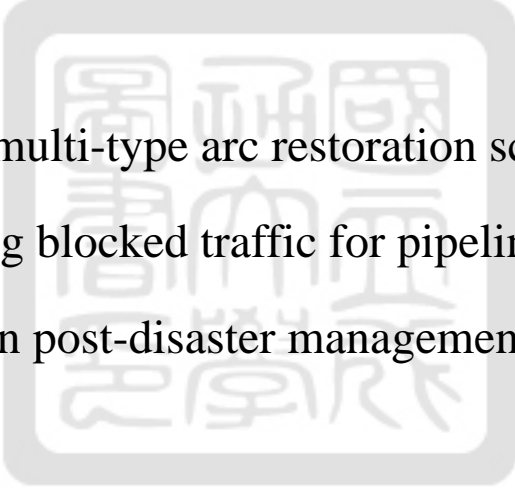


國立成功大學  
資訊管理研究所  
碩士論文

考慮多類管線與交通不便之災後  
管線網路最佳修復排程研究

Optimal multi-type arc restoration scheduling  
considering blocked traffic for pipeline networks  
in post-disaster management



研究生：洪筱昀

指導教授：王逸琳 博士

中華民國一百零六年六月

國立成功大學

碩士論文

考慮多類管線與交通不便之災後管線網路最佳修復排  
程研究

Optimal multi-type arc restoration scheduling  
considering blocked traffic for pipeline networks in  
post-disaster management

研究生：洪筱昀

本論文業經審查及口試合格特此證明

論文考試委員：

王逸琳

李家宏

盧孝成

李宇欣

指導教授：

王逸琳

系(所)主管：

李興燦

中華民國 106 年 6 月 14 日

## 摘要

埋在地下的多類管線若因天災人禍而造成毀損，將導致民生需求之水、電、瓦斯等物資無法以地下管線輸送給居民。修復這些毀損的多類管線皆需要歷經開挖、修復、及回填等多項工程。由於此類管線經常埋在道路下，在其開挖後但尚未回填的期間極可能造成交通壅塞不便，特別當類型不同的管線可能個別需要特定修復技能、且可能隸屬不同公司的修復工作隊時，如何合宜地安排開挖與各類管線修復等各項工程之進行時程，以讓各段節線上的民眾等待水、電、瓦斯等物資的時間以及因維修造成之交通不便時間總和越短越好，是災後管理中的重要議題，也是本研究主要探討的管線網路修復排程問題。

本研究之主要貢獻在於首度將同段節線的道路開挖、多類型管線修復、以及對交通不便的影響性一併列入排程考慮，首先我們以單類管線修復排程文獻的整數規劃模式為基礎，將此排程問題視為一個具資源限制下之專案排程問題(Resource Constrained Project Scheduling Problem, RCPSP)，並推導出一個時空網路中基於多商品流量的排程整數規劃模式；為能更有效地求解，我們設計兩種貪婪演算法，依排程當下各節線各類型管線的物資連通狀態，優先選取至少一端未連通的毀損邊界節線，來進行各類開挖及修復工程；為能再更進一步改善求解品質，我們亦參考平行機台文獻中的基因演算法，修改成符合本研究題目的流程。測試結果顯示本研究所提出之貪婪演算法的確能有效地在短時間內獲得品質還好的可行解，若以其當最佳化軟體 GUROBI 或基因演算法的初始解，亦可幫助提早收斂求解過程。然而可能因為本問題困難度實在太高，目前為止我們所發展的基因演算法疊代多次之後仍不易跳脫出貪婪演算法所獲得的初始解，最後我們提出包括納入道路開挖而未修復的時間上限、如何偵測評估毀損狀況排程等數個富挑戰性的研究議題。

**關鍵字：**網路修復排程問題、專案排程、整數規劃、資源限制、災後管理

# **Optimal multi-type arc restoration scheduling considering blocked traffic for pipeline networks in post-disaster management**

Xiao-Yun Hong

I-Lin Wang

Institute of Information Management

## **SUMMARY**

This thesis investigates how to effectively restore damaged underground pipeline networks in post-disaster management. Suppose the location and restoration time for each damaged arc of different commodities (water, gas, electricity, ..., etc.), as well as the number of excavation and restoration teams for different commodity networks, have been estimated. The arc restoration scheduling problem seeks the optimal restoration schedule that effectively assigns restoration teams of different commodities to restore all the damaged arcs in specific time and order, so that the total waiting time of all demands of all commodities, including the induced traffic inconvenience, can be minimized. We view this NP-hard problem as a specialized resource constrained project scheduling problem with network structure. It is especially difficult since the precedence relations between arcs are not clear, and highly dependent on the earlier scheduling decisions. In addition, the excavation has to be done before any restoration tasks, yet parallel pipeline restorations of different commodities on the same arc can be conducted at the same time.

We propose an integer programming formulation and derive a few valid inequalities in the hope to shorten the solution time. Two greedy heuristic algorithms, Generic Greedy Algorithm (GGA) and Boundary Greedy Algorithm (BGA), are designed to calculate good scheduling decisions in greedy fashions. BGA performs better than GGA since it selects boundary arcs of shorter restoration time or larger demands to restore. A Genetic Algorithm (GA) modified from previous works in parallel machine scheduling has been implemented. We have generated random networks of tree or general topologies. The computational experiments indicate the integer programming formulation solved by GUROBI, a state-of-the-art solver, is too time consuming. BGA produces better schedules than GGA. Using the BGA solution as a start for GA and GUROBI does improve the qualities of solutions, but then we are troubled in getting off local optimum.

**Keywords:** Network restoration scheduling problem, Project scheduling, Integer programming, Resource constrained, Post-disaster management

## INTRODUCTION

The underground pipelines ship important daily living commodities like water, gas, electricity or telecommunication packets. There are demands over each segment (i.e., arc) of the pipeline network. Suppose these pipelines are damaged by disasters. How to effectively restore these pipelines so that the demands of all commodities can be satisfied as soon as possible is an important quick response task in post-disaster management. This arc restoration scheduling can be viewed as a specialized resource constrained project scheduling problem, which is NP-hard. It is extremely difficult because the precedence relations between arcs depend on the connectivity of nodes and earlier scheduling decisions. In addition, in this thesis we further take the inconvenience of blocked traffic due to the excavation and restoration tasks into consideration. This is also the first research work to restore multi-type pipelines at the same time.

## MATERIALS AND METHODS

Based on previous work by Lin (2016) that only restores a single-type undergraduate pipeline network, we further extend his mathematical model and propose an integer programming (IP) formulation to deal with multi-type pipelines. Our IP can also deal with the precedence relation between excavation and any restoration tasks, and includes the total traffic blocking time in the objective function. However, this IP model is very time consuming, even for solving small tree-like networks. In order to calculate good solutions within shorter time, we design two greedy heuristic algorithms, named GGA and BGA, where the former assigns an idle team to restore or excavate a pipeline of shortest time and the latter selects a boundary damaged arc of shortest processing time with higher priority. We also develop a Genetic Algorithm (GA) using the arc-team assignment as a chromosome.

## RESULT AND DISCUSSION

We generate random connected network families of two shapes: tree and general, with small and medium sizes. The IP formulation is solved by GUROBI, the state-of-the-art optimization solver. We use 5000 secs as a time limit for testing, since the convergence of the solving procedure does not improve much even for 10000 secs. In general, BGA does calculate better solutions than GGA. Using the BGA solution to the IP as a warm start, we could usually get a better solution. Similarly using the BGA solution to GA as a parent chromosome does improve the solution quality but then the local optimum is difficult to escape. The computational results show the difficulty of this problem.

## CONCLUSION

This thesis investigates an important NP-hard arc restoration scheduling problem in post-disaster management. Each damaged pipeline has to be excavated and then restored. One arc may contain several types of damaged pipelines. The traffic over each arc will be blocked in the duration of excavation and restoration. We propose an IP formulation, two greedy heuristic algorithms, and one genetic algorithm. The computational experiments indicate the boundary greedy algorithm can calculate a better solution, which can be used as a warming start for solving the IP or as a parent chromosome for the genetic algorithm. Our major contributions are in three folds: (1) we develop the first IP formulation for this problem; (2) we propose the boundary arc first concept and design an effective greedy algorithm; (3) we conduct extensive computational experiments, and show the computational difficulties. Although our proposed solution methods still do not have satisfactory performance yet, we hope our efforts can intrigue more researchers to investigate related topics.



## 誌謝

感覺昨天才身為 ilin-lab 碩 0 的學生，今天就在寫誌謝，這兩年過得很充實，時間才會那麼快的流逝，首先我要感謝恩師王逸琳 教授，我這麼廢還收我當研究生，我腦筋不靈活且又不會舉一反三，比起大我一屆的小秉、真泰、采緹，我真是望塵莫及，就是因為我太廢了，真的很感激逸琳老師沒有放棄我，願意花時間耐心的教導我與栽培我，常常發現老師為了跟學生 meeting，不想讓學生等太久，馬拉松式的跟不同學生 meeting，而沒吃午餐或晚餐，老師您對學生的用心，我都看在眼裡，有許多的感謝想對老師說，但是不是三言兩語就能表達完整，最後還是要跟您說聲：老師您辛苦了!!!

接下來我要感謝小秉、貞泰、采緹，你們三個讓我體會到，有強者學長姐罩著，天塌下來我也不怕，從你們思考方式與桌遊功力，就可以發現聰明的學生，推理邏輯就是跟別人不一樣；偉德的魔術讓我大開眼界，宛如世界級的魔術秀，直接搬進實驗室，讓實驗室隨時充滿歡樂；感謝富元帶來豐富的桌遊；柏寬要常來實驗室喔！冠瑋很會講話，從他身上學到許多講話的技巧；思涵的數學能力真的超強。在我碩士的最後一個暑假，很謝謝碩 0 的昀軒、嘉豪、雪湄都待在 lab，讓我不會覺得是孤單一個人，跟你們三個還有泰嘉一起吃午餐和晚餐，因為有你們四個，這個暑假，我真的很快樂。當然還有許多 Lab 歷屆學長姐和惠嘉 Lab 朋友、邑築也幫我很多，無法在此一致謝。

最後我要感謝我的家人、男友和找工作過程中幫我的貴人，一直默默的在背後支持我。隔行如隔山，因為有你們，我才有辦法堅持到最後，進到聯發科工作，這過程很艱辛，一個我完全不太熟悉卻充滿熱忱的領域，希望我可以好好珍惜，這得來不易的工作機會。

最後，還是要跟幫過我的人，說聲謝謝!!!

## 目錄

摘要.....	I
誌謝.....	V
目錄.....	VI
圖目錄.....	IX
表目錄.....	XII
第 1 章 緒論.....	1
1.1 研究背景與動機.....	1
1.2 研究目的.....	2
1.3 研究假設與範圍.....	4
1.4 論文架構.....	4
第 2 章 文獻回顧.....	5
2.1 網路修復相關文獻.....	5
2.2 資源限制下之專案排程問題 (RCPS) 相關文獻.....	10
2.2.1 符號定義.....	12
2.2.2 數學模式.....	13
2.3 小結.....	14
第 3 章 網路節線修復問題之整數規劃模式.....	15
3.1 問題描述.....	15
3.2 問題假設.....	15
3.3 NP-Hard 說明.....	16
3.4 用多商品網路流(Multi-commodity Network Flow，MCF)之不同類型管線多	



工作隊整數規劃模式 .....	17
3.4.1 符號定義 .....	18
3.4.2 整數規劃模式 .....	20
3.4.3 有效不等式 .....	24
3.5 小結 .....	24
第 4 章 節線修復排程演算法之設計 .....	25
4.1 貪婪式演算法 .....	25
4.1.1 基本貪婪式演算法(GGA)介紹 .....	25
4.1.2 邊界貪婪式演算法(BGA)介紹 .....	26
4.1.3 邊界貪婪式演算法範例介紹 .....	29
4.1.4 節線打通時刻計算方式 .....	31
4.1.5 節線打通時刻計算演算法(NCTA) .....	33
4.2 基因演算法(Genetic Algorithm , GA) .....	36
4.2.1 基於多平行機台(unrelated parallel machine)排程問題的基因演算法 ..	36
4.2.2 基因演算法範例說明 .....	40
4.3 小結 .....	42
第 5 章 整數規劃模式與演算法之數值分析 .....	43
5.1 測試網路資料 .....	43
5.1.1 產生網路結構圖工具 .....	43
5.1.2 視覺化網頁工具 .....	43
5.2 整數規劃模式比較 .....	44
5.3 貪婪式演算法測試 .....	47
5.3.1 不同類型貪婪式演算法測試 .....	47
5.3.2 貪婪選擇納入單位需求考量之測試 .....	49
5.4 基因演算法測試 .....	50

5.5 小結.....	51
第 6 章 結論與未來研究.....	52
6.1 結論.....	52
6.2 未來研究方向.....	53
參考文獻.....	55



## 圖目錄

圖 1.1 欲修復網路圖 1 .....	3
圖 1.2:第一種修復順序排程圖 .....	4
圖 1.3:第二種修復順序排程圖 .....	4
圖 2.1: PERT 圖 .....	11
圖 3.1 管線配置圖(圖片來源:高雄市政府).....	16
圖 3.2:使用 MCF 之多工作隊整數規劃示意圖 .....	17
圖 3.3:使用 MCF 之多工作隊時間示意圖 .....	18
圖 4.1 邊界節線修復圖 1.....	27
圖 4.2 邊界節線修復圖 2.....	27
圖 4.3 節點連通時間更新示意圖.....	28
圖 4.4 待修復網路圖範例.....	29
圖 4.5 邊界貪婪式演算法排程範例.....	30
圖 4.6 需求等待修復圖 1.....	31
圖 4.7 需求等待修復圖 2.....	31
圖 4.8 需求等待修復圖 3.....	31
圖 4.9 需求等待修復圖 4.....	31
圖 4.10 需求等待修復圖 5.....	31
圖 4.11 需求等待修復圖 6.....	32
圖 4.12 NCTA 欲修復網路圖 .....	33
圖 4.13 NCTA 步驟 1 圖 .....	33
圖 4.14 NCTA 步驟 2 圖 .....	33
圖 4.15 NCTA 步驟 3 圖 .....	34
圖 4.16 NCTA 步驟 4 圖 .....	34

圖 4.17 NCTA 步驟 5 圖 .....	34
圖 4.18 NCTA 步驟 6 圖 .....	34
圖 4.19 NCTA 步驟 7 圖 .....	35
圖 4.20 NCTA 步驟 8 圖 .....	35
圖 4.21 NCTA 步驟 9 圖 .....	35
圖 4.22 NCTA 步驟 10 圖 .....	35
圖 4.23 NCTA 步驟 11 圖 .....	35
圖 4.24 NCTA 步驟 12 圖 .....	35
圖 4.25 NCTA 步驟完成圖 .....	36
圖 4.26 染色體表示圖 .....	37
圖 4.27 選擇父母及分割點 .....	38
圖 4.28 進行複製與交換 .....	39
圖 4.29 最後小孩的結果 .....	39
圖 4.30 突變策略 .....	40
圖 5.1 GOOGLE CHART 範例資料 .....	44
圖 5.2 GOOGLE CHART 視覺化範例截圖 .....	44
圖 5.3 <i>Treem1</i> 各類型限制目標值分析圖 .....	45
圖 5.4 <i>Treem1</i> 各類型限制時間分析圖 .....	45
圖 5.5 <i>Generals1</i> 各類型限制時間分析圖 .....	46
圖 5.6 <i>Generalm1</i> 各類型限制時間 .....	46
圖 5.7 <i>Treem1</i> 測資中 $N_0$ 分析圖 .....	46
圖 5.8 <i>Treem1</i> 測資中 $N_1$ 分析圖 .....	46
圖 5.9 <i>Treem1</i> 測資中 $G_0$ 分析圖 .....	46
圖 5.10 <i>Treem1</i> 測資中 $G_1$ 分析圖 .....	46
圖 5.11 貪婪式演算法比較 1 .....	48

圖 5.12 貪婪式演算法比較 2 .....	48
圖 5.13 貪婪式演算法比較 3 .....	48
圖 5.14 貪婪式演算法比較 4 .....	48
圖 5.15 不同權重計算表 .....	49
圖 6.1 加入填補類型工作隊.....	53



## 表目錄

表 2-1:網路修復相關文獻與本研究之比較.....	10
表 4-1 節線權重表 .....	28
表 4-2 瓦斯管線權重計算表 .....	30
表 5-1 測試網路結構資訊 .....	43
表 5-2 有效不等式與加入初始解之測資情境代號表 .....	45
表 5-3 <i>Trees1</i> 、 <i>Treem1</i> 有效不等式與加入初始解測試表 .....	47
表 5-4 <i>Generals1</i> 、 <i>Generalm1</i> 有效不等式與加入初始解測試表 .....	47
表 5-5 貪婪式演算法符號表 .....	48
表 5-6 貪婪式演算法測試比較 .....	48
表 5-7 不同權重所得 GAP.....	50
表 5-8 基因演算法代號.....	50
表 5-9 基因演算法測試 1.....	50

# 第1章 緒論

## 1.1 研究背景與動機

台灣有八大類地下管線：電信管線、電力管線、自來水管線、下水道管線、瓦斯管線、水利設施管線、輸油管線、以及綜合管線。這些地下管線大都埋在道路下面，管線修復可能必須調查鄰近建築物結構、鄰近地下構造物的設施位置及構造形式、基地底下是否含有障礙物等。當馬路上某一段區域受到地震或爆炸等破壞時，很可能會造成多種類型地下管線同時毀損。譬如 2014 年的高雄氣爆事件，造成一心路上的天然氣管線、水管、瓦斯管等同時毀損。

這些毀損的地下管線在其被修復之前必須先在該處（節線，arc）進行開挖作業，一旦開挖之後，該節線之地上交通隨即開始受到影響，直至該節線上的最後一組毀損管線被完成修復之後，該處之交通才能恢復正常。我們假設每段節線上的各類管線物資需求量皆為已知，因此所有的毀損管線都必須被修復（亦即滿足所有需求量），然而已完成修復的管線只是具備可以輸送物資（即「連通」）的能力，至於當下該類管線是否已可立即輸送物資（即「連通外界」），則取決於其兩端點之中是否在當下至少有一端已連通外界。因此，毀損管線的「連通性」、以及管線修復的決策都與管線的「網路連通性」息息相關。這個管線的「網路連通性」也使得本問題比傳統的修復排程更加複雜，因為管線的選擇修復決策還要同時考慮到其與附近管線的連通性。

此外，不同類型的毀損管線通常各自分屬不同的公司或單位負責修復，除了必須先開挖才能開始修復之外，基本上可將各類管線的網路視為各自在獨立的不同網路層。由於單一類（同網路層）的管線維修排程已是 NP-hard 的問題，本研究同時處理多類（多網路層）管線維修排程當然更加困難；而且我們還首度將同一節線內因為管線尚未全修復完，該節線上的道路尚無法回填復原，進而引發的交通不便性列入排程考量。由於這類民生物資的可獲性以及交通的便利性對災民的生活品質影響甚大，是災後復建管理的重要指標之一。然而要精算出各毀損管線的開挖、修復時程，必須同時考量

挖掘、修復工作隊的能力、效率及其排程順序對管線連通性影響等諸多因素，是個極為困難的組合最佳化問題。本研究希望能藉由作業研究、數學規劃的理論，針對本排程問題推導出整數規劃模式，以求解理論上的最佳排程；然而求解最佳排程，在實務上經常耗時甚久，因此本研究亦設計貪婪演算法、基因演算法等數種啟發式演算法，以在較短的時間內計算出品質不錯的可行排程解。

## 1.2 研究目的

本研究將探討因人為或天災因素致使地下多類管線同時遭受大面積毀損時，如何有效地整合開挖與各類管線的維修排程，以使災民等待可由管線獲取物資、以及因開挖修復而影響交通的時間越短越好。

如果將整個管線網路修復問題視為一個修復專案，修復工作隊即為有限的救災資源，同一時刻能被指派的人力資源有其上限，則本研究可被視為一個具有網路架構的「具資源限制之專案排程問題」(Resource Constrained Project Scheduling Problem, RCPSP)。傳統 RCPSP 中的個別作業間之先後順序 (Precedence Relation) 通常已被定義清楚，然而本研究僅定義「挖掘」作業必須早於「修復」作業，並將修復完畢的「回填」作業時間整併於修復作業中 (因此可忽略回填作業)，然而對同一區域 (節線，arc) 內的不同類型管線之修復作業並無明確的先後順序要求；所有的毀損管線皆可被視為無向節線，可自其任一端點開始修復、以及連通輸送物資。

以圖 1.1 的水管網路為例，假設所有節線皆毀損，有五組挖掘工作隊、兩組水管修復工作隊、兩組瓦斯管修復工作隊；各毀損節線必須先挖掘才能開始修復，而在同條節線最後一組執行修復的工作隊亦負責回填作業；針對某一毀損節線( $i,j$ )而言， $D_{ij}^W$ 、 $D_{ij}^G$ 、 $P_{ij}^D$ 、 $P_{ij}^W$ 、 $P_{ij}^G$  分別代表該段節線的水需求量、瓦斯需求量、挖掘時間、修復水管時間以及修復瓦斯管時間。



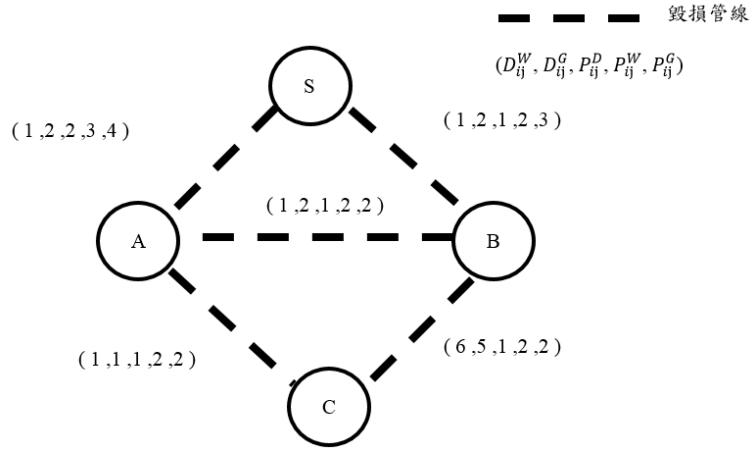


圖 1.1 欲修復網路圖 1

圖 1.2 的修復方式，在時刻 1 之前，只有節線(S,A)、(S,B)、(A,B)、(A,C)、(B,C)被挖掘完畢，但仍未被修復，所有節線尚未修復完畢，無法滿足所有需求，計算此時之需求總等待時間必須將所有節線上的單位需求加總起來，在時刻 0 至時刻 1 這段期間，水的等待需求單位為  $T_{SA}^W + T_{SB}^W + T_{AB}^W + T_{AC}^W + T_{BC}^W = 1 + 1 + 1 + 1 + 6 = 10$ ，瓦斯的等待需求單位為  $T_{SA}^G + T_{SB}^G + T_{AB}^G + T_{AC}^G + T_{BC}^G = 2 + 2 + 2 + 1 + 5 = 12$ 。時刻 3 之後，因節線(S,B)已被  $W_2$  修復且與供給點 S 連通，所以  $D_{SB}^W$  的 1 單位需求已被滿足。依此類推，時刻 5、6、7 之後會分別滿足  $D_{SA}^W + D_{AB}^W$ 、 $D_{SB}^G$ 、 $D_{AC}^W + D_{BC}^W$  單位的水需求，如此可得水需求之總等待時間為  $T_{SA} + T_{SB} + T_{AB} + T_{AC} + T_{BC} = (5*1 + 9*2) + (3*1 + 6*2) + (5*1 + 8*2) + (7*1 + 10*1) + (7*6 + 11*5) = 173$ 。而此排程之交通阻塞時間為每條節線從開始挖掘到所有管修復完成的時間加總，亦即  $T_{SA} + T_{SB} + T_{AB} + T_{AC} + T_{BC} = 9 + 6 + 8 + 10 + 11 = 44$ 。同理針對圖 1.3 的修復方式，雖然其最後的需求總等待時間與圖 1.2 同為 173，然而其交通阻塞時間僅為  $T_{SA} + T_{SB} + T_{AB} + T_{AC} + T_{BC} = 9 + 6 + 6 + 6 + 7 = 34$ ；由本例可知，雖然圖 1.2 的排程會早點開挖完成所有路段，卻囿於維修工作隊數較少，導致部分早已開挖完成的節線必須閒置一段時間才能被修復，也造成該排程之交通阻塞的時間比圖 1.3 的排程更長。因此一般實務上儘量不讓工作隊閒置的排程法則，在本問題不見得會得到較好的排程效果。

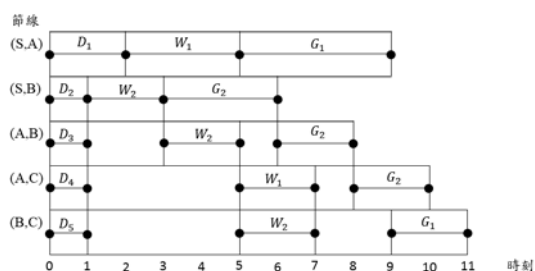


圖 1.2:第一種修復順序排程圖

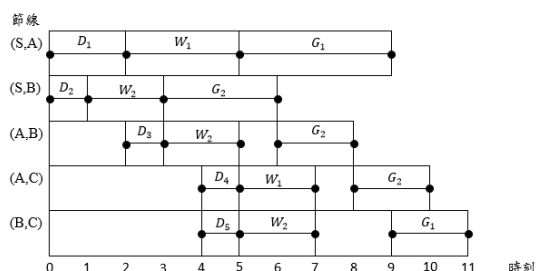


圖 1.3:第二種修復順序排程圖

### 1.3 研究假設與範圍

本研究的研究假設與範圍如下：

1. 假設各類地下管線之毀損與連通狀況、各工作隊針對各毀損管線之挖掘及修復時間、各管線上的單位時間需求及交通影響量等資訊皆為已知。
2. 假設必須修復所有的毀損管線，每管線僅能被一個工作隊處理（挖掘或修復），工作隊一旦開始處理不得中途停工。
3. 欲求解管線搶修排程中，如何指揮不同類型多工作隊，同時進行多類型管線修復工作，使得居民等待不同資源的總時間以及因修復所造成的交通不便性越小越好，如此才能儘快恢復正常生活與交通流暢。
4. 將設計數學規劃模式以及排程演算法求解。

### 1.4 論文架構

本論文架構如下:第二章為回顧網路修復問題與具資源限制下之專案排程問題(RCPSP)的相關文獻；第三章闡述本研究提出之整數規劃模式；第四章探討排程演算法的設計方式，提出貪婪式演算法與基因演算法；第五章為數值測試與分析，分別針對整數規劃與貪婪式演算法與基因演算法進行數值比較；第六章總結本論文並建議數個未來研究主題。

## 第2章 文獻回顧

本章將先介紹網路修復的相關文獻，說明本研究探討議題與相關性，在網路毀損的修復排程中，如何以工作隊數有限的資源下，進行毀損節線與節點的搶修，使整體的修復時間最短；讓原本中斷的資源運輸，在毀損網路修復後，能儘快地被運送到需求端。在相關文獻中，「整合網路設計與排程修復問題」(Integrated Network Design and Scheduling Problem, INDS)的過往研究大都假設需求量發生在節點而非節線之上，且較少探討不同類型工作隊執行不同任務(挖掘、修復)的優先順序關係；反之，以模擬方法為主的網路修復相關文獻較常探討不同類型工作隊的優先順序關係；由於本研究的整數規劃模式是以 RCPSP 為基礎而設計，因此我們亦將回顧 RCPSP 相關文獻。

### 2.1 網路修復相關文獻

根據 Heath et al. (2016)的文獻介紹，網路修復問題大致可分為二大類：

1. **排程問題、任務指派**：大部份的文獻都在研究此問題，如何在有限的資源下，指派工作隊，進行毀損網路的修復，此類問題著重於問題發生後，如何在有限資源下進行網路的修復。
2. **改善網路回復力(Improving Network Resilience)**：近幾年探討在緊急事件發生後，預測將可能毀損區域降到最低，降低破壞資源供給的嚴重程度，分為長程與短程計畫。此類問題著重於預防，例如當管線被攻擊，預測有哪些地區會毀損的比較嚴重，並在其附近事先安置備用管線，以使攻擊造成主管線毀損後，能立即啟用備用管線。

本研究可被歸類於第一類問題，在網路毀損後，如何在有限資源下，指派工作隊先開挖、再執行多類管線的修復，與傳統的第一類文獻主要有以下兩處不同：(1) 挖掘或修復等不同類型的工作隊有明顯優先順序關係（先挖後修）；(2)針對修復工作隊

而言，依管線類別（譬如水、瓦斯、電力等）不同，個別有其對應技能或器具的不同修復工作隊，甚至針對同技能或器具的工作隊，亦可能有不同的工作效率。

以往的網路修復問題在任務指派方式可分「即時排程」(Online Scheduling)與「非即時排程」(Offline Scheduling)兩類，其差別在於後者已經事先知道所有網路與排程的相關資訊，而前者則僅知道當下為止的資訊，對於尚未發生的事件無資訊可參考。兩者的進一步說明如下：

**(1)任務指派中的即時排程問題：**Nurre and Sharkey (2014, 2015)提出「整合網路設計與排程問題」(INDS)，求解重建或修復網路的排程問題，更詳細的 INDS 架構說明可參考 Nurre and Sharkey (2014)，該研究考慮三種類型的工作隊：(a)只能維修(Installation)工作隊、(b)只能清除(Supplementary)工作隊(清除障礙物，必須先完成清除工作才能開始維修)、(c)多功能工作隊(既能清除又能維修)。一條毀損的節線，經過清除與維修後，變成一條沒有毀損的節線；毀損的節線必須在特定的「釋放時刻」(Release Time)後才可以開始被修理；而「彈性的釋放時刻」代表清理工作隊可在比原本已排定維修的時刻更早完成清理，所以可以提早開始維修節線。即時排程問題並不知道毀損節線可以開始修理的釋放時刻，藉由彈性的釋放時間讓工作隊可以提早修理節線。該作者先將未知的路徑透過設計出的派遣法則，找出最適合的路徑，將最適合的路徑求解出來，再將求解出的路徑當作已知，透過建模方式進行任務的指配。

**(2)任務指派中的非及時排程問題：**Averbakha (2012)論文目標是找到最少的總節點等待修復時間或是最快修完最後一個節點(即最小化 makespan)。該研究提到每個工作隊的維修路徑速度是固定的，假如多個工作隊一起維修路徑，此段維修路徑的速度是各工作隊的加總，舉例來說，假如 A 工作隊修復路徑的時間是每小時 0.1 公里，B 工作隊修復路徑的時間是每小時 0.2 公里，C 工作隊修復路徑的時間是每小時 0.3 公里，則工作隊 A、B、C 合作一起修同一段路徑的速度是每小時 0.6 公里。此文獻中將「緊急路段修復問題」(Emergency Path Restoration Problems)與「路徑選擇排程問題」(Routing-scheduling Problems)做區別，其中，前者能重複經過可服務的地區，而後者

則不能重複經過。假如每個工作隊的修復速度相同，該研究證明本問題可以在多項式時間內找到全域最佳解；反之工作隊的修復速度不同之問題，可被證明為 NP-hard。本研究與該研究的主要差別在於我們假設需求量發生於節線，而該研究的需求則在節點上。

Averbakha and Pereira (2012)考慮合作修復，探討最小化修復完成時刻(Makespan)的網路修復排程問題，認為若要最小化修復完成時刻，必須讓所有工作隊一起合作修復，如此即可將本問題由多工作隊簡化為單工作隊（因為每條節線都是大家一起合作修復）。該研究提出三種混合整數線性規劃模式：(a)使用單商品流(Single-commodity Flow)去尋找最小展開樹問題(Minimum Spanning Tree Problem)，假如每一個節點都有收到一單位的流量，就能確定與資源的源頭連通；(b)只使用一種二元變數，建構出最小化修復時刻，選一個已修復的節線往外連出，再試圖確保避免產生獨立子迴圈(Isolated Directed Cycles)；(c)將每個點加入權重(譬如需求量或重要程度)，最小化加權總等待修復時刻，使用 Branch-and-Bound 搭配不同法則找更精確的解之上下界，其中使用累積展開樹放鬆法(Cumulative Spanning Tree Relaxation)是最有效率。

Nurre et al. (2012) 求解網路流量重建的問題，目標為在修復完成前能最大化累積之總加權流量。該研究建立單一商品流量的整數規劃模式，針對多工作隊對於選擇節點與節線的排程問題，主要有三種變數(i)網路流變數(Network Flow Variables)(ii)網路設計變數(Network Design Variables)(iii)排程變數(Scheduling Variables)，透過節點的需求是否被滿足而確認是否與供給點連通，加入加權流量的觀點以修改 Smith (1956)經典權重最短處理時間(Classic Weighted Shortest Processing Time)的派遣法則，設計 (i)最短處理時間路徑(ii)流量覆蓋(iii)節線修復完成時刻的保存等三類有效不等式，目標為在修復完成前能最大化累積之總加權流量，該研究亦使用單商品流量整數規劃模式來偵測網路連通性，而本研究則再將其衍生為多商品流量整數規劃模式。

Nurre and Sharkey (2014)證明數個 INDS 相關的問題皆為 NP-hard，並整理出 INDS 相關文獻的最佳化目標函式值大致有以下數類：加權的總等待修復完成時刻(Total

Weighted Recovery Time)、平均之修復完成時刻(Average Recovery Time)、為滿足需求的成本、總系統成本、節點的修復時刻(Recovery Time of Nodes)、節點在路徑網路的修復完成時刻(Recovery Time of Nodes in Path Network)、最大流量、最短路徑、最小展開樹、最小成本流量。目標式也分成不同類型，分別為 Cumulative、-Threshold、Multi-objective: Cumulative and Cost。該研究提出新的派遣法則，這法則主要是每次都挑出一條權重最大的最短路徑。該研究證明出單一種類型工作隊修復問題為 NP-hard。

Tabucchi et al. (2010)使用系統模擬的方式，模擬地震發生後，水供給系統的修復狀況，目標式為最小化總體等待修復時間，該研究分為五種工作隊：(a)水資源分布偵察隊，其工作為偵查所有管線、(b)水資源分布維修隊、(c)馬路幹線維修隊、(d)供水公司操作員，其工作偵查蓄水池和抽水機、(e)供水公司工人，其工作為偵查一般的站點。偵查隊必須先進行偵查毀損地區，維修隊才能開始維修。而地區又分為(a)毀損卻未被發現、(b)視察資源未被送到、(c)等待視查資源到達、(d)正在視察階段、(e)等待維修資源、(f)正在維修、(g)正在維修且等待另一種維修資源繼續維修、(h)等待重新規劃路徑資源、(i)正在進行重新規劃路徑操作、(j)所有毀損都修復等十種。該研究假設每種工作隊都有一個待修復對象清單，各對象皆有其優先權，優先權高的(例如醫院)先修復；工作隊有輪班機制，以震央最近的地方最先派偵查隊偵查，離水源最近的地方先修復毀損管線，在工作隊有限的情況下必須最小化總等待修復完成時間，該研究是以系統模擬的方法模擬不同類型工作隊優先順序。

Heath et al. (2016)探討改善網路回復力的問題，經由預測即將發生的毀損情境，提早指派工作隊前往待命或佈署備用資源，以讓受毀損的資源供給區域越小越好。在預測出有哪些地區可能遭受攻擊或是天災而導致供給資源的管線毀損之後，事前短期的作法為針對單一事件，派工作隊提早前往將會毀損區域，使事件發生後能儘快搶修；而長期的作法則針對多個事件，在地底下掩埋備用管線，提供即時的電力或水力等。該研究對於事前的預測分為兩階段，第一階段為預測會受到影響的區域，第二階段為針對第一階段所挑選出會受影響的區域，進 GSP (Group Screening Procedure)演算法搭

配統計方法，使這些區域快速回復到最接近受影響前的最佳狀況（不見得需要完全回復為原狀）；該研究亦使用單商品流量整數規劃模式以偵測連通性。

Huang (2015)探討災後緊急救援物流管理之網路重建問題，假設需求在  $n-1$  個節點上，物資的輸送必須從單一供給點開始，因此工作隊只能藉由已修復完成的節線來移動並選擇可準備被修復的毀損節線。毀損的節線必須經過修復才能輸送物資，為了讓物資以最快時間送達需求點，只需修復部分毀損節線即可。該研究證明此問題之解結構必為一展開樹，若目標為最小化所有節點的最晚修復完成時刻的話，單一與  $n-1$  組工作效能相同的工作隊之最佳解，將剛好會對應到「最小展開樹」及「最短路徑樹」。而針對一般介於 1 與  $n-1$  間的  $K$  組工作隊的修復排程問題，該研究提出一個整數規劃模式求解，另外也設計分段式啟發演算法(Sequential Segment Heuristic, SSH)與粒子群最佳化演算法(Particle Swarm Optimization, PSO)求近似最佳解，其中 PSO 的求解品質不好，反之 SSH 大多能快速求得很好的解。

Lin (2016) 探討資源需求分佈在節線（非節點）上的網路修復排程問題，與以往大都針對需求分佈在節點上的災害修復文獻有所不同；由於假設地下每條管線皆有其對應的需求量，因此必須修復所有的毀損管線，且假設可隨時直接於各毀損管線開挖與修復，此與 Huang (2014)僅需修復部分節線、且其可開始修復與否倚賴該節線端點是否已連通供給點的假設有所不同。該研究以 Nurre and Sharkey (2014)的 NP-hard 證明為基礎，證明該研究問題亦為 NP-hard。該研究針對單一工作隊的情境，提出一整數規劃模式；再針對多組工作隊的情境，提出基於 Brand-and-Cut 與多商品網路流量等兩種整數規劃模式，其模式亦可被延伸處理工作隊效能不同、或合作等情境之排程。為能更快速求解，該研究亦修改 Huang (2015)之分段式啟發演算法(SSH)，並設計兩類貪婪式演算法、基因演算法等數種啟發式演算法。其測試結果顯示 SSH 仍有最佳的效能表現。

表 2-1 整理了本研究與先前相關文獻的異同比較。本研究以 Lin (2016)為基礎，同樣是處理需求分佈在節線、必須挖掘與修復所有毀損節線，但 Lin (2016)僅能處理

單一類的管線修復且沒有將開挖作業列入排程考量；反之，本研究可同時處理先開挖後修復的先後作業排程，且將多類地下管線分別由不同類工作隊負責、且管線開挖後對交通不便的影響也一併列入排程考量，這些都是本研究比文獻更貼近現實生活之處。

文獻	多工作隊	整數規劃模型	不同類型工作先後順序	多層種類管線	需求位置	網路結構
Tabucchi et al. (2010)	✓		✓		節點	General
Averbakha (2012)	✓				節點	Path
Averbakha and Pereira (2012)		✓			節點	General
Nurre et al. (2012)	✓	✓			節點	General
Nurre and Sharkey (2014)	✓	✓			節點	General
Nurre and Sharkey (2015)	✓	✓	✓		節點	General
Huang(2015)	✓	✓			節點	Grid、Tree、Path
Heath et al.(2016)	✓	✓			節點	General
Lin(2016)	✓	✓			節線	General、Tree
本研究	✓	✓	✓	✓	節線	General、Tree

表 2-1:網路修復相關文獻與本研究之比較

由於本研究探討的排程可被視為一個具有網路架構的特殊 RCPSP，因此在下一小節將從 RCPSP 角度回顧相關文獻。

## 2.2 資源限制下之專案排程問題 (RCPSP) 相關文獻

現實生活中的各項工作任務均可被視為專案，專案管理通常會將其必須進行的子任務展開，依子任務相互間的先後順序關係可畫出 PERT (Program



Evaluation and Review Technique) 圖 (如圖 2.1 所示)。而專案在實際執行時幾乎都必須考量有限的可用資源限制，但 PERT 圖未能同時考量各子任務執行的資源需求量，因此如何將資源耗用列入排程的 RCPSP 比 PERT 圖的排程更符合實際專案規劃的需要，而 RCPSP 已被證明是個 NP-hard 問題(Blazewicz et al., 1983)。

以圖 2.1: PERT 圖為例，索引值為 0 的節點為虛擬起點，索引值為 6 的節點為虛擬的訖點，整個專案從虛擬的起點開始到虛擬的訖點結束，要進行作業 5 前必須先進行作業 3 和作業 2，但作業 3 和作業 2 並無明顯的先後順序要求；另一方面必須先進行作業 1 才能進行作業 4；待作業 4 和作業 5 都完成後，整個專案才會完成。

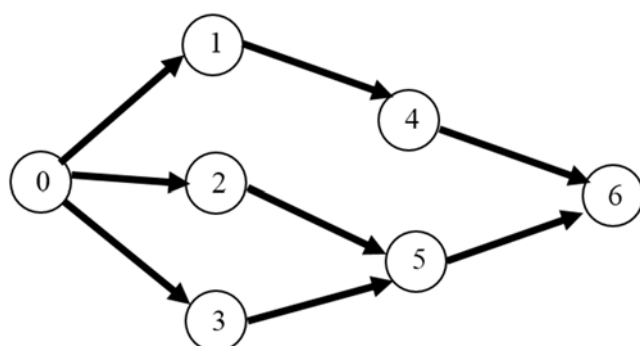


圖 2.1: PERT 圖

RCPSP 可將專案轉化為一 PERT 圖，假設一專案有  $A$  個作業(Activity)，索引值  $a$  從 1 到  $A$ ，每個作業  $a$  有其處理時間  $p_a$  (Processing Time) 與先行作業集合 (Predecessor Activities)  $P_a$ ，先行作業集合是指假設必須先處理完作業  $C$ 、作業  $D$ 、作業  $F$ ，最後才能處理作業  $A$ ，則  $A$  的先行作業集合  $P_a = \{C, D, F\}$ ，在專案中有  $K$  種資源，假設每單位時間 (或每單期) 內，資源  $k$  有其資源可獲量上限  $R_k$ ，而每一作業  $a$  對每一資源  $k$  有需求量  $r_{jk}$ 。

MRCPSP 是由 RCPSP 延伸而來，其開頭的 M 代表多種模式(mode)。一般的 RCPSP 中通常假設是單模式作業，亦即各作業在任何時刻之執行時間及資源需求量皆為固定且已知。在 MRCPSP 中，各作業在任何時刻之執行時間及資源需求量是由其選擇之

作業方式(mode)決定，且各作業方式之執行時間及資源需求量固定且已知。專案中每個作業開始進行後，都不可以被中斷(non-preemptive)，資源又可以分為可更新(renewable)資源(譬如廠房中的機械設備)，與不可更新(non-renewable)資源(譬如製造產品所需的原物料)。以下將針對 RCPSP 常見的整數規劃模式進行說明：

## 2.2.1 符號定義

### 參數

$N$	節點之集合，包含一虛擬起始節點 (Dummy Start Node) 及虛擬完成節點 (Dummy Finish Node)，各別編號 0 及 $n+1$
$A$	節線集合，任一節線表示兩作業間的優先順序，節線 $(i, j)$ 表示作業 $i$ 在作業 $j$ 之前
$R^r$	可更新資源集合
$R^n$	不可更新資源集合
$a_k^r$	可更新資源 $k$ 在每一期（或單位時間）的資源上限量
$a_l^n$	不可更新資源 $l$ 在整個專案下的總資源上限量
$M_i$	作業 $i$ 之作業方式(mode)集合
$d_{i,m}$	作業 $i$ 在其作業方式 $m$ 下之所需之作業時間
$r_{i,m,k}^r$	作業 $i$ 在其作業方式 $m$ 下，對可更新資源 $k$ 之需求量
$r_{i,m,l}^n$	作業 $i$ 在其作業方式 $m$ 下，對不可更新資源 $l$ 之需求量
$T$	專案完工時刻之上限
$es_i$	作業 $i$ 最早可處理時刻
$ls_i$	作業 $i$ 最晚可處理時刻

### 決策變數

$x_{i,m,t}$  作業  $i$  在時刻  $t$  以作業方式  $m$  開始執行為 1。相反，作業  $i$  在時刻  $t$  以作業方式  $m$  沒有開始執行為 0

### 2.2.2 數學模式

Talbot (1982) 提出以下整數規劃模式，目標式為最小化完工時刻(Makespan)。

$$\text{Minimize } \sum_{t=es_{n+1}}^{ls_{n+1}} tx_{n+1,1,t} \quad (2.1)$$

#### 限制式

限制式(2.2)規範專案中各子任務（作業）的優先順序，PERT 圖中的任一節線  $(i, j)$ ，作業  $j$  必須等作業  $i$  完成才可以開始處理，所以作業  $j$  的開始處理時刻必須大於或等於作業  $i$  的完工時刻。

$$\sum_{m=1}^{M_i} \sum_{t=es_i}^{ls_i} (t + d_{i,m}) x_{i,m,t} \leq \sum_{m=1}^{M_j} tx_{j,m,t} \quad \forall (i, j) \in A \quad (2.2)$$

限制式(2.3)規範任一作業  $i$  只能使用某個 mode，且必須遵守其開工時窗限制。

$$\sum_{m=1}^{M_i} \sum_{t=es_i}^{ls_i} x_{i,m,t} = 1 \quad \forall i \in N \quad (2.3)$$

限制式(2.4)代表任一種可更新資源  $k$  在時刻  $t$  的使用總量不可超過其資源上限  $a_k^r$ 。

$$\sum_{i=1}^n \sum_{m=1}^{M_i} r_{i,m,k}^r \sum_{s=\max(t-d_{i,m}, es_i)}^{\min(t-1, ls_i)} x_{i,m,s} \leq a_k^r \quad \forall k \in R^r, t=1, \dots, T \quad (2.4)$$

限制式(2.5)代表任一種不可更新資源  $l$  在時刻  $t$  的使用總量，不可超過其資源上限  $a_l^n$ 。

$$\sum_{i=1}^n \sum_{m=1}^{M_i} r_{i,m,l}^n \sum_{t=es_i}^{ls_i} x_{i,m,t} \leq a_l^n \quad \forall l \in R^n, t=1, \dots, T \quad (2.5)$$

限制式(2.6)設定各作業模式在某時刻的開工排程決策變數為 0 或 1

$$x_{i,m,t} \in \{0, 1\} \quad \forall i \in N; m=1, \dots, M_i, t=1, \dots, T \quad (2.6)$$

## 2.3 小結

本章節回顧「整合網路設計與排程問題」(INDS)文獻，以及排程問題、任務指派、改善網路回復力、即時排程與非即時排程等文獻。其中，大部分文獻皆探討需求設在節點上，不能像 Lin (2016)與本研究而直接處理需求分佈在節線上的地下管線修復問題，本研究以 Lin (2016)為基礎，將其僅考慮單一類管線維修的基本模式再更進一步延伸成可處理多類管線修復以及多類工作隊（挖掘、修復）的排程問題。此外，本研究考慮各工作隊可逕行於各節線開始修復（即不見得需從供給點開始修復）、修復前必須先開挖、同節線上的多類管線可同時進行其修復作業、將開挖至修復完回填的這段期間於該節線的交通不便性列入排程目標式，以上更符合實際的排程考量都是本研究有別於傳統文獻之處；由於本研究為具有網路特性的特殊 RCPSP，因此我們也回顧 RCPSP 基本數學規劃模式，以其為基礎設計本問題的整數規劃模式(詳見下章)。

### 第3章 網路節線修復問題之整數規劃模式

本章將針對有多種類型(譬如水、瓦斯、電力等)地下管線，與具先後作業順序限制的不同類型工作隊（譬如挖掘、修復等）的修復問題，提出整數規劃模式 $P_{many}$ 。

#### 3.1 問題描述

假設有一連通網路(Connected Network)  $G=(N, A_d, A_c)$ ，其中  $N$  為網路內之所有節點(nodes)集合， $A_d$  為毀損的節線、 $A_c$  為沒有毀損的節線而所有的節線皆為無向節線。假設各節線的需求量、及其被各工作隊修復所需的修復時間皆已知，如果節線無法與供給點連通，則物資無法送達該節線；假設不同的種類的物資會用不同管線輸送，工作隊分為「挖掘」管線工作隊與「修復」管線工作隊，而修復管線工作隊又可再依據管線類別細分為個別特定管線的修復工作隊(例如修復瓦斯管線工作隊、修復水管管線工作隊等)。管線毀損時，必須先挖掘毀損管線所埋設路面，再進行管線修復；挖掘工作隊自成一類，可以挖掘任一類型管線；各類修復管線工作隊只能修該特定類型管線(e.g., 修復水管工作隊不能修瓦斯管線)，不同修復類型工作隊可同時在同一節線進行修復，然而一旦開始進行修復工作後將不得中途暫停，而必須將該管線修完；假設每段毀損節線皆必須被修完；先不考慮工作隊可以合作，因此一條節線的同種類管線只由一組工作隊修(其實本研究提出的數學模式亦可用多模式的方式來處理工作隊合作的情況)，目標是使總體需求的等待時間以及交通阻塞造成的影響越小越好，可見第 1.2 節的圖 1.1 範例，以及可能排程結果之甘特圖(圖 1.2、1.3)。

#### 3.2 問題假設

為適當簡化模式的發展與界定模式的使用限制，本研究有下列基本假設：

1. 假設網路為連通(connected)，即所有節點接連在一起，物資需求量分佈在節線而非節點上，只要管線經過的地方就會有需求量。

2. 假設已經事前精密定位出各類管線的位置分布，並已知各類管線的毀損分佈與毀損所需的修復時間、民眾對各類物資的需求量、各工作隊的工時等資訊。
3. 假設各管線不受地勢、溫度、氣候、等外在因素影響，一旦管線修復與供給點連通即可滿足此管線上的需求。
4. 假設資源的運輸時間遠小於管線修復時間，故忽略資源運輸時間。
5. 各工作隊不需回工作站補充能源，不用休息，工作不中斷。
6. 管線沒有修復先後順序，但是必須挖掘後才能修復，各工作隊可任選一已開挖好的管線來修復。
7. 忽略工作隊在不同節線間的移動時間（設定為零）。
8. 假設某段節線的最後一組維修管線的工作隊，在其維修完工後會立即填補完路面，且填補路面的時間為零。
9. 如圖 3.1，假設不同類型管線可以平行修復，多種類型管線毀損時，可以同時進行不同類型管線修復。



圖 3.1 管線配置圖(圖片來源:高雄市政府)

### 3.3 NP-Hard 說明

Nurre and Sharkey (2014)已證 $P_m|\sum SP|C_{max}$ -Threshold 為 NP-hard，Lin (2016)也說明該研究為 $P_m|\sum SP|C_{max}$ -Threshold 亦為 NP-hard，如果能將 Lin (2016)的 INDS 問題在

多項式時間內轉換為本研究問題，則代表本研究為 NP-hard。由於 Lin (2016) 只考慮單一類型管線與修復工作隊，是本研究多類管線與工作隊的特例，因此本研究問題亦為 NP-hard。

### 3.4 用多商品網路流(Multi-commodity Network Flow, MCF)之不同類型管線多工作隊整數規劃模式

本節使用 MCF 的建模方式，利用各節點的單位虛擬需求-1 是否可被供給點  $s$  滿足，以判斷各節點與供給點的連通性。以圖 3.2 為例，由  $s$  提供 10 單位，各節點的虛擬需求-1 如果被滿足，表示該節點可與  $s$  連通。依此原理，在求解過程中可依當下各節點的虛擬需求被滿足狀態，反推出當下哪些節點已可連通  $s$ ，以此判斷哪些節線是至少一端連通  $s$  的毀損邊界節線(Boundary Arc)、哪些是兩端皆為連通  $s$  的毀損內部節線(Inside Arc)。

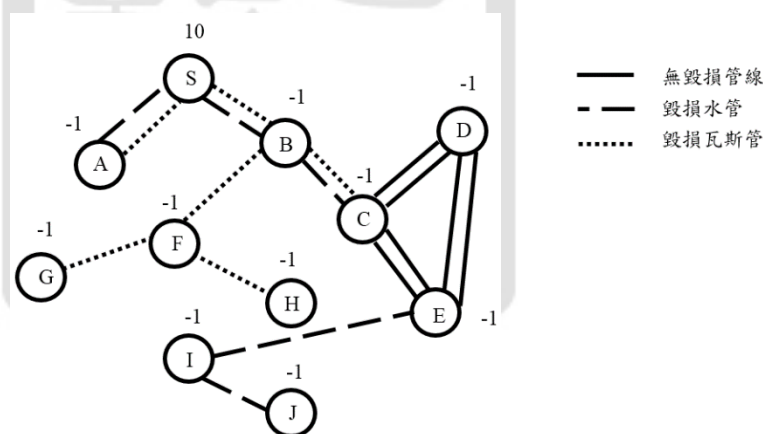


圖 3.2: 使用 MCF 之多工作隊整數規劃示意圖

本節將考慮只挖一次且回填一次的情況下，以圖 3.3 為例，在多種不同類型管線下(例如水管、瓦斯管、輸油管)，不同類型工作隊的修復排程問題。在工作隊有限的資源下，可能造成某些尚未修復完成的節線被閒置(如圖 3.3 在時刻 2 到時刻 3、時刻 8 到時刻 9)，直至該節線上所有類型的管線完成修復才會回填(如圖 3.3 的時刻 12)。

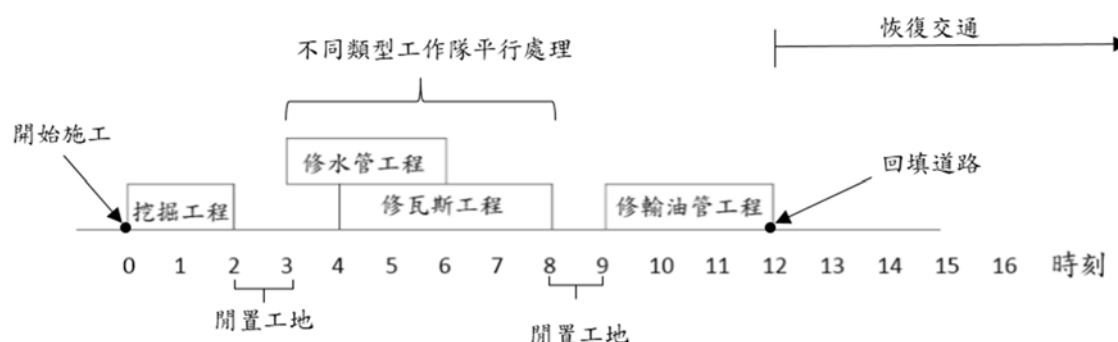


圖 3.3:使用 MCF 之多工作隊時間示意圖

### 3.4.1 符號定義

#### 集合

$N$	所有節點集合(其中唯一的供給點記作 $s$ , $s \in N$ )
$A$	毀損的節線集合
$A^*$	未毀損的節線集合
$A(i)$	與 $i$ 相連的毀損節線集合
$K^a$	節線 $a$ 上的管線種類( $a \in A \cup A^*$ )
$K$	所有管線種類( $K := \bigcup_{a \in A \cup A^*} K^a$ )
$R^k$	修復管線種類 $k$ 的工作隊集合( $R^k \cap R^{k'} = \emptyset, \forall k, k' \in K, k \neq k'$ )
$R^D$	挖掘工作隊集合( $R^D \cap R^k = \emptyset, \forall k \in K$ )

註:  $i, j \in N$  ,  $i \neq j$  ,  $(i, j) = a \in A \cup A^*$  , 定義符號  $\bar{a} = (j, i)$  ,  $tail(a) = i$  ,  $head(a) = j$  , 其中  $a, \bar{a}$  代表同節線但不同方向。

#### 參數

$T$	總專案結束時刻上限( $t = 0, 1, \dots, T$ )
$p_a^k$	修復節線 $a$ 上管線 $k$ 所需之時間( $k \in K^a$ )
$\bar{p}_a$	節線 $a$ 上各毀損管線中所需最長的修復時間 (亦即 $\bar{p}_a := \max_{k \in K^a} p_a^k$ )



$p_a^D$	挖掘節線 $a$ 所需之時間
$d_a^k$	節線 $a$ 上資源 $k$ 的單位時間需求量( $k \in K^a$ )
$c_a$	節線 $a$ 上的單位時間交通影響程度
$IND_i$	節點 $i$ 的內向臨邊節線個數( $IND_i = \left\  \left\{ (j, i) \in A \cup A^* \mid i \in N \right\} \right\ $ )
$M$	充分大的常數

### 決策變數

$s_a^D$	節線 $a$ 的開始挖掘時刻
$s_a^k$	節線 $a$ 的管線 $k$ 之開始修復時刻( $k \in K^a$ )
$\delta_{atr}^{DS}$	工作隊 $r$ 是否在時刻 $t$ 開始挖掘節線 $a$ ，已開始挖掘則為 1;反之為 0
$\delta_{atr}^{kS}$	工作隊 $r$ 是否在時刻 $t$ 開始修復節線 $a$ 上的管線 $k$ ，已開始修復則為 1;反之為 0 ( $k \in K^a$ )
$\delta_{atr}^{DF}$	工作隊 $r$ 是否在時刻 $t$ 挖掘完節線 $a$ ，已挖掘完則為 1;反之為 0
$\delta_{atr}^{kF}$	工作隊 $r$ 是否在時刻 $t$ 修復完節線 $a$ 上的管線 $k$ ，已修復完則為 1;反之為 0( $k \in K^a$ )
$y_{at}^k$	時刻 $t$ 節線 $a$ 上的管線 $k$ 是否與外界連通，已連通則為 1;反之為 0 ( $k \in K^a$ )
$z_{at}^k$	時刻 $t$ 節線 $a \in A$ 上的管線 $k$ 是否已被修復，已修復則為 1;反之為 0(針對 $a \in A^*$ ，一律設定 $z_{at}=1$ ， $k \in K^a$ )
$x_{at}^k$	時刻 $t$ 節線 $a$ 上的管線 $k$ 的流量大小( $k \in K^a$ )
$v_{it}^k$	時刻 $t$ 節點 $i$ 上的管線 $k$ 是否已與外界(即供給點)連通，已連通則為 1;反

之為 0(針對  $i = s$ ，一律設定  $v_{it}^k = 1$ ， $k \in K^a$ )

$F_a^k$  節線  $a$  的管線  $k$  之結束修復時刻

$f_a$  節線  $a$  之所有管線結束修復時刻

$w_a^k$  節線  $a$  的管線  $k$  是否為最晚結束修理完畢的管線，是最晚結束則為 1;反之為 0( $w_a^k := 1 \Leftrightarrow k = \arg \max_{k' \in K^a} F_a^{k'}$ )

### 3.4.2 整數規劃模式

在管線的挖掘與修復過程中會造成道路縮減與交通阻塞，因此本研究將目標設定為希望在時刻  $T$  內全部需求之總等待時間(3.4.1 式之第一項)以及引發之交通阻塞時間成本(3.4.1 式之第二項)越短越好。

$$\text{Min} \sum_{t=0}^T \sum_{\substack{a \in A \cup A^+ \\ \text{tail}(a) < \text{head}(a)}} \sum_{k \in K^a} d_a^k (1 - y_{at}^k - y_{at}^k) + \sum_{a \in A} c_a (f_a - S_a^D) \quad (3.4.1)$$

限制式(3.4.2)定義各節線  $a$  的開始挖掘時刻。

$$S_a^D = \sum_{t=0}^{T - \bar{p}_a - p_a^D} t \sum_{r \in R^D} \delta_{atr}^{DS} \quad \forall a \in A \quad (3.4.2)$$

限制式(3.4.3)定義各節線  $a$  的開始修復時刻。

$$S_a^k = \sum_{t=p_a^D}^{T - p_a^k} t \sum_{r \in R^k} \delta_{atr}^{kS} \quad \forall k \in K^a, \forall a \in A \quad (3.4.3)$$

限制式(3.4.4)規範各節線必須先開挖才能進行修復。

$$\sum_{t=0}^{T - \bar{p}_a - p_a^D} (t + p_a^D) \sum_{r \in R^D} \delta_{atr}^{DS} \leq S_a^k \quad \forall k \in K^a, \forall a \in A \quad (3.4.4)$$

限制式(3.4.5)定義整體完工時刻上限  $T$  必須在各節線的結束修復時刻之後。

$$\sum_{t=p_a^D}^{T-p_a^k} (t + p_a^k) \sum_{r \in R^k} \delta_{atr}^{kS} \leq T \quad \forall k \in K^a, \forall a \in A \quad (3.4.5)$$

限制式(3.4.6)規範各節線只能在某時刻被某工作隊以某方向挖掘一次。

$$\sum_{t=0}^T \sum_{r \in R^D} (\delta_{atr}^{DS} + \delta_{atr}^{DS}) = 1 \quad \forall a \in A \quad (3.4.6)$$

限制式(3.4.7)規範各節線之某毀損管線只能在某時刻被某工作隊以某方向修復。

$$\sum_{t=0}^T \sum_{r \in R^k} (\delta_{atr}^{kS} + \delta_{atr}^{kS}) = 1 \quad \forall k \in K^a, \forall a \in A \quad (3.4.7)$$

限制式(3.4.8)規範任一時刻與供給點  $s$  連通的節點個數上限。

$$\sum_{\substack{a \in A \cup A^* \\ \text{tail}(a)=s}} x_{at}^k - \sum_{\substack{a \in A \cup A^* \\ \text{head}(a)=s}} x_{at}^k \leq |N| - 1 \quad \forall t = 0, \dots, T, \forall k \in K^a \quad (3.4.8)$$

限制式(3.4.9)定義任一時刻每個節點  $i$  是否與供給點  $s$  連通。

$$\sum_{\substack{a \in A \cup A^* \\ \text{tail}(a)=i}} x_{at}^k - \sum_{\substack{a \in A \cup A^* \\ \text{head}(a)=i}} x_{at}^k \leq -v_{ut}^k \quad \forall t = 0, \dots, T, \forall i \in N, i \neq s, \forall k \in K^a \quad (3.4.9)$$

限制式(3.4.10)規範各節線須修復完才能有流量流經該節線。

$$0 \leq x_{at}^k \leq (|N| - 1) z_{at}^k \quad \forall t = 0, \dots, T, \forall k \in K^a, \forall a \in A \quad (3.4.10)$$

限制式(3.4.11)設定節線的修復方向  $((i, j) \text{ or } (j, i))$ 。

$$y_{at}^k + y_{at}^k \leq 1 \quad \forall t = 0, \dots, T, \forall k \in K^a, \forall a \in A \quad (3.4.11)$$

限制式(3.4.12) (3.4.13) (3.4.14)設定節線  $a$  的連通條件：該節線必須修復且其  $tail$  節

點必須連通，節線  $a$  才能連通。亦即  $(y_{at}, z_{at}, v_{tail(a)t})$  滿足以下四種可能性

$(1, 1, 1), (0, 0, 1), (0, 1, 0), (0, 0, 0)$ 。

$$y_{at}^k \leq z_{at}^k \quad \forall t = 0, \dots, T, \forall k \in K^a, \forall a \in A \cup A^* \quad (3.4.12)$$

$$y_{at}^k \leq v_{tail(a)t}^k \quad \forall t = 0, \dots, T, \forall k \in K^a, \forall a \in A \cup A^* \quad (3.4.13)$$

$$v_{tail(a)t}^k + z_{at}^k - 1 \leq y_{at}^k \quad \forall t=0, \dots, T, \forall k \in K^a, \forall a \in A \cup A^* \quad (3.4.14)$$

限制式(3.4.15)定義節線  $a$  在時刻  $t$  是否已被修復完。

$$z_{at}^k - \sum_{u=0}^t \sum_{r \in R^k} \delta_{atr}^{kF} = 0 \quad \forall t=0, \dots, T, \forall k \in K^a, \forall a \in A \quad (3.4.15)$$

限制式(3.4.16)規範毀損節線在前  $p_a^D$  時段內不可能被修復。

$$\sum_{t=0}^{p_a^D-1} z_{at}^k = 0 \quad \forall k \in K^a, \forall a \in A \quad (3.4.16)$$

限制式(3.4.17)規範毀損節線在前  $p_a^D$  時間內不可能結束其挖掘作業。

$$\sum_{t=0}^{p_a^D-1} \sum_{r \in R^D} \delta_{atr}^{DF} = 0 \quad \forall a \in A \quad (3.4.17)$$

限制式(3.4.18)規範毀損節線在前  $p_a^D + p_a^k + 1$  時間內不可能結束其修復作業。

$$\sum_{t=0}^{p_a^D + p_a^k - 1} \sum_{r \in R^k} \delta_{atr}^{kF} = 0 \quad \forall k \in K^a, \forall a \in A \quad (3.4.18)$$

限制式(3.4.19)規範毀損節線不可能開始被挖掘的時窗。

$$\sum_{t=T-\bar{p}_a-p_a^D+1}^T \sum_{r \in R^D} \delta_{atr}^{DS} = 0 \quad \forall a \in A \quad (3.4.19)$$

限制式(3.4.20) (3.4.21)規範毀損節線不可能開始被修復的時窗。

$$\sum_{t=T-p_a^k+1}^T \sum_{r \in R^k} \delta_{atr}^{kS} = 0 \quad \forall k \in K^a, \forall a \in A \quad (3.4.20)$$

$$\sum_{t=0}^{p_a^D-1} \sum_{r \in R^k} \delta_{atr}^{kS} = 0 \quad \forall k \in K^a, \forall a \in A \quad (3.4.21)$$

限制式(3.4.22)規範各毀損節線其開始與結束挖掘時刻的關係。

$$\delta_{a(t+p_a^D)r}^{DF} = \delta_{atr}^{DS} \quad \forall a \in A, \forall t = 0, \dots, T - p_a^D - \bar{p}_a, \forall r \in R^D \quad (3.4.22)$$

限制式(3.4.23)規範毀損節線開始與結束被修復時刻的關係。

$$\delta_{a(t+p_a^k)r}^{kF} = \delta_{atr}^{kS} \quad \forall t = p_a^D, \dots, T - p_a^k, \forall r \in R^k, \forall k \in K^a, \forall a \in A \quad (3.4.23)$$

限制式(3.4.24)規範毀損節線在還沒被挖掘前不能被修復。

$$\sum_{u=0}^t \sum_{r \in R^k} \delta_{aur}^{kS} \leq \sum_{u=0}^t \sum_{r \in R^D} \delta_{aur}^{DS} \quad \forall t = 0, \dots, T, \forall k \in K^a, \forall a \in A \quad (3.4.24)$$

限制式(3.4.25) (3.4.26)規範挖掘工作隊一旦開始挖掘不能中途暫停。

$$\sum_{u=t}^{t+p_a^D-1} \sum_{a' \neq a} \delta_{a'ur}^{DS} \leq M(1 - \delta_{atr}^{DS}) \quad \forall a \in A, \forall t = 0, \dots, T, \forall r \in R^D \quad (3.4.25)$$

$$\sum_{u=t+1}^T \delta_{aur}^{DS} \leq M(1 - \delta_{atr}^{DS}) \quad \forall a \in A, \forall t = 0, \dots, T, \forall r \in R^D \quad (3.4.26)$$

限制式(3.4.27) (3.4.28)規範修復工作隊一旦開始修復不能中途暫停。

$$\sum_{u=t}^{t+p_a^k-1} \sum_{a' \neq a} \delta_{a'ur}^{kS} \leq M(1 - \delta_{atr}^{kS}) \quad \forall t = 0, \dots, T, \forall r \in R^k, k \in K^a, \forall a \in A \quad (3.4.27)$$

$$\sum_{u=t+1}^T \delta_{aur}^{kS} \leq M(1 - \delta_{atr}^{kS}) \quad \forall t = 0, \dots, T, \forall r \in R^k, k \in K^a, \forall a \in A \quad (3.4.28)$$

限制式(3.4.29) (3.4.30) (3.4.31)規範節線  $a$  的最後修復完成的時刻(Makespan)。

$$F_a^k = \sum_{t=p_a^D}^{T-p_a^k} (t + p_a^k) \sum_{r \in R^k} \delta_{atr}^{kS} \quad \forall k \in K^a, \forall a \in A \quad (3.4.29)$$

$$F_a^k \leq f_a \quad \forall k \in K^a, \forall a \in A \quad (3.4.30)$$

$$f_a \leq F_a^k + T(1 - w_a^k) \quad \forall k \in K^a, \forall a \in A \quad (3.4.31)$$

限制式(3.4.32) 定義任一毀損節線  $a$  之最晚被修復的管線類別  $k$ 。

$$\sum_{k \in K^a} w_a^k = 1 \quad \forall a \in A \quad (3.4.32)$$

### 3.4.3 有效不等式

為加速整數規劃模式的收斂過程，本研究設計數個有效不等式(即限制式)以限縮求解區域。其中，限制式(3.4.33)規範若在時刻 $t$ 時，節點 $i$ 尚不可與供給點連通，則 $i$ 的所有進入節線 $(j,i)$ 當下應皆無法與供給點連通。

$$\sum_{\substack{a \in A \cup A^* \\ \text{head}(a)=i}} y_{at}^k \leq \text{IND}_i \cdot v_{it}^k \quad \forall i \in N \setminus \{s\}, \forall t = 0, \dots, T, \forall k \in K^a, a \in A \cup A^*, \text{head}(a) = i \quad (3.4.33)$$

限制式(3.4.34)規範節線的修復連續性（若時刻 $t$ 已被修復，該狀態將繼續保持）。

$$z_{at}^k \leq z_{a(t+1)}^k \quad \forall t = 0, \dots, T-1, \forall k \in K^a, \forall a \in A \quad (3.4.34)$$

限制式(3.4.35) 規範節線的連通連續性（若時刻 $t$ 已連通，該狀態將繼續保持）。

$$y_{at}^k \leq y_{a(t+1)}^k \quad \forall t = 0, \dots, T-1, \forall k \in K^a, \forall a \in A \cup A^* \quad (3.4.35)$$

限制式(3.4.36) 規範節點的連通連續性（若時刻 $t$ 已連通，該狀態將繼續保持）。

$$v_{it}^k \leq v_{i(t+1)}^k \quad \forall i \in N, \forall t = 0, \dots, T-1, \forall k \in K^a, \forall a \in A(i) \quad (3.4.36)$$

### 3.5 小結

本章闡述本研究提出的整數規劃模式，首先說明此問題為 NP-Hard，以 Lin (2016) 的整數規劃模式為基礎，加入先行的挖掘作業、多類管線修復、以及交通影響性的考量，提出 P<sub>many</sub> 整數規劃模式。為了加速求解，我們亦推導數個有效不等式。然而此問題實在太過複雜，使用目前公認效率不錯的求解軟體 GUROBI 仍需耗費很長的求解時間，因此我們將在下一章設計更有效率的求解演算法。

## 第4章 節線修復排程演算法之設計

在第三章所提出的整數規劃模式因為限制式甚較多，連處理小例子都會耗時甚久，因此本章將提出兩種求解演算法：4.1 節的貪婪式演算法、4.2 節的基因演算法，期望能在較短的時間內得到品質不錯的排程，較符合現實的需求。為方便解說，本章的演算法皆假設僅考慮兩類管線（譬如水、瓦斯）、三種類型工作隊（譬如挖掘、修水管、修瓦斯管等）、且同類型的所有工作隊皆有同樣的修復能力與效率。

### 4.1 貪婪式演算法

本節的「貪婪式演算法」有兩個版本：其中，「基本貪婪式演算法」(Generic Greedy Algorithm, GGA)僅利用 FCFS(First-Come, First-Served)法則來指派排程決策；之後我們再優先處置毀損的「邊界節線」，發展「邊界貪婪式演算法」(Boundary Greedy Algorithm, BGA)；此外，在求解過程中，為能快速計算出「各節點需求的等待總時間」，我們參考最短路徑的 Bellman-Ford Algorithm 來設計一個計算各節點開始連通（簡稱為「打通」）時刻的演算法(Node Connecting Time Algorithm, NCTA)相關步驟。

#### 4.1.1 基本貪婪式演算法(GGA)介紹

---

**Algorithm Generic Greedy Algorithm (GGA)**

---

**Require:**G, TotalWaitingTime

1: **Function** GGA(G,TotalWaitingTime)

2: **initialize**

3:     **while** RQ is not empty **do**

4:         get first element w in RQ   &&   put w in idle queue

5:         assign a dig task if possible by dig(List,Arc,w)

6:         assign a restore task if possible by repair(List,Arc,w)

7:     **end while**

8:     compute demandwaitingtime(List)

9:     compute trafficwaitingtime(List)

10     TotalWaitingTime = demandwaitingtime + trafficwaitingtime

11: **end function**

---

本演算法為每組閒置工作隊，選擇當下最短修復(挖掘)時間且未修復(挖掘)的毀損節線排程處理。現有一網路  $G=(N,A)$ ，設定  $TotalWaitingTime$  為總等待時間，包含「交通阻塞總時間」與「需求等待總時間」，利用 GGA 求解。在 GGA 中，正在修復中的工作隊集合設為  $RQ$ ，而  $RQ$  中的元素為工作隊編號設為  $w$ 。將  $RQ$  的工作隊以其預期的當下作業之完工時刻由小到大排序。因此每次取出  $RQ$  中的第一個元素，該元素必為當下即將最早結束修復或挖掘的工作隊之一；將此已結束工作的工作隊放進閒置的  $idleQ$  中，依工作隊結束時刻由小到大排序。同時，將每條節線的開始挖掘時刻、挖掘結束時刻，修復結束時刻等等資訊記錄在  $List$  中。接下來判斷是否有節線還未被挖掘，若有，則從閒置的  $idleQ$  中，尋找最早閒置的挖掘工作隊來挖掘該節線，直到沒有閒置的挖掘工作隊為止。再來判斷是否有節線已被挖掘卻還未被修復，若有，則從閒置的  $idleQ$  中尋找閒置的修復工作隊來修復，直到沒有閒置的修復工作隊為止。最後，當所有的工作隊都閒置時，代表全部的毀損節線皆已修復完畢，此時可計算此排程決策的目標函式值，包含兩部分：(1)總需求等待時間(即為  $compute demandwaitingtime$  的步驟，將在 4.1.4 節進行更詳細的介紹)，以及(2)交通不便的總時間(即為  $compute trafficwaitingtime$  的步驟)。

#### 4.1.2 邊界貪婪式演算法(BGA)介紹

由於 GGA 只考慮優先選取較短修復時間的管線，可能會選到離供給點太遠的毀損內部節線(Inside Arc)來處理，而該節線即使被修復完亦仍無法立即連通，對改善目標函式值沒有效果。因此本節提出「邊界貪婪式演算法」(BGA)，此演算法每次僅會挑選至少一端已連通的毀損邊界節線(Boundary Arc)來處理，倘若以此原則來選取待挖或待修節線的話，每次的修復完工經常會產生新的毀損邊界節線(亦即讓某些與邊界節線連結的內部節線進而轉變成邊界節線)，而每次修復界節線一定會連通該節線滿足其需求，因此必會改善目標函式值。倘若閒置工作隊較當下的邊界節線個數還多的話，此時才會指派某些閒置工作隊去處理當下的內部節線，而此時的內部選取方式可同時考量是否鄰近某一邊界節線以及其修復時間是否夠短等因素。



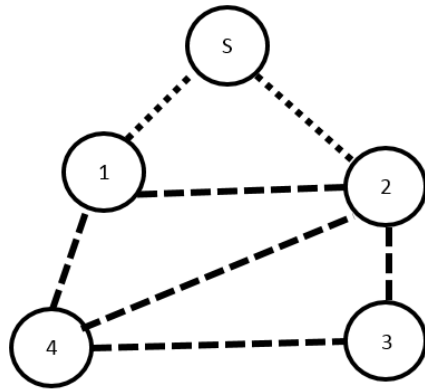


圖 4.1 邊界節線修復圖 1

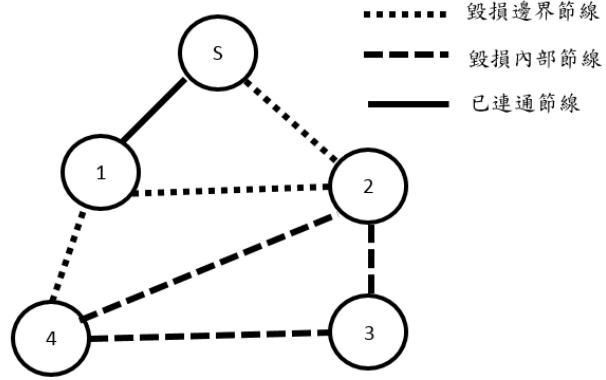


圖 4.2 邊界節線修復圖 2

圖 4.1 為一毀損網路圖，一開始與供給點  $s$  連通的毀損節線  $(s,1)$ 、 $(s,2)$  為邊界節線，其餘毀損節線皆為內部節線。當毀損節線  $(s,1)$  被修復完成即為圖 4.2，此時節點 1 與供給點連通，邊界節線更新為  $(s,2)$ 、 $(1,2)$ 、 $(1,4)$ ，內部節線更新為  $(2,3)$ 、 $(2,4)$ 、 $(3,4)$ 。

---

#### Algorithm Boundary Greedy Algorithm (BGA)

---

**Require:**  $G$ ，TotalWaitingTime

```

1: Function BGA( $G, RQ, idleQ, List$  TotalWaitingTime)
2:   while  $RQ$  is not empty do
3:      $a = \text{get the first element in } RQ$ 
4:     while  $idleQ$  is not empty and has assignment do ▶ step 1
5:        $b = \text{get the first element in } idleQ$ 
6:       if there is a boundary arc then ▶ step 2
7:         assign that boundary arc to  $b$  && Compute
           demandwaitingtime(List)
8:       else
9:         put in  $idleQ(b)$  ▶ step 3
10:      end if
11:    end while
12:    if select boundary arc then ▶ step 4
13:      select boundary arc(List,  $a$ ) && Compute demandwaitingtime(List)
14:    else
15:      put in  $idleQ(a)$ 
16:    end if
17:  end while
18:  compute trafficwaitingtime(List)
19:  TotalWaitingTime = demandwaitingtime + trafficwaitingtime
20: end function

```

---

在邊界貪婪式演算法 BGA 中，RQ 為正在挖掘或修復中的工作隊集合(依預期的完工時間排序)，idleQ 為閒置工作隊集合(依照閒置時間排序)，List 記錄每條毀損節線相關資訊(開始挖掘時間、結束挖掘時間等)，TotalWaitingTime 為總等待時間。

**Step 1:**每一回合開始指派工作時，會先檢查閒置工作隊中，是否有節線可以修復或挖掘。首先會檢查是否有邊界節線，尋找在當下所有邊界節線中權重(節線單位需求量/節線修復時間)最大者，每修完一條節線，皆需更新可能造成的連鎖效應，工作隊被指派完任務後皆需依序放入 RQ 中。

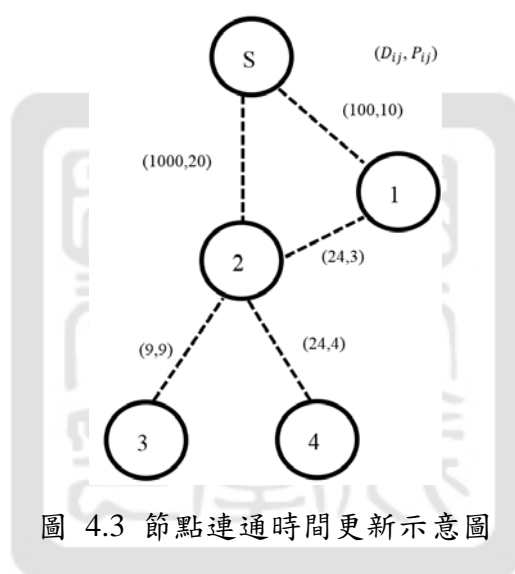


圖 4.3 節點連通時間更新示意圖

節線編號	(s,2)	(s,1)	(1,2)	(2,3)	(2,4)
節線權重	$1000/20 = 50$	$100/10 = 10$	$24/3 = 8$	$9/9 = 1$	$24/4 = 6$

表 4-1 節線權重表

以上圖為例，為了簡化問題描述，假設毀損節線都已挖掘完畢，只有一種類型毀損管線、兩組修復工作隊。第一步會先選擇與供給點連通且權重較大者進行修復，因此工作隊 1 先選擇節線(s,2)，節點 2 在時刻 23 後連通；第二步工作隊 2 會先選擇(s,1)，因為當下所有節點中，節點 s 最快被連通、節點 1 在時刻 13 後連通；第三步

工作隊 2(目前只有工作隊 2 閒置)會先選擇(1,2) , 因為與毀損節線相連的節點中, 節點 1 是最快連通, 節線(1,2)修復完後, 節點 2 最快可連通的時刻應更新為時刻 16。

**Step 2:**假如無邊界節線可以選擇, 則優先選擇在當下離邊界節線最近的內部節線中其權重(節線單位需求量/節線修復時間)最大者。本研究最近的計算方式, 依照相連節點打通的時間, 先挑選連通時間最短且可以修復(此節先必須已經被挖掘過)或挖掘的節線。每修完一條節線, 皆需更新其可能造成的連鎖效應, 工作隊被指派完任務後皆需依序放入 RQ 中。

**Step 3:**假設當下皆無可節線可被修復(可能還未挖掘或是已全數修復完畢)或挖掘(已全數挖掘完畢), 則將該工作隊放入 idleQ 中等待。

**Step 4:**檢查完 idleQ 後, 取出先前 RQ 最快完工之工作隊, 來挑選欲處理的毀損節線。

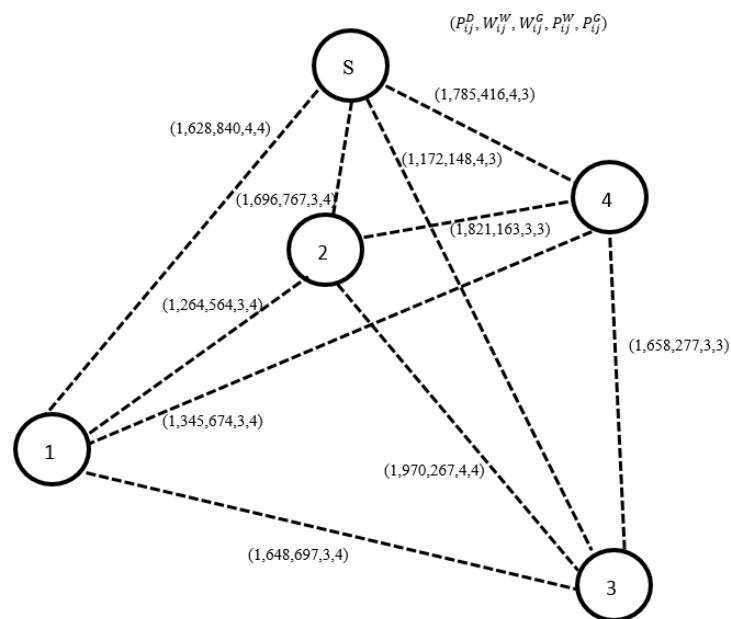


圖 4.4 待修復網路圖範例

#### 4.1.3 邊界貪婪式演算法範例介紹

以圖 4.4 為例, 有挖掘工作隊兩組, 修瓦斯工作隊三組, 修水管工作隊兩組。在挖掘的部分依序挑選與供給點連通的毀損節進行挖掘, 接下來依照節點連通的順

序進行挖掘節線的挑選，挖掘工作隊、不同類型修復工作隊，有各自網路節點的連通時間，假設多條毀損節線，同時其中一端節點最快連通時間相同，則選擇 Priority Queue 排序中，第一個節點中的第一條毀損節線，挖掘工作隊排程完的結果如圖 4.5。

在修復的部分以修復瓦斯工作隊為例，下表為節線相關資訊：

節線編號	(s,1)	(s,2)	(s,3)	(s,4)	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
節線權重	840/4	767/4	148/3	416/3	564/4	697/4	674/4	267/4	163/3	277/3
加權結果	210	191.7	49.3	138.7	141	174.3	168.5	66.8	54.3	92.3

表 4-2 瓦斯管線權重計算表

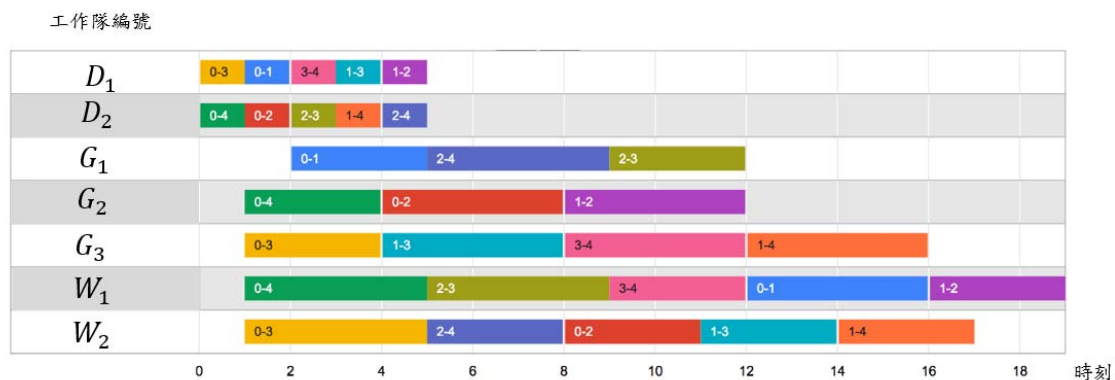


圖 4.5 邊界貪婪式演算法排程範例

**Step 1:**時刻 1 此時只有供給點 s 連通，會先挑選(s,3)、(s,4) 給工作隊 2、3，並更新節點 3 與節點 4 的連通時刻為 4。

**Step 2:**時刻 2 此時只有供給點 s 有連通，會先挑選(s,1)、(s,2)，因只有工作隊 1 閒置，因此挑(s,1) 給工作隊 1 並更新節點 1 的連通時為 5。

**Step 3:**時刻 4 此時工作隊 2、3 皆可被指派，而可被修復的節線依權重排序分別為 (s,2), (1,3), (2,4), (3,4), (2,3), (1,4)，因此挑(s,2)、(1,3) 給工作隊 2、3 並更新節點 2、3 的連通時皆為 8。依此類推，直到所有毀損節線皆修復完畢，排程如圖 4.5。

#### 4.1.4 節線打通時刻計算方式

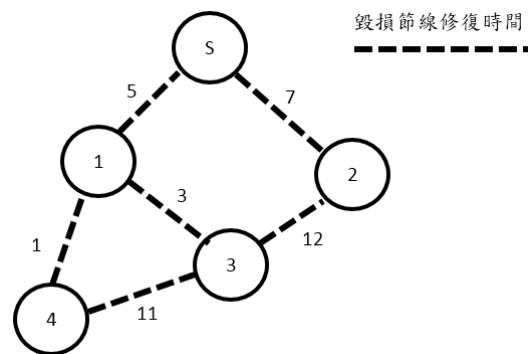


圖 4.6 需求等待修復圖 1

求解演算法可以很容易地知道工作隊何時修復好哪條節線上的某類管線，然而修復好管線並不保證該管線已可連通外界（即被打通），因此我們必須由排程結果再推導出各類管線在各節線上的打通時刻。以圖 4.6 為例，由節點  $s$  到節點 4 的修復順序總共有 4 種路徑(如圖 4.7, 4.8, 4.9, 4.10)，節點  $i$  的打通時刻(即為與供給點連通時刻)為  $p_i$ ，節線  $(i,j)$  的修復完成時刻為  $f_{ij}$ ，節線  $(i,j)$  的打通時刻(即為與供給點連通時刻)為  $q_{ij}$ 。

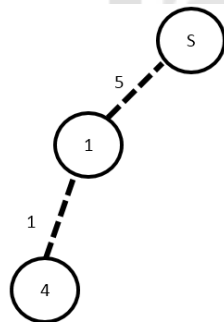


圖 4.7 需求等待修復圖 2

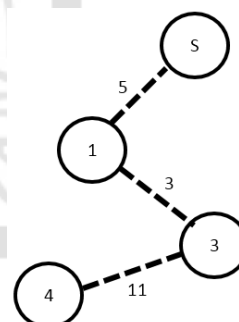


圖 4.8 需求等待修復圖 3

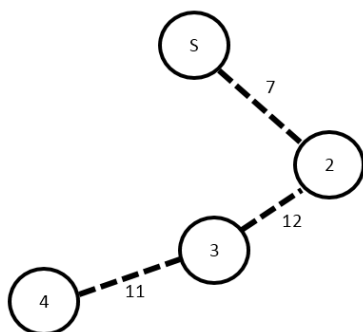


圖 4.9 需求等待修復圖 4

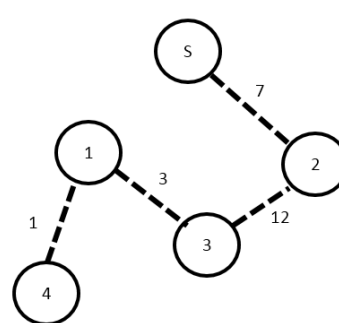


圖 4.10 需求等待修復圖 5

圖 4.7 節點 4 的打通時刻為  $\max\{1, 5\} = 5$ ，圖 4.8 節點 4 的打通時刻為  $\max\{5, 3, 11\} = 11$ ，圖 4.9 節點 4 的打通時刻為  $\max\{7, 11, 12\} = 12$ ，圖 4.10 節點 4 的打通時刻為  $\max\{7, 12, 3, 1\} = 12$ ，由此四張圖可發現，節點 4 最快打通的時間等待時間為  $\min\{\max\{1, 5\}, \max\{5, 3, 11\}, \max\{7, 11, 12\}, \max\{7, 12, 3, 1\}\} = 5$ ，因此可推論節點  $i$  的打通時刻  $p_i = \min_{(i,j) \in A} \{\max\{f_{ij}, p_j\}\}$ 。



圖 4.11 需求等待修復圖 6

以圖 4.11 為例，假設  $p_4 = 8$ ， $p_5 = 5$ ， $f_{4,5} = 9$  則  $q_{4,5} = 9$ ，因為即使左右兩邊節點已打通，但是要等節線(4,5)修復完才會打通。若  $p_4 = 8$ ， $p_5 = 8$ ， $f_{4,5} = 7$  則  $q_{4,5} = 8$ ，因為即使節線(4,5)已修復，但是要等兩邊節點打通完，節線(4,5)才會打通。另外， $p_4 = 8$ ， $p_5 = 5$ ， $f_{4,5} = 7$  為不合理的，因為  $f_{4,5}$  修復完應該會使  $p_4$  被立刻打通，因此  $p_4$  應為 7。由以上推論過程可推論節線  $(i,j)$  的打通時刻  $q_{i,j} = \max\{f_{i,j}, p_j, p_i\}$ 。

---

**Algorithm Node Connecting Time (NCTA)**

---

**Require:**List

1: **Function** NCTA(List,Arc)

2: **initial all**  $p_j$   $j \in N$  **and put source**  $s$  **in Queue**

3:     **while** Queue is not empty **do**

4:          $i$  = get first element in Queue

5:         **for each** arc  $(i,j)$

6:             **if**  $\max\{p_i, f_{i,j}\} < p_j$  **then**

7:                  $p_j = \max\{p_i, f_{i,j}\}$

8:             **if** node  $j$  is not in Queue **and then**

9:                 put  $j$  in Queue

10:             **end if**

11:         **end if**

12:         **end for**

13:     **end while**

14: **end function**

---

#### 4.1.5 節線打通時刻計算演算法(NCTA)

上小節中我們得知必須先推導出各節點  $i$  的打通時刻  $p_i = \min_{(i,j) \in A} \{\max\{f_{ij}, p_j\}\}$ ，才能再推導出各節線  $(i,j)$  的打通時刻  $q_{i,j} = \max\{f_{i,j}, p_j, p_i\}$ 。由於各節點  $i$  的打通時刻計算方式類似計算由外界供給點  $s$  至各節點  $i$  的最短路徑，因此本小節中我們修改計算最短路徑常用的 Bellman-Ford 演算法，提出一個節點打通時刻演算法(Node Connecting Time Algorithm, NCTA)來計算各節點  $i$  的打通時刻  $p_i$ ，再計算各節線  $(i,j)$  的打通時刻  $q_{i,j}$ 。

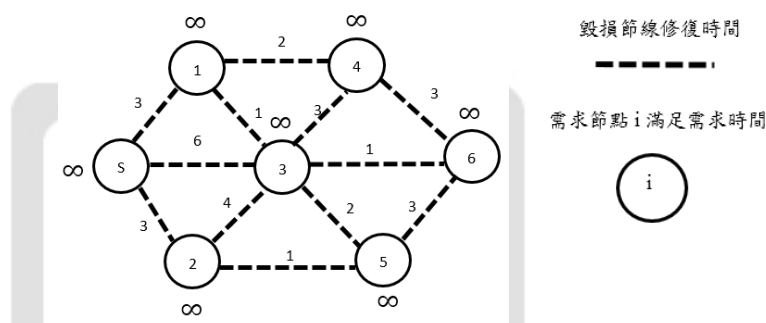


圖 4.12 NCTA 欲修復網路圖

以圖 4.12 為例，對 NCTA 進行步驟講解。

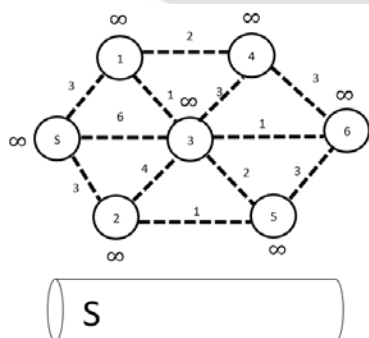


圖 4.13 NCTA 步驟 1 圖

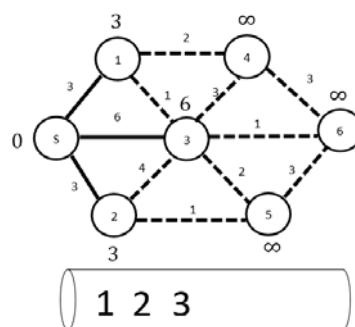


圖 4.14 NCTA 步驟 2 圖

先將所有需求節點的打通時刻的初始值設為無限大，並將供給點  $s$  放入 Queue 中，判斷 Queue 不為空並取出 Queue 中的第一個節點  $s$ ，並判斷是否更新相鄰節點打通時刻。以節點 1 為例： $p_s + f_{s,1} < p_1$  ( $0+3 < \infty$ )，因此更新  $p_1 = 3$ ，依此類推也

需更新節點 2 和節點 3，並確認 Queue 中是否有與節點 s 相鄰的節點，有則不用放入 Queue，沒有則需依序放入 Queue 中。

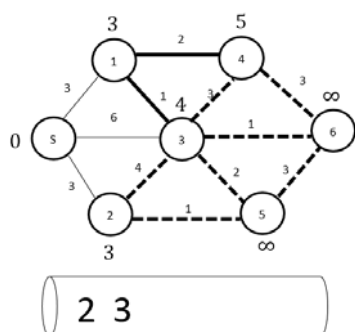


圖 4.15 NCTA 步驟 3 圖

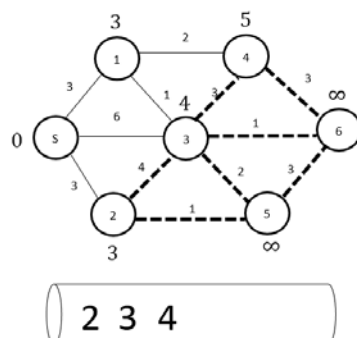


圖 4.16 NCTA 步驟 4 圖

判斷 Queue 不為空並取出 Queue 中的第一個節點 1，並判斷是否更新相鄰節點打通時刻，在此需更新節點 3 和節點 4，其中  $p_1 + f_{1,3} < p_3$  ( $3 + 1 < 6$ )， $p_3$  必須更新為 4，因為節點 3 已在 Queue 中，因此只需放入節點 4。

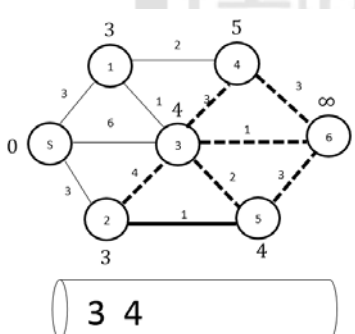


圖 4.17 NCTA 步驟 5 圖

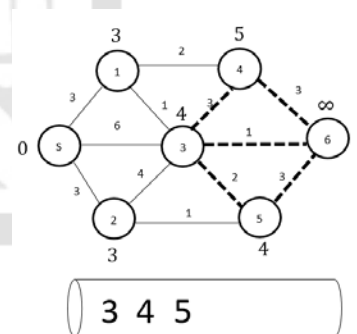


圖 4.18 NCTA 步驟 6 圖

判斷 Queue 不為空並取出 Queue 中的第一個節點 2，並判斷是否更新相鄰節點打通時刻，在此只需更新節點 5，其中  $p_2 + f_{2,5} > p_5$  ( $3 + 4 > 4$ )， $p_5$  不必更新，因此只需放入節點 5。



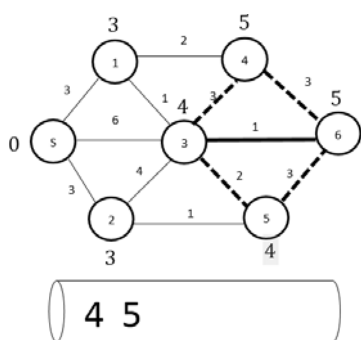


圖 4.19 NCTA 步驟 7 圖

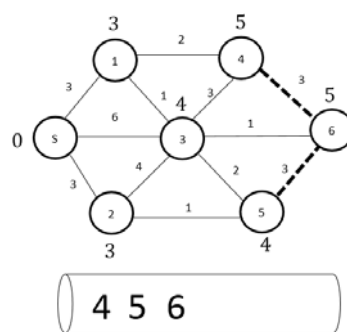


圖 4.20 NCTA 步驟 8 圖

判斷 Queue 不為空並取出 Queue 中的第一個節點 3，並判斷是否更新相鄰節點打  
通時刻，在此只需更新節點 6，因此只需放入節點 6。

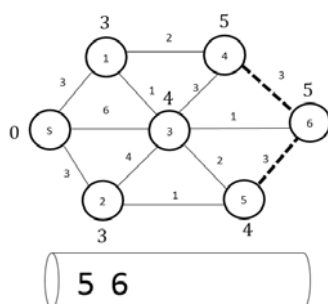


圖 4.21 NCTA 步驟 9 圖

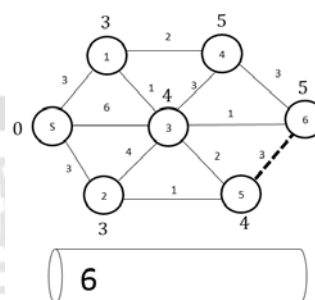


圖 4.22 NCTA 步驟 10 圖

判斷 Queue 不為空並取出 Queue 中的第一個節點 4，並判斷是否更新相鄰節點打  
通時刻，在此不需更新節點。

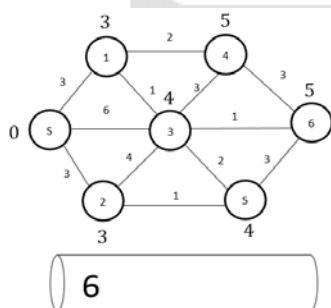


圖 4.23 NCTA 步驟 11 圖

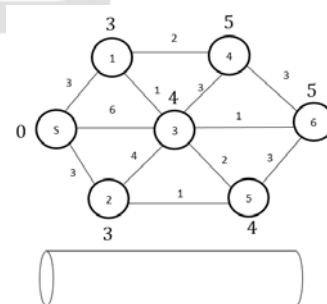


圖 4.24 NCTA 步驟 12 圖

判斷 Queue 不為空並取出 Queue 中的第一個節點 6，並判斷是否更新相鄰節點打  
通時刻，在此不需更新節點，直到 Queue 為空。

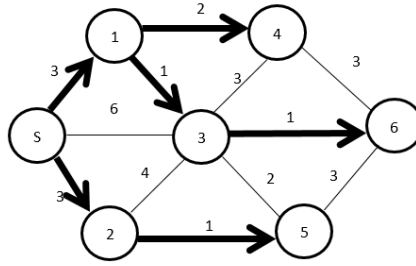


圖 4.25 NCTA 步驟完成圖

最後找到從供給點 S 到各需求節點最短的打通時刻，即為最短路徑圖。

以上 NCTA 演算法可以有效地計算各節點  $i$  的打通時刻  $p_i = \min_{(i,j) \in A} \{\max\{f_{ij}, p_j\}\}$ ，

之後必須再用一個迴圈掃過各節線  $(i,j)$ ，即能計算其打通時刻  $q_{i,j} = \max\{f_{i,j}, p_j, p_i\}$ 。

## 4.2 基因演算法(Genetic Algorithm, GA)

由於貪婪式演算法為一次性的解法，不會逐步收斂，因此本研究以 Vallada and Ruiz(2011)提出的 GA 為基礎，將其修改，符合本研究的問題，以下將 4.2.1 節說明 Vallada and Ruiz (2011)的原型設計方法。

### 4.2.1 基於多平行機台(unrelated parallel machine)排程問題的基因演算法

Vallada and Ruiz (2011)透過實驗設計，測試結果顯示其提出的方法可比整數規劃模式在較短的時間內得到不錯解。本研究假設工作隊的能力皆相同，若將各工作隊視為一機台，則可將本問題視為具網路結構的多平行機台的排程問題。本研究根據 Vallada and Ruiz (2011)提出的編碼方式、挑選策略、突變策略進行修改，並將貪婪式演算法所得出的解拿來當成基因演算法的初始解，以期加速收斂。

#### 染色體的編碼方式

一條染色體包含多組工作隊，而多組工作隊有不同的節線修復順序，此表示方式與傳統的 0、1 組合編碼不一樣。以圖 4.26 為例，假設毀損節線編號為 1~12，染色體有 4 組工作隊，工作隊 1 必須先完成毀損節線 1 的修復，接下來依序是 4、8、10，最後完成 12。而工作隊 1 到工作隊 4 的毀損節線編號不能重複，因此一條染色體就

能代表一組解，初始染色體的設定方式可採取隨機分配每工作隊其應修復的節線編號（之後我們亦會嘗試用貪婪演算法的結果來設定初始染色體）。以圖 4.26 為例，每個工作隊需修復節線數量不一定相同。

工作隊 1	1	4	8	10	12
工作隊 2	2	3			
工作隊 3	5				
工作隊 4	6	7	9	11	

圖 4.26 染色體表示圖

### 適應函式(fitness function)

在基因演算法中，必須透過適應函式求目標值，本研究的目標值為最小化需求總等待時間以及交通影響的不便性。當適應函式值越小越接近最佳解，在工作隊依序修復完節線後，依 4.1.4 小節的方式計算該組染色體的需求總等待時間，藉由交配與突變策略，使目標值越小越好。

### 挑選策略(selection operator)

每次疊代一開始，都會進行輪盤法挑選染色體組合。假設本研究有三種類型工作隊，分別是挖掘工作隊、修水管工作隊、修瓦斯管工作隊。假設每種類型工作隊各有 2 條染色體，此 2 條染色體為不同條。每次選各類型工作隊的其中一條染色體，依此類推，會有  $2 \text{ (挖掘工作隊)} \times 2 \text{ (修水管工作隊)} \times 2 \text{ (修瓦斯管工作隊)} = 8$  種染色體組合，每種組合會有一個適應函式值，適應函式值越小被挑中的機率越大，以下介紹其方法。

假設目前有  $n$  組染色體組合， $R_1$  為第 1 種組合， $R_2$  為第 2 種組合，依此類推，其適應函式值分別為  $F(R_1)$ ， $F(R_2)$ ， $\dots$ ， $F(R_n)$ 。

1. 將任一適應函式值減去當中最小者後再加 1 並取其倒數：

$$F(R_i) = \frac{1}{F(R_i) - \min_{j=1, \dots, n} F(R_j) + 1} \quad \forall i=1, \dots, n$$

2. 計算所有新適應函式值的加總:  $F_{total} = \sum_{i=1}^n F(R_i)$ 。
3. 計算各組染色體被挑中的機率:  $P_i = F(R_i) / F_{total}$ 。
4. 計算每組染色體的累積機率  $A_i$ 。
5. 隨機產生一個介於  $A_i \leq x \leq A_{i+1}$  的亂數  $x$ ，則選第  $i$  組染色體組合。

算機率時取倒數是為了讓適應函式值越小，被挑中的機率越大。又因適應函式值取倒數後非常接近，所以用平移的方式減去最小解，提高較佳解被選中的機率。

由上面所介紹的輪盤法可發現，較佳解被挑中的機率比較高，較差解被挑中的機率比較低，但不會因此就被排除，因為有可能透過交配或突變得到較佳解。

### 交配策略(crossover operator)

在交配策略的部分，進行有機率的交配，主要目的是讓挑選出來兩個染色體交換彼此的基因，以產生更優良的基因組合。以其中一種修復工作隊染色體為例，因為三種工作隊染色體交配方式皆相同。為確保每條毀損節線只會被一組工作隊修復，利用下列表示法：

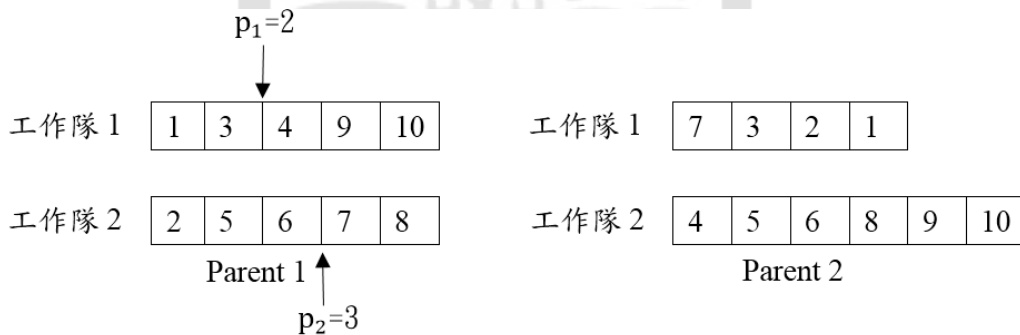


圖 4.27 選擇父母及分割點

如圖 4.27，一開始先隨機挑選兩條染色體當做父母，並選擇第一條染色體(Parent 1)的每個工作隊，隨機挑選切割點，譬如工作隊 1、2 的切割點分別為  $p_1$ 、 $p_2$ 。

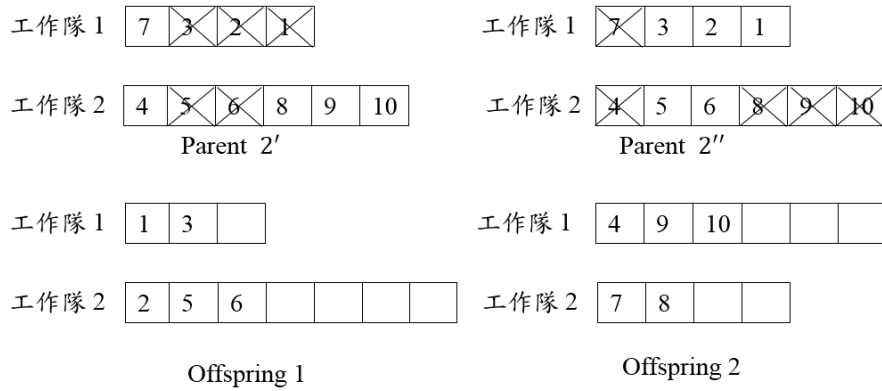


圖 4.28 進行複製與交換

如圖 4.28，針對 Parent 1 的每個工作隊中，切割點的兩端，分別加入新的染色體 Offspring 1 (Parent 1 切割點的左半邊)與 Offspring 2 (Parent 1 切割點的右半邊)。複製兩條 Parent 2'、Parent 2'' 染色體，分別對應給 Offspring 1、Offspring 2，並將 Parent 2'、Parent 2'' 的染色體分別刪去其與 Offspring 1、Offspring 2 重複的節線編號(見圖 4.28 上半部)。

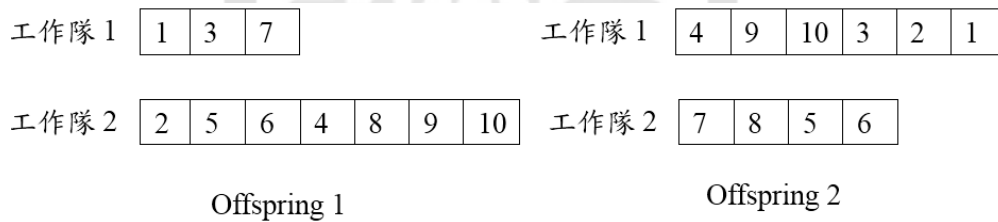


圖 4.29 最後小孩的結果

最後再將 Parent 2'、Parent 2'' 刪除後的剩餘節線，分別插入其所對應的 Offspring 1 與 Offspring 2 染色體中。本研究採取不變順序、直接將這些剩餘節線接續於對應的 Offspring 染色體後面，結果如圖 4.29 所示。若時間許可，或可在這個步驟執行一些鄰近搜尋步驟（譬如嘗試將剩餘節線插入其它位置，甚至可再亂調這些節線順序等等），使產生的子代染色體更有變化性。

### 突變策略(mutuation operator)

突變策略是為了避免基因演算法會落入區域最佳解，此部分的突變是有機率性，隨機挑選任一條染色體中的任一組工作隊，再隨機交換其任兩個工作，如圖

4.30，選到工作隊 2，並交換其工作 5 和工作 9。

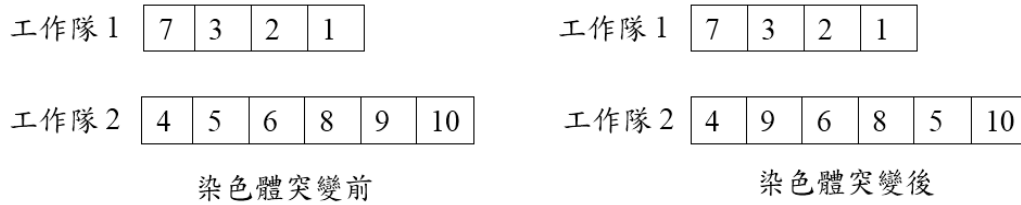


圖 4.30 突變策略

基因演算法 GA 的整個流程敘述如下，其中  $M$  為每回初始染色體組合的數量，每組染色體包含三個種類(挖掘工作隊染色體、修水管工作隊染色體、修瓦斯工作隊染色體)， $P_c$  為每回合交配的機率， $P_m$  為每回合突變的機率。

---

#### Algorithm Genetic Algorithm (GA)

---

```

1: Function GA( $M, P_c, P_m$ )
2:   Generate initial population(randomly generate  $M$  pairs of chromosome)
3:   time = 0
4:   while time < TotalIteration do
5:     calculate each pair of chromosome fitness in population
6:     select  $M$  pairs of chromosome and copy them to new population
7:     crossover by probability  $P_c$  and copy them to new population
8:     mutate by probability  $P_m$  for each chromosome
9:     time++
10:  end while
13: end function

```

---

#### 4.2.2 基因演算法範例說明

假設欲修復的毀損節線共有 7 條，分別為 $[1,2,...,7]$ ，有挖掘工作隊(D)、修復水管工作隊(R1)、修復瓦斯工作隊(R2)等三種類型工作隊，各類型工作隊又分別有兩小組工作隊，且每次疊代所保留的染色體數  $M$  為 3，交配機率 $P_c$ 及突變機率 $P_m$ 分別為 0.6 及 0.2。

[步驟一]疊代前，產生初始染色體，工作分配給各類型工作(假設採用隨機分配)。

$D_1 = [1,2,4], [3,5,6,7]; R1_1 = [1,3,2], [4,5,6,7]; R2_1 = [2,1,4], [7,6,3,5]$

$D_2 = [2,4,3], [5,1,6,7]; R1_2 = [1,3,5], [2,6,4,7]; R2_2 = [4,5,3], [2,1,6,7]$

$D_3 = [2,5,3], [5,4,7,6]; R1_3 = [4,2,5], [1,3,6,7]; R2_3 = [3,5,4], [2,1,7,6]$

**[步驟二]進行選擇策略**，一開始先計算每組染色體的適應函式值，採用 4.1.2 小節的適應函式值挑選策略，總共有三個： $K_1(D_1, R1_1, R2_1)$ 、 $K_2(D_2, R1_2, R2_2)$ 、 $K_3(D_3, R1_3, R2_3)$ 。 $E(K_1)=\alpha_1, E(K_2)=\alpha_2, E(K_3)=\alpha_3$ ，假設 $\alpha_3$ 為 $\alpha_1, \alpha_2, \alpha_3$ 中數值最小)

接著將原適應函式值轉成新適應函式值，如 4.2.1 小節所述： $E(K_1) = \frac{1}{(\alpha_1 - \alpha_3) + 1} = \beta_1$ ;

$$E(K_2) = \frac{1}{(\alpha_2 - \alpha_3) + 1} = \beta_2; E(K_3) = \frac{1}{(\alpha_3 - \alpha_3) + 1} = \beta_3$$

將其新適應函式值加總  $A = \beta_1 + \beta_2 + \beta_3$ ，並計算各機率  $P_1 = \frac{\beta_1}{A}, P_2 = \frac{\beta_2}{A}, P_3 = \frac{\beta_3}{A}$

計算累積機率： $Q_1 = \beta_1, Q_2 = \beta_1 + \beta_2, Q_3 = \beta_1 + \beta_2 + \beta_3 = 1$

因為  $M$  為 3，隨機產生三個亂數介於 0 到 1 之間，分別為  $r_1, r_2, r_3$ 。假設  $Q_1 \leq r_1 \leq Q_2$ ，則將  $K_2$  加入新的疊代當中；若  $0 \leq r_2 \leq Q_1$ ，則將  $K_1$  加入新的疊代當中；若  $0 \leq r_3 \leq Q_1$ ，則將  $K_1$  加入新的疊代當中，最後新的群體為  $[K_2, K_1, K_1]$ 。

**[步驟三]進行交配策略**，此部分因為  $M=3$  將會進行三次，每次會先產生一個介於 0 到 1 之間的亂數，若該亂數小於  $P_c$ ，便會隨機挑選兩條染色體進行突變。假設新的群體為  $[K_2, K_1, K_1]$  中，隨機亂數結果挑中  $K_2, K_1$  產生突變，以挖掘染色體為例如下：

$$D_1 = [1,2,4], [3,5,6,7] \quad D_2 = [2,4,3], [5,1,6,7]$$

隨機挑選  $D_1$  各工作隊的切割點，假設分別為 2、3，此時新產生出來的  $D_4$  和  $D_5$  如下：

$$D_4 = [1,2], [3,5,6] \quad D_5 = [4], [7]$$

接著複製兩個  $D_2$  並使其對應  $D_4$  和  $D_5$ ，將重複刪除可得到如下：

$$D'_2 = [4], [7] \quad D''_2 = [2,3], [5,1,6]$$

$$D_4 = [1,2], [3,5,6] \quad D_5 = [4], [7]$$

分別將  $D'_2$  和  $D_4$  結合、 $D''_2$  和  $D_5$  結合，採用 4.1.2 小節的交配策略適應函式值的計算方式，可得到的結果如下：

$$D_4 = [1,2,4], [3,5,6,7] \quad D_5 = [4,2,3], [7,5,1,6]$$

依此類推， $R1_1$ 和 $R1_2$ 也會隨機新切割點，產生相對應 $R1_4$ 和 $R1_5$ ； $R2_1$ 和 $R2_2$ 也會隨機新切割點，產生相對應 $R2_4$ 和 $R2_5$ ，並將所得到的新染色體組合 $K_4(D_4, R1_4, R2_4)$ 、 $K_5(D_5, R1_5, R2_5)$ 加入染色體群當中。

**[步驟四] 進行突變策略**，每一組解都有突變的可能，每次會先產生一個介於 0 到 1 之間的亂數，若其數字小於 $P_m$ 則進行突變。突變方式為隨機挑選一條染色體，將其中任一工作隊隨機選出的兩個不同工作交換。假設 $K_4$ 需要突變，以 $D_4$ 為例，隨機選到毀損節線 3 和毀損節線 5，如下：

突變前  $D_4=[1,2,4]$  ,  $[3,5,6,7]$

突變後  $D_4=[1,2,4]$  ,  $[5,3,6,7]$

依此類推， $R1_4$ 、 $R2_4$ 也會突變，產生新的一組 $K'_4$ 取代原 $K_4$ 。

以上大概是整個基因演算法的流程。

## 4.3 小結

由於本研究第三章所提出的整數規劃模式，即使使用專業的求解軟體 GUROBI，對於小規模的例子卻仍十分耗時。因此本章提出兩種演算法，希望能在短時間內得到品質不錯的解。第一種為基本貪婪式演算法 GGA，選取當下所需修復時間最短的毀損節線，此演算法能在極短的時間內得到不錯的解；而我們又進一步加以改良成邊界貪婪式演算法 BGA，將優先選取邊界節線的原則納入考量，更符合實際且比 GGA 的結果更好。這些貪婪演算法的解皆可被用來當最佳化軟體 GUROBI 的初始解，縮短其求解時間。第二種為基因演算法 GA，我們修改 Vallada and Ruiz (2011)提出的 GA，使其更貼近本研究問題的設定(加入多類型工作隊與特定工作隊有先後順序關係等)。而兩種演算法的排程解的目標值以及適應函式值的計算方式，則使用 NCTA 演算法以快速計算出節點與節線的連通性。



## 第5章 整數規劃模式與演算法之數值分析

本章將會對本研究所建立整數規劃模式、有效不等式及演算法，分別測試其在具樹狀結構(tree)與一般結構(general)的隨機網路圖之求解表現。測試環境為 Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz\*8 處理器與 16GB RAM 的個人電腦，UBUNTU 14.04.4 版本的作業系統，並利用最佳化軟體 GUROBI 版本 6.5.1 求解整數規劃模式。

### 5.1 測試網路資料

#### 5.1.1 產生網路結構圖工具

在現實生活中，管線並無特定形狀，管線末端像樹狀結構般，將資源分送到各需求單位，因此採用樹狀結構的網路。本研究使用 python 開發的 NetworkX 套件 (<https://networkx.github.io>) 來產生隨機的網路結構圖，其中樹狀結構圖使用 random\_powerlaw\_tree，藉由給定節點數來產生樹狀結構；隨機網路圖則使用 gnm\_random\_graph，依據給定節點數與節線數產生隨機網路，但必須檢查所有節點都有互相連通。表 5-1 列出本研究產生各種測試網路結構的參數設定。

網路名稱	節點數	節線數	節線修復 時間	需求 大小	工作隊數	筆數
$Tree_{s1}$	[10,15]	[9,14]	[3,5]	[3,5]	[2,3]	5
$Tree_{m1}$	[16,20]	[15,19]	[3,5]	[3,5]	[2,3]	5
$General_{s1}$	[5,10]	[11,21]	[3,5]	[3,5]	[2,3]	5
$General_{m1}$	[11,15]	[21,31]	[3,5]	[3,5]	[2,3]	5
$Tree_{s2}$	[10,15]	[9,14]	[3,5]	[100,1000]	[2,3]	5
$Tree_{m2}$	[16,20]	[15,19]	[3,5]	[100,1000]	[2,3]	5
$General_{s2}$	[5,10]	[11,21]	[3,5]	[100,1000]	[2,3]	5
$General_{m2}$	[11,15]	[21,31]	[3,5]	[100,1000]	[2,3]	5

表 5-1 測試網路結構資訊

#### 5.1.2 視覺化網頁工具

本研究希望透過視覺化工具，更清楚排程的變化方式，因此使用 Google 開發

的 timeline chart(<https://developers.google.com/chart/interactive/docs/gallery/timeline>)，會使用到的 HTML/CSS/javascript 來呈現整個排程的甘特圖。

### 工具操作說明

首先必須經由演算法或是整數規劃模式的求解結果，產出符合 google chart 的.txt 格式。該格式共有三個欄位：工作隊名稱、起始時間、結束時間，只要有這三個欄位的資訊，系統即可自動產生出視覺化結果。

```
dataTable.addRows([
  [ 'Dig Server - 第 0 組', new Date(2017, 1, 1, 0, 0, 1), new Date(2017, 1, 1, 0, 0, 20) ],
  [ 'Repair Server - 第 0 組', new Date(2017, 1, 1, 0, 0, 2), new Date(2017, 1, 1, 0, 0, 19) ] ]);

chart.draw(dataTable);
```

圖 5.1 google chart 範例資料

### 資訊呈現

如圖 5.2，挖掘工作隊有兩組(Dig Server 第 0、1 組)、修水管工作隊有三組(Repair water Server 第 0、1、2 組)、修瓦斯管工作隊有兩組(Repair gas Server 第 0、1 組)，共有 12 條毀損節線，每組工作隊的起始和結束時間皆可從該甘特圖顯示出來。

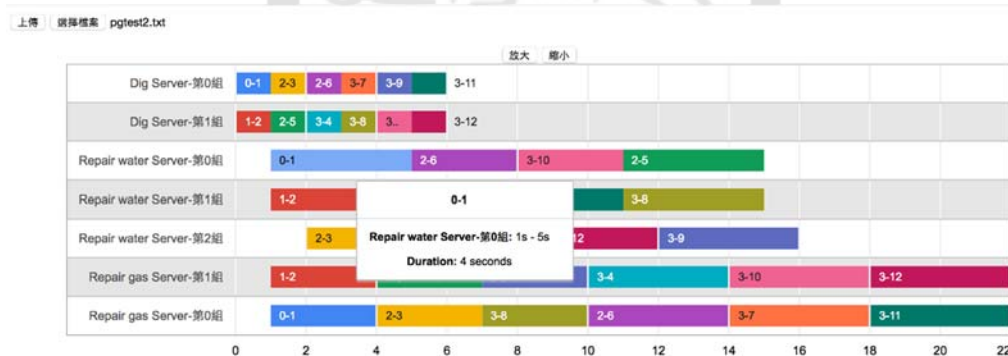


圖 5.2 google chart 視覺化範例截圖

## 5.2 整數規劃模式比較

實驗後發現求解  $Tree_{m1}$  時，在 5000 秒後其求解品質改善不大，因此  $Tree_{s1}$ 、

$General_{s1}$ 、 $General_{m1}$  分別以 5000 秒為時間上限。由於 GUROBI 求解這些問題皆十分耗時，我們測試改善初始解(將 BGA 之解當初始解)或加入 3.4.3 節所提出的四個有效不等式等技巧是否真能加速求解。表 5-2 列出整數規劃模式加入不同影響因素（是否使用 BGA 解當初始解、是否加入有效不等式以及加入哪條有效不等式）之情境代號表，希望經由實驗來觀察各因素對求解時間的相對影響性。

情境代號	$N$ 代表未使用 BGA 初始解	情境代號	$G$ 代表有使用 BGA 初始解
$N_0$	未加入有效不等式	$G_0$	未加入有效不等式
$N_1$	(3.4.33)	$G_1$	(3.4.33)
$N_2$	(3.4.34)	$G_2$	(3.4.34)
$N_3$	(3.4.35)	$G_3$	(3.4.35)
$N_4$	(3.4.36)	$G_4$	(3.4.36)

表 5-2 有效不等式與加入初始解之測資情境代號表

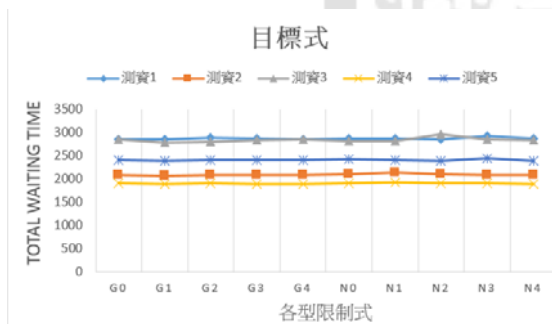


圖 5.3  $Tree_{m1}$  各類型限制目標值分析圖

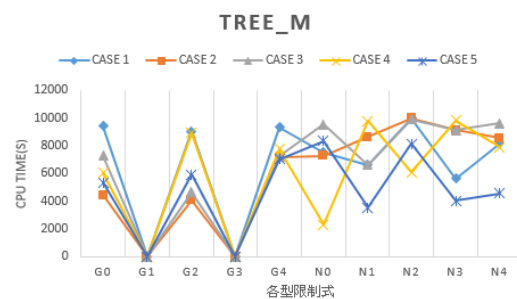


圖 5.4  $Tree_{m1}$  各類型限制時間分析圖

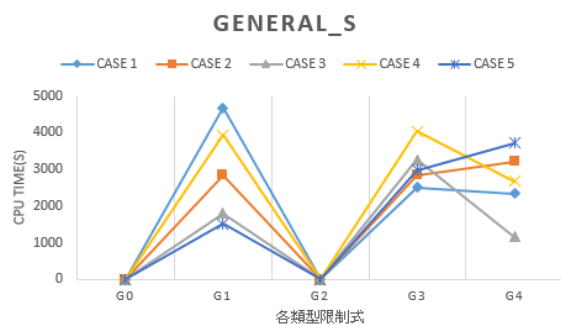


圖 5.5  $General_{s1}$  各類型限制時間分析圖

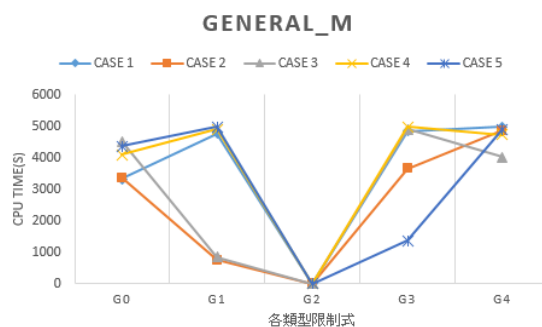


圖 5.6  $General_{m1}$  各類型限制時間分析圖

由圖 5.3 可發現在 10000 秒的時間限制下，GUROBI 在各種情況下找到的最好可行解有相似的品質，但是由圖 5.4 ~5.6 找最好的可行解所花的時間差異很大。其中以 BGA 解當起始解的情境所花的時間平均而言較少，而  $G_1$ 、 $G_3$  對 tree 網路結構比較有利， $G_2$  對 general 網路結構比較有利。

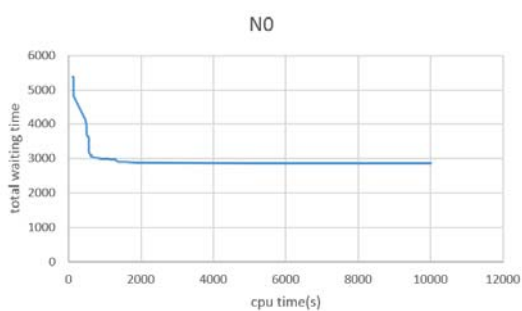


圖 5.7  $Tree_{m1}$  測資中  $N_0$  分析圖

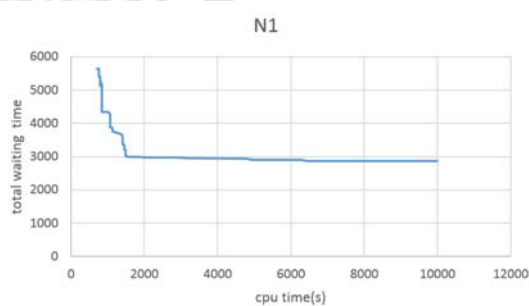


圖 5.8  $Tree_{m1}$  測資中  $N_1$  分析圖

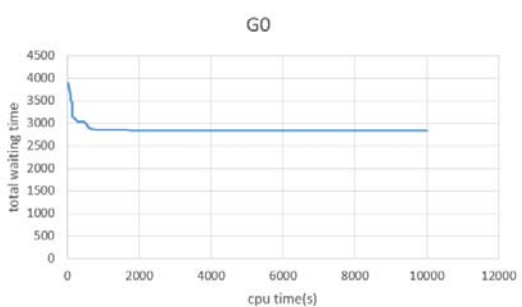


圖 5.9  $Tree_{m1}$  測資中  $G_0$  分析圖

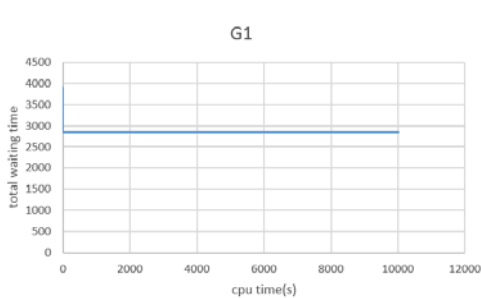


圖 5.10  $Tree_{m1}$  測資中  $G_1$  分析圖

代號	<i>Tree<sub>s1</sub></i>		<i>Tree<sub>m1</sub></i>	
	時間(s)	Gap(%)	時間(s)	Gap(%)
<i>N<sub>0</sub></i>	4105	6.958	10000	36.34
<i>N<sub>1</sub></i>	5000	6.318	10000	41.54
<i>N<sub>2</sub></i>	5000	12.534	10000	43.64
<i>N<sub>3</sub></i>	5000	12.554	10000	39.4
<i>N<sub>4</sub></i>	5000	12.656	10000	35.52
<i>G<sub>0</sub></i>	4318	14.072	10000	34.38
<i>G<sub>1</sub></i>	4119	7.886	10000	39.52
<i>G<sub>2</sub></i>	5000	14.796	10000	37.76
<i>G<sub>3</sub></i>	5000	12.184	10000	42.72
<i>G<sub>4</sub></i>	5000	10.18	10000	37.24

表 5-3 *Tree<sub>s1</sub>*、*Tree<sub>m1</sub>* 有效不等式與加入初始解測試表

將圖 5.3 的結果分別再針對其中 *N<sub>0</sub>*、*N<sub>1</sub>*、*G<sub>0</sub>*、*G<sub>1</sub>* 情境進行更詳細的求解過程分析，我們發現在 10000 秒內，*G<sub>0</sub>*、*G<sub>1</sub>* 較 *N<sub>0</sub>*、*N<sub>1</sub>* 更快收斂，且又以加入限制式(3.4.33)的 *G<sub>1</sub>* 的效果最好。由圖 5.7 到 5.10 可發現，5000 秒內解即可收斂，之後的求解品質改善不大，因此之後實驗的時間上限皆設 5000 秒算合理。由表 5-3 可發現 5000 秒內求解小規模樹狀測資其 Gap 仍大於 5%，而中規模的測資在 10000 秒內其 Gap 仍大於 30%。

代號	<i>General<sub>s1</sub></i>		<i>General<sub>m1</sub></i>	
	時間(s)	Gap(%)	時間(s)	Gap(%)
<i>G<sub>0</sub></i>	5000	37.62	5000	68.96
<i>G<sub>1</sub></i>	5000	32.24	5000	73.88
<i>G<sub>2</sub></i>	5000	33.32	5000	74.42
<i>G<sub>3</sub></i>	5000	31.6	5000	72.7
<i>G<sub>4</sub></i>	5000	36.52	5000	70.5

表 5-4 *General<sub>s1</sub>*、*General<sub>m1</sub>* 有效不等式與加入初始解測試表

而一般(general)構造的隨機網路結構又更為複雜，因此我們直接測試使用 BGA 的解當初始解的情境，比對表 5-3、5-4 可發現同樣時限下，隨機網路結構求解更為困難。

## 5.3 貪婪式演算法測試

### 5.3.1 不同類型貪婪式演算法測試

表 5-5 記錄兩類貪婪演算法測試情境，貪婪演算法過程中的指派任務選擇雖然皆為一次性的決策方式，但當有多重最佳的指派選擇時，或許嘗試其它同樣最佳的選

擇亦有不錯的結果，因此我們嘗試以隨機方式選取多重最佳選擇，如此一來每次執行 BGA/GGA 時，多做幾次(增加疊代次數)可能會找到更好的貪婪可行解。

疊代次數	BGA	GGA
1000	$BG_{1000}$	$G_{1000}$
10000	$BG_{10000}$	$G_{10000}$

表 5-5 貪婪式演算法符號表

	$BG_{1000}$		$BG_{10000}$		$G_{1000}$		$G_{10000}$	
	時間 (s)	Gap(%)	時間 (s)	Gap(%)	時間 (s)	Gap(%)	時間 (s)	Gap(%)
$Tree_{s1}$	0.6	11.9254	5.4	11.9373	0	48.5695	1.6	48.4750
$Tree_{m1}$	1.2	15.5832	12.2	15.5673	0	67.3028	2.2	67.2013
$General_{s1}$	0.2	14.0584	5	14.0530	0.2	27.1833	1.8	27.2648
$General_{m1}$	1.4	15.1631	13.8	15.1631	0.6	26.0672	3.2	26.0613

表 5-6 貪婪式演算法測試比較

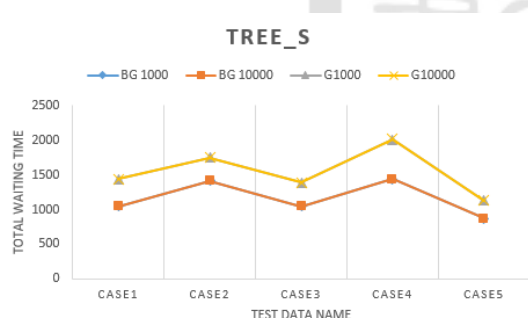


圖 5.11 貪婪式演算法比較 1

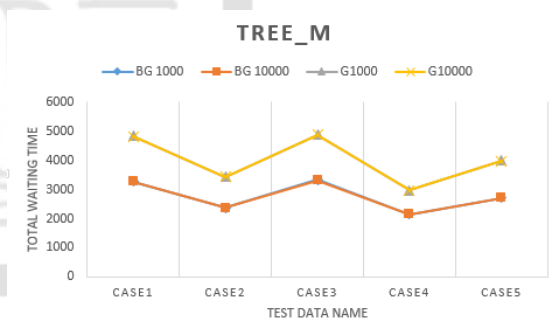


圖 5.12 貪婪式演算法比較 2

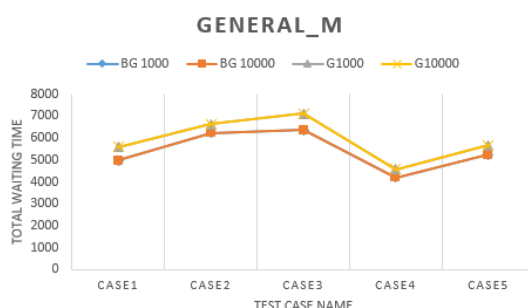


圖 5.13 貪婪式演算法比較 3

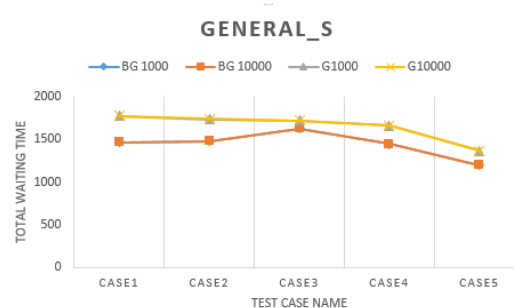


圖 5.14 貪婪式演算法比較 4

由表 5-6 與圖 5.11~5.14 得知，貪婪式演算法的求解品質似乎與疊代次數較無關，

BGA 所得到的解普遍優於 GGA 所得到的解。由表 5-6 可發現本方法經常必須搜尋多重最佳的指派選擇，而這個過程似乎有點費時，因此在 5.3.2 小節，我們將嘗試使用 Priority Queue 的排序方式，測試是否能加速求解。

### 5.3.2 貪婪選擇納入單位需求考量之測試

本研究先前提出的貪婪演算法主要以維修時間的長短來決定其貪婪法則，卻忽略了毀損節線上的需求量大小這個變因。因此本節的貪婪式演算法將單位需求量一併納入貪婪法則，選擇單位時間內的修復可以滿足最多需求量的節線來修復(亦即選取單位需求量/修復時間的比值最大者)。為了觀察此一變因的影響性，我們將測資的單位需求量範圍設在 100 到 1000 之間(原先僅在 3 到 5 之間)，如此設定亦會減少貪婪法則的多重選取機會。

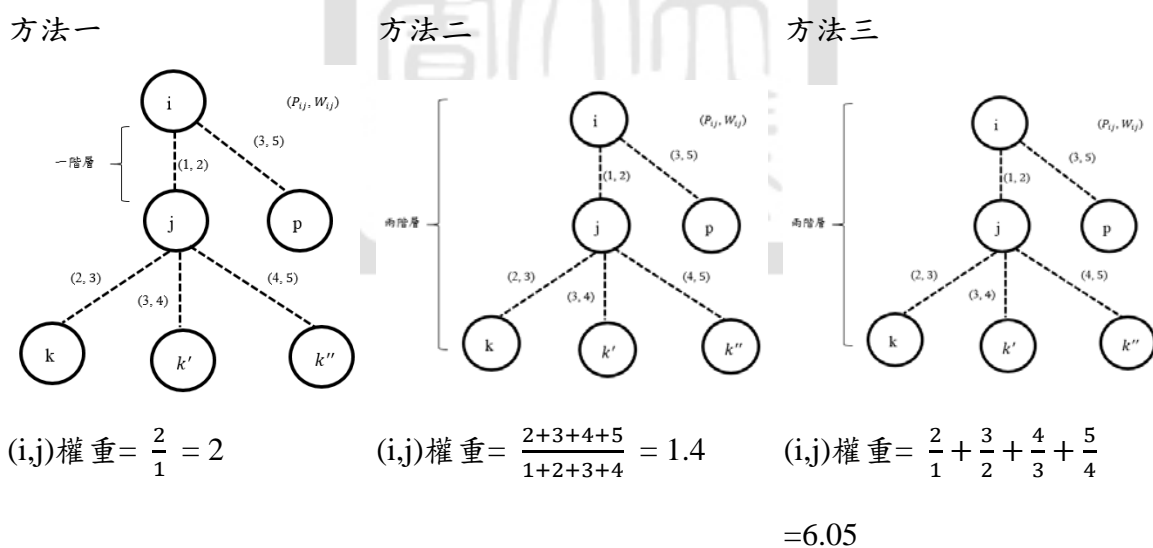


圖 5.15 不同權重計算表

此外，這些節線的選取方式通常是考慮搜尋第一層，若再往下多搜尋一層亦可能選取到不錯的節線。本節測試了三種不同權重計算方式來測試，其中第一種方式僅搜尋當下那一層節線，而第二、三種方式搜尋了兩層節線但使用不同權重方式：第一種方式

選擇當下待修復節線之最大權重( $\frac{W_{ij}}{P_{ij}}$ )者，第二種方式選擇當下修復節線(i,j)與其一邊

節點 j 相鄰的所有節線(j,k)中最大權重( $\frac{\sum W_{jk}+W_{ij}}{\sum P_{jk}+P_{ij}}$ ) 者，第三種方式選擇當下修復節線

(i,j)與其一邊節點 j 相鄰的所有節線(j,k)的最大權重( $\sum \frac{W_{jk}}{P_{jk}} + \frac{W_{ij}}{P_{ij}}$ ) 者，如圖 5.15 所示。

(i,p)的權重為 1.67，方法一和三會優先選擇(i,j)；方法二會優先選擇(i,p)。

	方法一(GAP%)	方法二(GAP%)	方法三(GAP%)
<b><i>Tree<sub>s2</sub></i></b>	23.2469	23.2469	28.2158
<b><i>Tree<sub>m2</sub></i></b>	9.6572	9.6055	14.2657
<b><i>General<sub>s2</sub></i></b>	8.0459	14.8339	29.4912
<b><i>General<sub>m2</sub></i></b>	16.1988	17.8048	22.9721

表 5-7 不同權重所得 GAP

表 5-7 的結果顯示納入需求量的選擇考量的確有效(對照圖 5.15)，然而多層搜尋的機制的效益至少在本測試仍不顯著。

## 5.4 基因演算法測試

基因演算法每回合的初始染色體數量分別測試 4 與 8 兩種不同的情況，交配的機率 $P_c$ 設為 0.6，突變的機率 $P_m$ 設為 0.1，使用表 5-1 中的 $Tree_{s1}$ 與 $General_{m1}$ 進行測試。

基因演算法條件	基因演算法設定情境代號
4 條染色體+疊代 1000 次	4_1000
4 條染色體+疊代 10000 次	4_10000
8 條染色體+疊代 1000 次	8_1000
8 條染色體+疊代 10000 次	8_10000

表 5-8 基因演算法代號

	<b><i>Tree<sub>s1</sub></i>(沒加 greedy)</b>		<b><i>General<sub>m1</sub></i>(加 greedy)</b>	
	時間(s)	Gap(%)	時間(s)	Gap(%)
<b>4_1000</b>	189.6	15.63022	458.4	11.15936
<b>4_10000</b>	1909.4	15.12592	4592	10.31468
<b>8_1000</b>	1935.8	16.08131	3390	9.809282
<b>8_10000</b>	14505	14.62846	34478.2	9.442604

表 5-9 基因演算法測試 1



由表 5-9 可觀察出測試  $General_{m1}$  (加 greedy) 的 gap 比  $Tree_{s1}$  (沒加 greedy) gap 低，可說明使用 BGA 初始解能較快收斂到較好的解。此外，本測試顯示染色體數增加或是疊代次數增加的影響有限，這可能是因為初始解品質太好，難以跳脫局部最佳解。

## 5.5 小結

本章首先比較數種加速整數規劃模式的求解技巧（以貪婪演算法的可行解當初始解、加入數條有效不等式），以  $Tree_m$  的測資分析數種技巧的求解表現，發現使用貪婪式演算法的可行解當初始解這個技巧會有較顯著的效益；而在測試的四種有效不等式中，又以限制式(3.4.33)有較好的加速效益。由求解小例子測資卻仍無法在 10000 秒內得到整數規劃模式的最佳解表現看來，本問題的確十分困難。以  $Tree_m$  的測資為例，GUROBI 執行 5000 秒與 10000 秒的求解結果相差不大，因此本研究大部分測資皆以 5000 秒為測試時間上限。一般架構的隨機網路因其結構比樹狀網路結構更為複雜，因此求解將更加耗時。為了更進一步加速求解，我們以 BGA 解當 GUROBI 或基因演算法的初始解，皆可在一開始即將 GUROBI 或基因演算法的解迅速收斂至一個品質還可以的區域最佳。可惜的是，目前為止我們仍無法找到有效的方式跳出此區域最佳解，這方面有待未來繼續努力。

## 第6章 結論與未來研究

### 6.1 結論

本研究探討當同時有多種類型的地下管線因天災或人禍而導致大規模毀損時，該如何派遣工作隊挖掘、修復所有毀損管線，以使倚賴這些管線輸送物資(例如：水、瓦斯、電、網路等)的需求之總體等待時間、以及肇因於管線修復的交通不便影響性能越小越好。此研究議題十分重要，算是災後管理的重要決策規劃問題。由於管線埋設在地下，必須先開挖才能啟動修復作業，而道路一旦開挖便開始影響交通，且各類管線通常有其專屬的修復工作隊，因此如何統籌指揮有限的挖掘與各類管線修復工作隊，是一個複雜且具有網路結構的特殊 RCPSP 問題。

若從求解 RCPSP 的角度來看，本研究的網路結構使得我們的問題比傳統 RCPSP 更為複雜難解，這主要是單看單一類管線網路的修復規劃排程問題(Lin, 2016)已十分複雜，更何況我們要同時處理多類管線網路及考量附帶的交通影響程度。以下總整本研究的主要貢獻：

1. 提出  $P_{many}$  整數規劃模式：我們延伸了 Lin (2016) 的研究，使其更符合多類地下管線的現實狀況，更是首度針對多類管線的修復、以及交通不便的影響性等考量的排程研究。
2. 首度提出 BGA、GGA 等貪婪式演算法，而實作中亦將隨機選取多重最佳的工作指派方式納入貪婪演算法的設計，讓貪婪演算法亦可執行多次（而非傳統僅能執行一次）。
3. 首度發現可利用類似最短路徑中的 Bellman-Ford 演算法來推算排程決策造成的節點最早打通時刻，並進而得知多項式時間內可由排程決策計算而得各節線的最早打通時刻，並據此推算目標函式值。
4. 盡可能測試多類隨機網路，目前為止，本研究發展的演算法之最佳表現仍至少有 10% 的 Optimality Gap，由此得知此排程問題十分困難。

## 6.2 未來研究方向

本研究還有許多待改善部分且亦可引發不少相關具挑戰下議題，茲臚列如下：

1. **加入道路回填的工作隊：**本研究目前只考慮挖掘工作隊和修復工作隊，把回填的工作假設由各節線的最後一組修復工作隊處理。若要更符合現實影響交通的考量，可能可考慮將道路回填的工作獨立成另一種類型工作隊，且可能可以規定不能讓已挖掘好的節線閒置過久，譬如若閒置超過一週則必須回填道路，以免影響交通太多。若是如此，則將變成同一節線可能會經歷多次挖掘、回填作業直至其所有毀損管線完成修復，如下圖 6.1 所示。然而這樣的設定，將導致其整數規劃模式的设计十分困難。

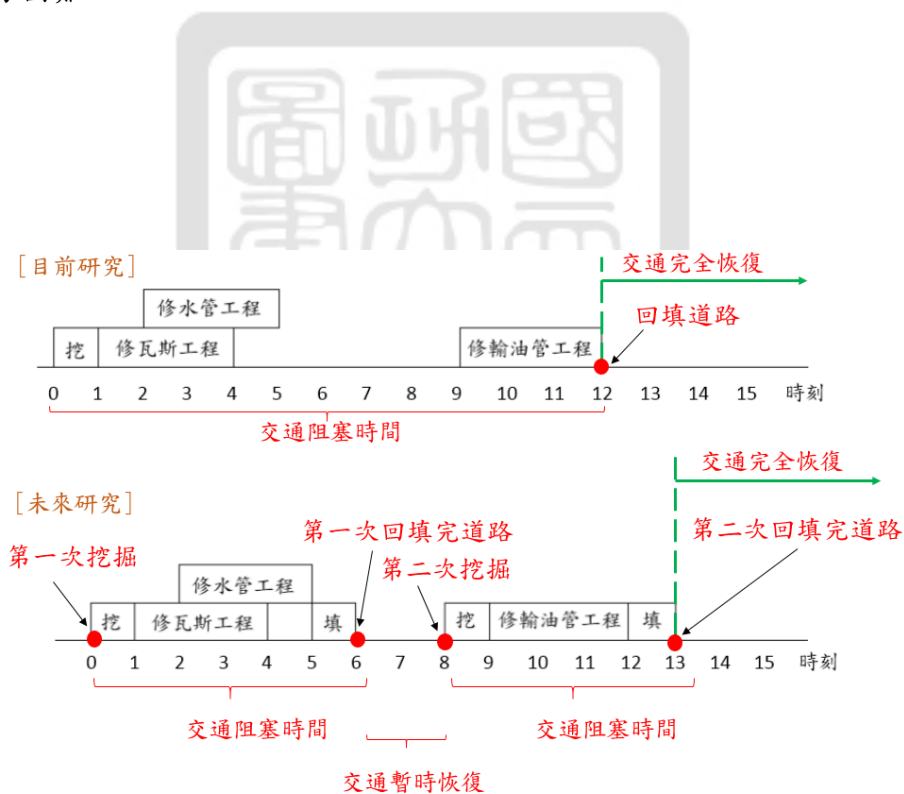


圖 6.1 加入填補類型工作隊

2. **設計更好的整數規劃模式或演算法：**本研究的整數規劃模式雖然正確，但連小規模的測資都難以在合理(譬如 5000 秒)內求得最佳排程，因此可能還有不少努力空間。在 Lin (2016)研究中使用 SSH 演算法有不錯的求解績效，或許將來可考慮發

展類似的求解演算法。此外，如何有效的改善貪婪演算法或基因演算法，避免它們陷入局部最佳解，這也是本研究一直無法突破之處。

3. **考量工作隊效益不同或合作等 multi-mode**：目前本研究假設同類修復工作隊的能力與效益均相同，然而現實生活中工作隊可能各有其擅長的修復技能或機具，如此將使本議題更加困難。解決之道可能可考慮將工作隊可能的組合各以不同的模式(mode)看待，而整數規劃模式將不是以工作隊為單位，而是以工作模式為單位來指派排程決策。
4. **毀損狀況的評估任務排程**：本研究假設管線的毀損狀態與其修復處理時間皆為已知，然而在實際狀況中這兩種資訊其實很難確認。舉例來說，水管是否破損目前幾乎倚靠使用者回報的方式來猜測，至於某些沒有使用者回報的地方其毀損狀態即難以得知或判斷。至於預期的修復處理時間又與管線毀損狀態息息相關，此資訊若預估錯誤，當然會影響整體的修復排程甚多。針對多處未有回報或難以評估毀損狀態的管線，該如何決定是否派員評估？又應該去哪裡評估（假設有多處未回報，而評估人力有限）？這些議題其實也都非常重要。
5. **汰換舊管線排程**：台灣氣候高溫、潮濕又高鹽分，加速了地下管線設備的氧化鏽蝕速度。舊管線須定期維護與更換，否則會造成意外、或大量的流量資源浪費。在眾多待汰換的舊管線中，應如何選取其汰換排程才能使需求越快被滿足且汰換管線期間的不方便性越小越好。

## 參考文獻

- Coelho, J. ,& Vanhoucke,M., (2011) , “Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers,” *European Journal of Operational Research*, **213(1)** , 73-82.
- Heath, E.A. , & Mitchell, J.E. , & Sharkey, T.C, (2016) , “Applying ranking and selection procedures to long-term mitigation for improved network restoration,” *Journal on Computational Optimization*, **4(3)** ,447-481.
- Averbakha, I. ,& Pereira, J. , (2012) , “The flowtime network construction problem,” *IIE Transactions*, **44(8)** , 681-694.
- Nurre,S.G. ,& Sharkey,T.C., (2014) , “Integrated network design and scheduling problems with parallel identical machines: Complexity results and dispatching rules,” *Networks*, **63(4)** , 306-336.
- Averbakha, I. , (2012) , “Emergency path restoration problems,” *Discrete Optimization*, **9** , 58-64.
- Nurre,S.G. ,& Cavdaroglua,B. ,& Mitchellb,J.E. ,& Sharkeya,T.C. ,& Wallacea,W.A. , (2012) , “Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem,” *European Journal of Operational Research*, **223(3)** , 794-806
- Nurre,S.G. ,& Sharkey,T.C. , (2015) , “Online Integrated Network Design and Scheduling Problems with Flexible Release Dates,”
- Tabucchi,T. ,& Davidson,R. ,& Brink,S.(2010) , “Simulation of post-earthquake water supply system restoration,” *Civil Engineering and Environmental Systems*, **27(4)** , 263-279
- Hung,T.M.(2015) , “A multi-mode network restoration problem in post-disaster

- humanitarian logistics management,” *National Cheng Kung University, Tainan, Taiwan*.
- Lin,P.C. , (2016) , “An arc restoration scheduling problem for pipeline networks in post-disaster management,” *National Cheng Kung University, Tainan, Taiwan*.
- Vallada, E., & Ruiz, R. , (2011) , “A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times., ”*European Journal of Operational Research*, **211(3)** , 612-622
- Debels,D.,& Vanhoucke,V. , (2005) , “A bi-population based genetic algorithm for the resource-constrained project scheduling problem,” *Lecture Notes in Computer Science*, **3484**, 378-387.
- Brucker,P. ,& Knust,S. ,& Schoo,A. ,& Thiele,O. , (1998) , “A branch and bound algorithm for the resource-constrained project scheduling problem,” *European Journal of Operational Research* , **107** ,272-288
- Vallada,E.,& Rubén,R. , (2011) , “A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times,” *European Journal of Operational Research* , **211** ,612-622
- Valls,V.,& Ballestín,F. , & Quintanilla,S. , (2008) , “A hybrid genetic algorithm for the resource-constrained project scheduling problem,” *European Journal of Operational Research* , **185** ,495-508