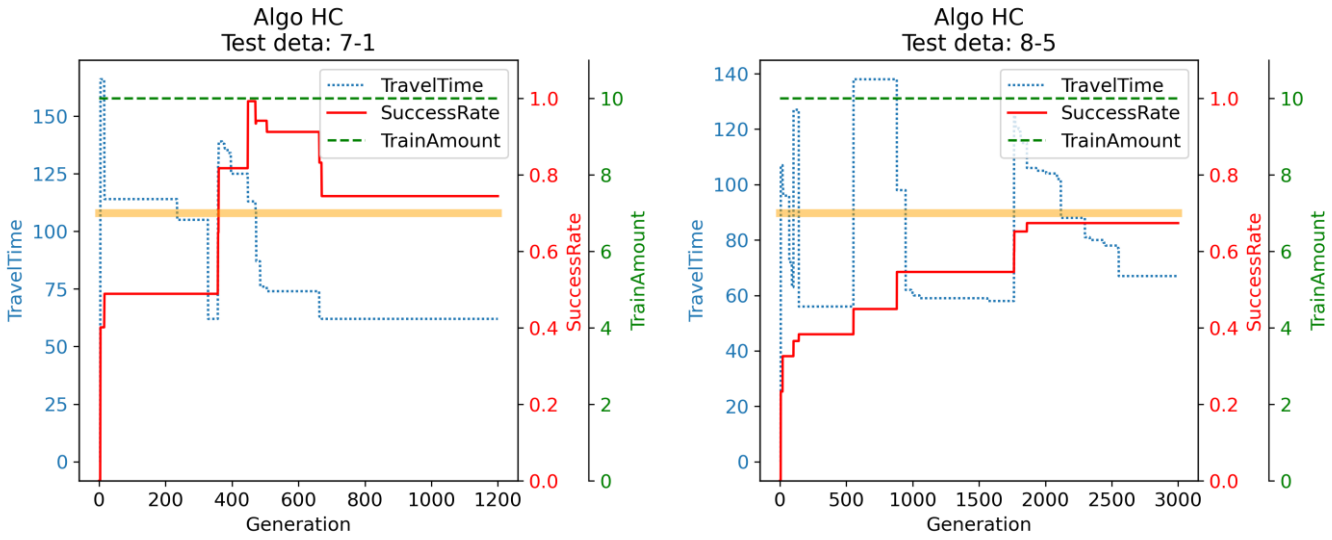


圖 25：$\beta 1$ 演算法 HC 迭代次數與時間、成功率、列車數之關係圖（Travel Time 單位：30 秒）

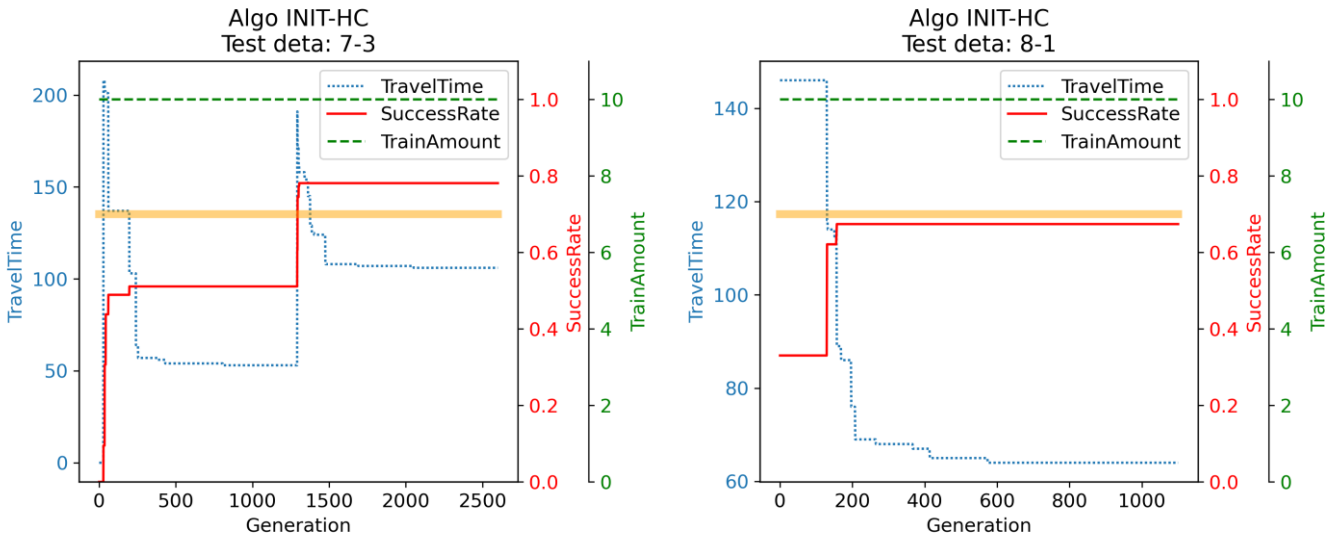


圖 26：β1 演算法 INIT-HC 迭代次數與時間、成功率、列車數之關係圖（Travel Time 單位：30 秒）

從實驗結果可以發覺 INIT-HC 在$\alpha 1$、$\alpha 2$、$\beta 2$中表現反而較沒有初始解優化的 HC 來的差，不但成功率門檻較低，旅途時間也較高；唯獨在β1時 INIT-HC 的 SuccessRate 表現較好，甚至有部分優於 HC-TN。

在分析迭代過程（圖 25、圖 26）之後，我們發覺相較於 HC 中可以看到 SuccessRate 隨著迭代次數逐步上升，INIT-HC 的 SuccessRate 圖形多半呈現直角形，也就是說成功率常常會直線上升到約 40%~50%，之後便停在該處不動。

我們推測此現象會出現的原因是我們的初始解優化時是按照各車站人數去做處理，並未考慮到方向以及時間等等，導致演化過程中可以在一開始達到約 50%的成功率，但是後續就陷入了所

謂的區域低點，以致於無法找到能載到更多人的解。因此，在人流方向較好預測的集中型測資（α1 & β1）中，初始解優化較可以發揮其效果，快速的讓成功率上升，但當到人流分散較不明顯的分散型測資（α2 & β2）時，有初始解優化的 INIT-HC 便比不上隨機性較高的 HC，或是擾動較多的 HC-TN 了。

3. 穩定度測試

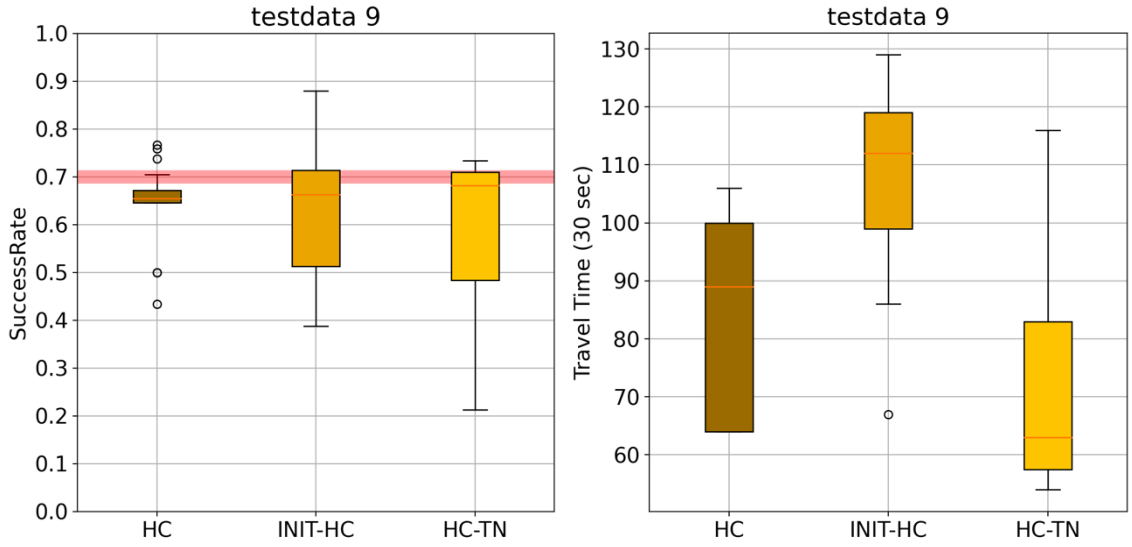


圖 27：測資β1-3 盒鬚圖統計（左圖：成功率/右圖：旅途時間）

由於本實驗所使用的演算法皆為隨機演算法，因此為了檢查各演算法的穩定度，我們將其中幾筆測資放入各演算法中跑十到二十次，並且將其結果製程上圖。可以發現對於上面兩張圖，INIT-HC 所分佈的範圍皆較廣，也就是在穩定度方面皆為 INIT-HC 最差，推測其原因是因為 INIT-HC 容易陷入區域最佳解當中，故每次的結果差異性較大。

另外我們可以比較 HC 和 HC-TN 在兩張圖中的表現。首先先看成功率（左圖），我們可以發覺 HC-TN 高於門檻值的情形以及中位數皆較 HC 好，但是，HC-TN 卻有部分次執行的成功率較低，使得整體穩定度較 HC 差。

但反觀旅途時間（右圖），我們可以看到 HC 的穩定度也降低了，但是 HC-TN 的時間明顯的較 HC 來得低，因此比較起來 HC-TN 的整體表現仍比 HC 好。

總結本實驗，我們成功透過 Hill Climbing 演算法在短時間內進行多筆不同的鐵路時刻表排點，並且發現 HC-TN 優化方式可以找出成功率皆高於門檻、旅途時間又較低的時刻表。另外，透過比較各種優化方式，我們發現初始解優化的效果僅能在乘客人流流向明顯時有較好的結果，但列車

數擾動在各組測資當中都有不錯的結果。因此可以歸論在本研究的問題模型當中，擾動越大對於求解是有較好的表現。

（二） Threshold Accepting 加入與否對於求解之影響

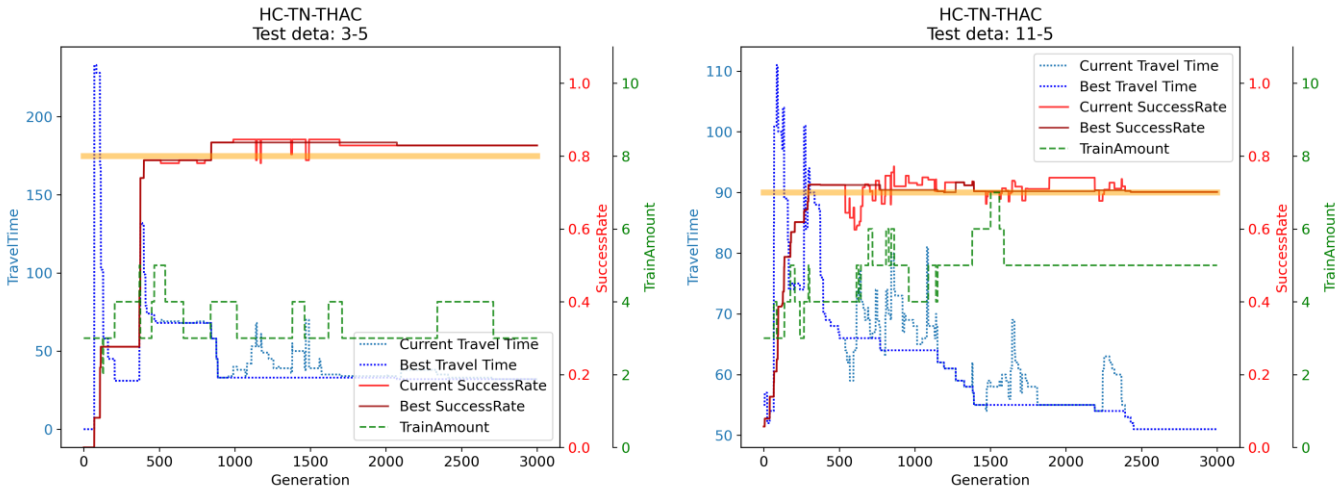
在本實驗當中，我們將比較有無加入 Threshold Accepting 對於求解品質的影響。我們把上一實驗當中表現最好的 HC-TN 演算法加入 Threshold Accepting，並且分析有無加入對於我們求解的影響。另外，本實驗所採用的測資以及實驗範圍皆與上一實驗相同。

整理本實驗所使用的演算法如下：

- HC-TN：包含「更動列車數」擾動之 Hill Climbing 演算法。由於測資皆和上個實驗相同，因此將不再重複實驗，直接以上個實驗的結果進行比較。
- HC-TN-THAC：將 HC-TN 加入 Threshold Accepting 的優化。其中，在本實驗當中，φ_{SR}值由 0.1 開始線性減少，而φ_{time}值由 100 個單位時間 (50 分鐘) 開始減少。另外，為避免成功率無限下降導致最終解過差，我們設計演算法每進行 30 次迭代才可以允許變差一次，且最低成功率不允許下降至 50%以下。

另外，在本實驗中，也暫時不加入待避模式。

- 實驗結果



上圖為本實驗中幾筆測資的 HC-TN-THAC 迭代過程與成功率、旅途時間折線圖。圖中淺藍/淺紅分別代表迭代過程中所「接受」的解的旅途時間/成功率。而深藍/深紅則代表演化到該次為止所出現過的最好的解的旅途時間/成功率。

首先可以觀察到的是 HC-TN-THAC 在迭代的過程中，相較於其他我們做過的演算法，可以看到十分明顯、劇烈的上下波動，尤其是 Current Travel Time 和 Current SuccessRate 的曲線。而此

現象主要便是因為加入 Threshold-Accepting 之後當前解所允許的範圍變大了，因此曲線擾動的量也隨之增加。

表 2：Threshold Accepting 加入與否的比較（該項目中最佳者以粗體字表示）

		$\alpha 1$		$\alpha 2$		
測資	Search algorithm	HC-TN	HC-TN-THAC	HC-TN	HC-TN-THAC	測資
	SuccessRate 平均	73.7%	**77.3%**	70.7%	**81.0%**	
	SuccessRate 中位數	**81.6%**	**81.6%**	**80.8%**	**80.8%**	
	SuccessRate 標準差	0.2131	0.1634	0.1930	0.0085	
1	SuccessRate 高於Θ_{SR}數量	6	**7**	7	**10**	4
	旅途時間平均（30 sec）	45	**37**	49.2857	**48**	
	旅途時間中位數	41	**37**	48	**47.5**	
	旅途時間標準差	10.9909	2.4495	7.6095	5.8500	
	SuccessRate 平均	58.9%	**82.3%**	77.1%	**80.9%**	
	SuccessRate 中位數	50.0%	**82.4%**	**81.5%**	81.3%	
	SuccessRate 標準差	0.1973	0.0231	0.1294	0.0066	
2	SuccessRate 高於Θ_{SR}數量	3	**9**	9	10	5
	旅途時間平均（30 sec）	57.3333	**50.8889**	53.8889	**47.5**	
	旅途時間中位數	51	**50**	56	**46.5**	
	旅途時間標準差	16.4418	16.0581	8.6955	5.8166	
	SuccessRate 平均	67.6%	**72.4%**	81.1%	**82.4%**	
	SuccessRate 中位數	78.9%	**80.1%**	81.8%	**82.4%**	
	SuccessRate 標準差	0.1954	0.1569	0.0429	0.0144	
3	SuccessRate 高於Θ_{SR}數量	3	**5**	9	**10**	6
	旅途時間平均（30 sec）	50.6667	**35.2**	47.3333	**36.2**	
	旅途時間中位數	36	**34**	47	**35.5**	
	旅途時間標準差	27.1539	3.2711	11.8533	5.5337	
		$\beta 1$		$\beta 2$		
測資	Search algorithm	HC-TN	HC-TN-THAC	HC-TN	HC-TN-THAC	測資
	SuccessRate 平均	59.1%	**65.8%**	**71.1%**	70.8%	
	SuccessRate 中位數	51.1%	**70.4%**	**70.6%**	70.3%	
	SuccessRate 標準差	0.1027	0.1029	0.0163	0.0110	
7	SuccessRate 高於Θ_{SR}數量	4	**7**	10	10	10
	旅途時間平均（30 sec）	51.5	**51**	64.4	**56.2**	
	旅途時間中位數	50	**51**	62	**55**	
	旅途時間標準差	5.9722	2.7689	10.2654	5.0509	
	SuccessRate 平均	59.2%	**63.3%**	70.5%	**70.8%**	
	SuccessRate 中位數	69.2%	**70.9%**	70.3%	**70.4%**	
	SuccessRate 標準差	0.1542	0.1433	0.0035	0.0092	
8	SuccessRate 高於Θ_{SR}數量	5	**7**	10	10	11
	旅途時間平均（30 sec）	59.4	**56.5714286**	61.9	**55.1**	
	旅途時間中位數	57	**52**	60.5	**55**	
	旅途時間標準差	10.5262	10.0309	8.3858	4.6536	
	SuccessRate 平均	**67.5%**	64.8%	71.0%	**71.6%**	
	SuccessRate 中位數	**71.3%**	70.8%	70.8%	**71.3%**	
	SuccessRate 標準差	0.1118	0.1102	0.0084	0.0118	
9	SuccessRate 高於Θ_{SR}數量	**6**	**6**	10	10	12
	旅途時間平均（30 sec）	76.3333	**68.1667**	51.3	**44.2**	
	旅途時間中位數	64	**62.5**	51	**43**	
	旅途時間標準差	29.8440	23.5407	7.8323	6.8443	

上表為 HC-TN 和 HC-TN-THAC 的執行結果整理。我們可以很明顯地看到在幾乎所有的測資當中，無論是成功率還是旅途時間，HC-TN-THAC 皆勝過 HC-TN。而各組測資所花費的旅途時間皆在 40~60 個單位時間（20~30 min）左右，皆十分接近本測資範圍理論上的最短乘車時間。

但是，如果看的仔細一點我們可以發覺 HC-TN-THAC 在成功率方面並沒有大勝 HC-TN 許多。但根據觀察迭代中途成功率的變化，在許多筆測試資料當中，HC-TN-THAC 甚至一度可以達成 100%的成功率，但後續隨著時間的優化，成功率變因此而下降，而 HC-TN 也時常可以達到約九成的成功率。因此我們推測我們無法明顯區分出兩者差異的主要原因是因為我們所設定的Θ_{SR}值對於此二演算法來說皆太容易達成，因此，沒有辦法明顯拉出兩者的差距。

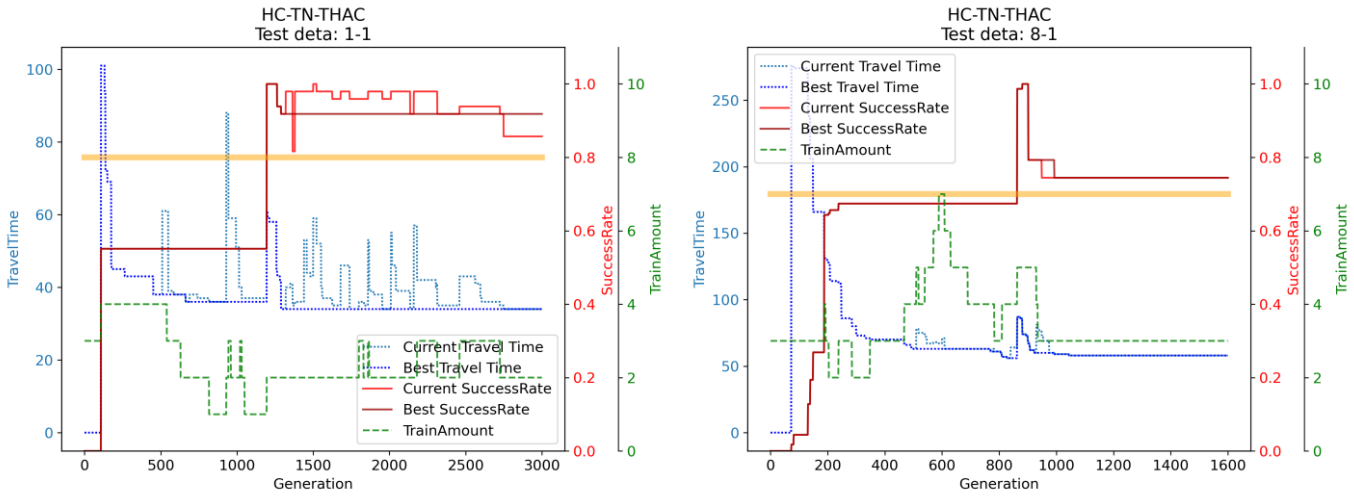


圖 28：HC-TN-THAC 有時可達成 100%成功率

比較其旅途時間的優化程度，我們可以看到在較為簡單的集中型測資（α1 & β1）當中，HC-TN-THAC 也很難明顯贏過 HC-TN，但是到了較困難的分散型測資當中，則可以觀察到 HC-TN-THAC 旅途時間較 HC-TN 短了約 10 個單位時間（5 min）。

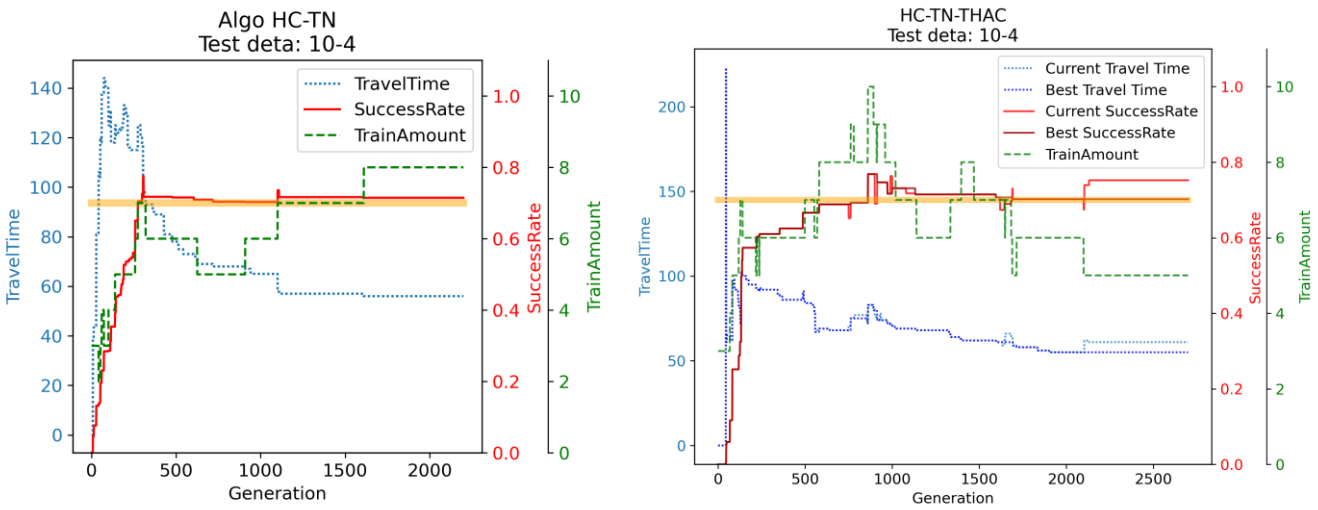


圖 29：HC-TN 與 HC-TN-THAC 最佳化過程圖比較

比較上面兩張圖片，我們似乎可以找出 HC-TN-THAC 旅途時間小勝 HC-TN 的原因。觀察旅途時間在迭代後半段，我們可以發覺 HC-TN 在 1000 次迭代之後，旅途時間基本上已達到一個夠低的值，因此曲線（藍線）持平一段時間之後，便被 multi start 判定為已陷入低點而停止。但是 HC-TN-THAC 由於 Threshold-Accepting 機制持續接受較差的解，反而使最佳解可以不停的小幅度進步，因此可以找到較 HC-TN 好的解。然而，HC-TN-THAC 有一個小缺點在於由於此演算法可以持續進行優化，因此在執行時所花費的時間皆較其他演算法長許多。

- 穩定度分析

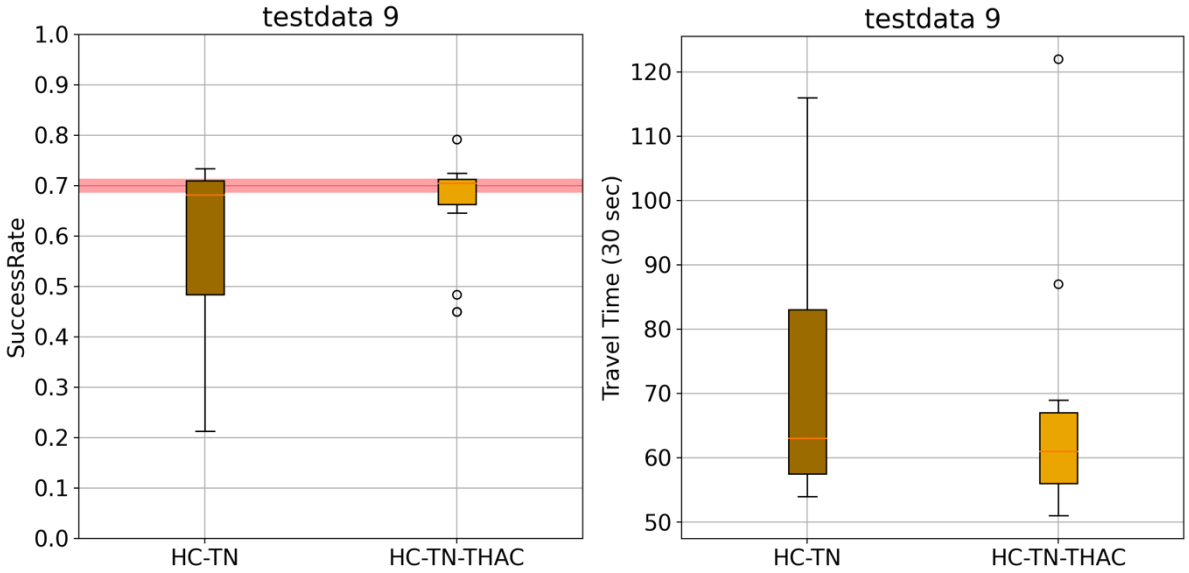


圖 30：測資 9（β2-3）加入 Threshold-Accepting 與否之盒鬚圖統計 （左圖：成功率/右圖：旅途時間）

如果觀察上面兩張圖片，我們可以發現 HC-TN-THAC 和 HC-TN 的中位數幾乎是一致的。但是，HC-TN 的穩定度卻比 HC-TN-THAC 低很多。

我們推測會有這個結果的原因是 HC-TN 雖然本身擾動性已經十分的高，但是只靠擾動依然無法避免其有時候會陷入區域低點的特性，也因此 HC-TN 在有些時候的表現會較差。但是加入 Threshold-Accepting 優化的 HC-TN-THAC 演算法可以直接「走」出這些區域低點，因此每一次皆可以找到一個相似且最佳化效果極好的解。

總結本實驗，我們比較了有無加入 Threshold-Accepting 優化對於求解的影響。我們發覺 HC-TN 演算法無論有沒有進行優化，在我們所給予的測資中皆有不錯的表現。但是透過 Threshold-Accepting 優化，不但減少了陷入區域最低點的情形，還使求解穩定度增加、以及對旅途時間擁有更佳的優化效果。因此加入 Threshold-Accepting 優化對於求解有正面的影響。

伍、 結論

本研究設計出了一個模擬系統，能夠在讀取編碼過後的時刻表（班表草稿）之後，透過模擬各種衝突情況的發生，以一定的方式有效率的化解衝突，產生一無安全疑慮、無衝突之時刻表，大幅節省人工化解衝突的時間與人力。並且本模擬器可以透過 OD 資料中記錄的旅運數據，分析一時刻表的乘客成功率以及所花費的旅途時間，以評估一時刻表的優劣。

另外，為了測試模擬器的效果，我們將南港彰化之間實際的時刻表退化成班表草稿，並且放回模擬器當中，重新進行排點。並且我們取得 2019 年的歷史 OD 資料，並針對此時刻表進行評估。根據實驗結果，我們發覺我們的模擬器可以在一分鐘內模擬 100 萬名乘客，且確實的產生無衝突之時刻表，並且還能以運行圖的方式供人工方式檢閱。

本研究利用 Hill Climbing 演算法，配合前面產生之模擬器，進行最佳化時刻表的搜尋。另外，我們還針對有無初始解優化、有無改變列車數量進行實驗，並且發覺有初始解優化的 INIT-HC 演算法只有在乘客空間中分佈集中、人流方式明顯的測試資料中有較好的成功率表現。而加入「更動列車數」擾動的 HC-TN 演算法則在各組測資中皆有較高的成功率以及較低的旅運時間，表現得比只有基本擾動的 HC 演算法好。另外，我們也分析了各個演算法求解的穩定度，發覺 HC-TN 在整體表現得最好，但是成功率穩定度略輸 HC 演算法。從以上實驗結果當中，我們歸納出增加 Hill

Climbing 演算法的擾動，相較於讓初始解優化，能夠有更好的表現，降低陷入區域低點的機率。因此，往後再設計演算法時應往提高擾動方面設計。

另外，針對容易 Hill Climbing 演算法容易陷入區域低點的問題，我們嘗試以 Threshold-Accepting 機制來進行優化，看看能否讓最佳化演算法更上一層樓。我們比較了 HC-TN-THAC 和 HC-TN 在各組測試資料中的結果，發覺 Threshold-Accepting 的確能有效的降低旅途時間。但由於兩演算法皆可以在成功率上面擁有非常良好的表現，因此成功率表現上的優化幅度較小。因此，後續應該設計別組更困難的測試資料，如此應該可以更明顯的看出兩者的差距。

透過以上方式改良演算法，我們成功將 Hill Climbing 演算法的成功率由原本 60%以下提升到可以穩定地達到 70%或 80%的門檻值，最多甚至直接提升 40%的成功率，達成 100%成功率的極佳表現。另外，也讓旅途時間降低 10~30 個單位時間，大幅減少旅客乘坐鐵路所要花費的時間。

陸、 未來展望與應用

一、 嘗試以更多演算法進行最佳化

本研究所採用的 Hill Climbing 演算法在本質上較容易陷入區域最佳解。因此，我們希望能夠測試其他的演算法，例如避免重複搜尋的禁忌搜尋法(Tabu Search)，或是可以進行全域搜尋的基因演算法 (Genetic Algorithm)，並且進行比較，以找出較好的作法

二、 增加模擬器功能、進行大規模的時刻表排點、建立使用者介面

本研究的模擬器依然有許多進步空間。我希望在後續可以加入對於多軌道（三軌甚至四軌並行）的支援，以及加入以編組作為排點單位的功能。並且在配合最佳化演算法搜尋能力的提升，進行實際大規模的鐵路時刻表排點。最後，我希望可以建立一個擁有使用者介面的軟體，使得鐵路運轉單位能夠更方便的使用此研究的結果協助其時刻表的安排。

三、 應用性

本時刻表排點系統擁有許多應用價值。例如興建全新路線之後可以用此系統進行排點，或是使用第二模擬器估算列車運量。而模擬器也允許台鐵局人工設計班表草稿的方式進行衝突化解與

排點，因此此系統若完成也可以提升排點工作的效率。另外，若當部分路線因為各種原因暫時性停駛時，也可以利用此系統快速地進行臨時列車班表的安排。

柒、 參考資料與其他

- 許書耕, 許修豪, 陳春益, 林東盈, 李威勳, 袁永偉, 郭昭佑, 吳美玲, 顏利憲, & 李宇欣. (2015). 鐵路列車自動化排點系統功能擴充與推廣應用 (初版). 交通部運輸研究所.
- 呂卓勳. (n.d.). 台灣鐵路運行圖網站. https://tradiagram.com/
- 呂卓勳. (2020). 台灣鐵路局公開資料 JSON 繪製鐵路運行圖(SVG 格式)程式 [Source code]. https://github.com/billy1125/TRA_time_space_diagram
- Caprara, A., Kroon, L., Monaci, M., Peeters, M., & Toth, P. (2007). Passenger railway optimization. Handbooks in operations research and management science, 14, 129-187.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269-271.
- Dueck, G., & Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Journal of computational physics, 90(1), 161-175.
- Leo Kroon, Gábor Maróti, Lars Nielsen (2014) Rescheduling of Railway Rolling Stock with Dynamic Passenger Flows. Transportation Science 49(2):165-184.
- McMinn, P. (2011, March). Search-based software testing: Past, present and future. In 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (pp. 153-163). IEEE.
- Niu, H., Zhou, X., & Gao, R. (2015). Train scheduling for minimizing passenger waiting time with time-dependent demand and skip-stop patterns: Nonlinear integer programming models with linear constraints. *Transportation Research Part B: Methodological*, *76*, 117-135.
- Robenek, T., Maknoon, Y., Azadeh, S. S., Chen, J., & Bierlaire, M. (2016). Passenger centric train timetabling problem. Transportation Research Part B: Methodological, 89, 107-126.
- Tormos, P., Lova, A., Barber, F., Ingolotti, L., Abril, M., & Salido, M. A. (2008). A genetic algorithm for railway scheduling problems. In Metaheuristics for scheduling in industrial and manufacturing applications (pp. 255-276). Springer, Berlin, Heidelberg.
- 列車模擬實驗之完整運行圖結果下載連結：
  https://drive.google.com/file/d/1rbwSYDhTrI3R_GpEr5wdkpLnvlUiZSxr/view?usp=sharing

捌、 致謝

本研究在進行過程當中有受到許多臺鐵局以及交通部同仁的協助，包含臺鐵局運務處與交通部資料中心的承辦人員協助我們取得臺鐵分時 OD 資料，以及運務處綜合調度所計劃組楊組長、調度員葉先生願意撥空和我們進行訪談，告訴我們臺鐵局內現行的時刻表安排作業方式以及他們遇到排點衝突時的化解方式。本人在此鄭重感謝他們的協助以及對於此研究的貢獻。

【評語】190013

Hill Climbing Algorithm 無法處理所有的列車、火車站的模型，其在本研究的實際價值如何？"我們會將乘客隨機切分成 5 人以內的小團體，並且隨機設定其可以接受的轉乘次數 k"，這樣做的理由是什麼？所排出來的時刻表與原來時刻表的差異如何？改進了多少？可以同時考慮到其他的排程限制，如人員出缺勤、假日加班因素？

本作品設計一個模擬系統，能夠在讀取編碼過後的時刻表（班表草稿）之後，透過模擬

各種衝突情況的發生，以一定 metaheuristic 的方式有效率的化解衝突，產生一無安全疑慮、無衝突之時刻表，期能大幅節省人工化解衝突的時間與人力。此外此作品使用登山演算法來找較好的時刻表。此作品對排時刻表這種問題有深入的研究和探討且具有實用的價值。

以英語報告成果非常流暢自然而且非常清楚。