國立成功大學

資訊管理研究所

碩士論文

災後管理之管線網路最佳節線修復排程研究

An arc restoration scheduling problem for pipeline networks in post-disaster management

研究生：林秉錚

指導教授：王逸琳 博士

中華民國一百零五年八月

國立成功大學

碩士論文

災後管理之管線網路最佳節線修復排程研究
An arc restoration scheduling problem for pipeline networks in post-disaster management

研究生：林秉錚

本論文業經審查及口試合格特此證明

論文考試委員：

2逸琳　李雨青

王中允　李宇欣

指導教授：2逸琳

系(所)主管：李兵慶

中　華　民　國　105　年　5　月　26　日

摘 要

地下管線系統常被用來輸送水、天然氣、電力、或通訊封包等生活必須物資，而其需求主要分佈在管線網路之節線上。倘若地震、颱風等天災或恐怖攻擊等人禍導致網路的節線受損，勢必將對倚賴該毀損管線生活的居民造成極大不便，因此這些管線的修復排程便成爲災後管理中非常重要的決策。本研究探討此種必須連通至供給點方能接受物資流量的特殊網路修復問題，假設每條毀損管線所影響的需求以及其修復時間皆爲已知，欲求解修復所有毀損管線的最佳排程，以使生活受其影響的居民之總體等待物資時間越短越好。

本研究依實務現況，將需求設定成分佈在節線而非傳統文獻之節點上。先證明該排程問題爲*NP-hard*，再提出一網路縮減機制整合未毀損節線的效能，將原網路簡化成較小但僅包含受損節線的等效網路，以提高其最佳修復排程之求解效率。針對以單一工作隊來修復所有毀損管線的特例，本研究以「具資源限制下之專案排程問題」(Resource Constrained Project Scheduling Problem，RCPSP)爲基礎，提出自可連通供給點的節點開始修復的整數規劃模式；而在考量多組工作隊修復模式時，爲了確保修復過程中已與外界連通節點或節線的「持續連通性」（該節點或節線在後續的修復過程中應持續保持與供給點的連通），我們提出兩個整數規劃模式：(1)利用Branch-and-Cut(B&C)架構之模式，在求解過程中一遇到獨立子迴圈即產生新限制式排除之，直至該整數規劃解不含獨立子迴圈之排程爲止，以及(2)使用多商品網路流量(Multi-Commodity Network Flows，MCF)之模式，以各節點是否已收到供給點提供的單位虛擬流量來判斷當下是否已連通外界，再推導可加速其求解之有效不等式(Valid Inequality)，並將其延伸成可處理不同修復能力的多組工作隊修復排程數學模式。

爲了解決當網路規模變大導致整數規劃模式求解緩慢的問題，本研究設計了「分段式啓發演算法」(Sequential Segment Heuristic，SSH)，將總修復時間切成多個較小求解區段(Time Horizon)再依序求解之；而針對單一工作隊的特例則設計了「貪婪樹生成法」(Greedy Tree method，GT)、「貪婪連通分量生成法」(Greedy

Connected Component method，GCC)等兩種貪婪式演算法，以不同的貪婪法則決定各節線修復順序，與「窮舉法」(Brute Force method)比較後得知GT與GCC在求解樹狀與一般網路時皆可在短時間內求出近似最佳解，而GT在稍加修正後亦可處理一般網路測資；以上針對單一工作隊所求解而得的管線修復排程皆可再進一步加上多組工作隊的指派決策，以處理多組工作隊修復排程問題。此外，鑑於多組工作隊修復排程問題與平行機台排程問題相關，本研究亦參考常用於平行機台排程的基因演算法流程，以各工作隊預計修復之節線編號依序排列成染色體，設計三個基因演算法：(1)使用平行機台排程文獻建議的交配與突變策略，(2)與(3)則改良(1)交配與突變後染色體的節線排序方式，讓較接近供給點的節線較能優先被修復。

在測試過數組隨機產生的樹狀網路(Tree Graph)、一般網路(General Graph)以及較符合現實情況大小的網路後，我們推薦使用SSH與GCC等兩種啓發式演算法以在短時間內求得品質不錯(Optimality Gap $< 2\%$)的近似最佳解；基因演算法雖在求解一般網路時表現尚可(數秒內Optimality Gap約 $< 4\%$)，但求解樹狀網路之效果不佳；而最佳解則建議使用MCF之模式。

關鍵字：網路修復問題、災後管理、整數規劃、專案排程問題

An arc restoration scheduling problem for pipeline networks in post-disaster management

Ping-Cheng Lin

I-Lin Wang

Institute of Information Management

SUMMARY

Pipeline networks that ship flows of gas, water, electricity, or packets are important for supporting our daily livings. Suppose arcs of a pipeline network are damaged by disasters and the limited resources (equipment, manpower, and time) required to restore each damaged arc have been estimated. We investigate the problem of when and who to restore which arcs such that the flows over pipelines become accessible for people among all arcs at minimum total waiting time in the post-disaster management. We first reduce the original network into an equivalent but smaller one where only damaged arcs are considered. Then, we propose two integer programs (Branch-and-Cut, Multicommodity Network Flows frameworks) for this special resource constrained project scheduling problem. Several valid inequalities are proposed to effectively shorten the computational time for integer programs. We also explain how to deal with more general cases where heterogeneous teams as well as their collaboration are considered for calculating an optimal network restoration schedule. Three fast heuristics are designed: Sequential Segment Heuristic (SSH), Greedy Tree method (GT), and Greedy Connected Component method (GCC), where SSH solves partitioned segments sequentially, and both GT and GCC first determine an arc restoration sequence in different greedy fashions and then assign tasks to different teams by a First-Come First-Served principle. We propose three variants of Genetic Algorithms (GA). Computational experiments indicate our heuristics could calculate solutions within 2% optimality gaps in seconds for large-scale networks. The GAs, on the other hand, perform a little worse with optimality gaps up to 5%.

Keywords: Network Restoration, Post-disaster Management, Integer Program, Project Scheduling Problem

INTRODUCTION

The quality of our daily living relies very much on the convenient accesses to the water, gas, electricity, or packets which are usually shipped by underground pipeline networks connected from remote reservoirs or plants to our buildings. In a pipeline network, a node is usually equipped with a control valve to connect flows to or disconnect flows from its adjacent arcs (i.e., links). Flows inside an arc are pressurized to emanate into smaller pipelines connected to individual buildings or facilities aboveground.

Large-scaled natural or man-made disasters such as earthquakes or terrorist attacks might damage the underground pipeline networks and cause leakages in arcs. Since the flows inside a pipeline network are pressurized, any arc leakage would result in continuous leaking flows. To stop the leaking flows, for each leaking arc, a restoration team has to excavate the earth above it, close the control valves of its end nodes, repair or replace it, and then backfill the earth above it. These restoration operations may take days or even weeks, depending on the level of damages and the length of damaged arcs. Take the earthquake that hit Tainan at Taiwan on Feb. 06, 2016 which caused 117 casualties for example, it took more than one month to make water accessible for all residents in the damaged area. On Jul. 31, 2014, a series of man-made gas explosions occurred in Kaohsiung at Taiwan which leaked 3.77 tons of propene. The explosions destroyed not only the gas pipelines, but also the pipelines of water, electricity, and even the paved roads aboveground. It took months to restore all the pipeline networks.

The duration of being inaccessible to these flows depends on the schedules for repairing the damaged pipelines. In general, the restoration procedures should aim to reduce the long waiting time of accessing flows for all the people affected. The resources (teams, equipment, and budgets) for conducting the restoration procedures are usually limited. How to design an effective restoration schedule so that all the limited resources can be effectively utilized to benefit all the demands as soon as possible becomes a very important and challenging combinatorial optimization problem. To simplify this scheduling problem, we make the

following assumptions: Suppose the set of leaking arcs, and the resources (e.g., time, budget, and manpower) to restore each leaking arc have been estimated beforehand. Each leaking arc is ready to be repaired by any available and qualified restoration team, and the restoration procedure is non-preemptive. A restoration team can at most repair a damaged arc at any time.

If we consider restoring all damaged arcs as a project, the restoration teams as resources, and restoring a damaged arc as an individual task, then the arc restoration scheduling problem corresponds to a special resource constrained project scheduling problem (RCPSP) which is an NP-hard problem. Our problem differs from conventional RCPSP since it involves the network structure, which means the adjacency relations between arcs should be considered to calculate an optimal restoration schedule. Although it seems intuitive to restore those arcs closer to the source (i.e., undamaged arcs full of pressurized flows) earlier, such an intuition may not always work for an optimal schedule in the cases of multiple restoration teams. It is also nontrivial to design good greedy heuristics, since an optimal schedule may be affected by several factors: processing times, sizes of arc demands, and adjacency relations between arcs.

MATERIALS AND METHODS

We assume the network considered is connected, should all arcs are restored. Although we allow some arcs to be undamaged, in fact these undamaged arcs can be reduced to self loops associated with nodes, so that the remaining non-loop arcs are all damaged. By our proposed network reduction scheme, we can reduce the original network to an equivalent but smaller one, which should be easier to deal with.

Assuming the network only contains damaged arcs after network reduction, here we propose three integer programming models to calculate an optimal restoration schedule that minimizes the total waiting time for all demands in all damaged arcs. In particular, the model M^s can deal with the situation for a single restoration team, while $M^m_{B\&C}$ can cope with the problems for multiple restoration teams. $M^m_{B\&C}$ adds subtour elimination constraints

to guarantee the connectivity between accessible nodes to the source, yet it is time consuming sine many cuts may require to be added and leads to longer computational time for solving an integer program in each iteration. We propose another model M^m_{MCF} that exploits the multicommodity network flows framework to calculate an optimal restoration schedule more efficiently than $M^m_{B\&C}$, yet M^m_{MCF} is still time consuming to deal with larger networks. In order to further shorten the computational time, we derive several valid inequalities and add them as new constraints to M^m_{MCF}. We also explain how to take the restoration teams of heterogeneous capabilities and efficiencies, as well as their collaboration into consideration in the modified formulation of M^m_{MCF}.

In order to calculate a good solution within shorter time for larger networks, we propose a modified SSH algorithm modified from Huang (2015), which solves partitioned segments of network sequentially, and two greedy heuristics GT and GCC which determine the arc restoration sequences in different greedy fashions and then assign each scheduled restoration task to a restoration team by a First-Come-First-Serve principle. In particular, SSH can be adjusted by the lengths of the planning horizon and overhead for each segment. In general, a segment of longer planning horizon and overhead can give a better schedule but more time-consuming. GT is more suitable for tree-like networks, and GCC is applicable for networks of general topology. We also mention how to modify GT to deal with general networks. Moreover, we design 3 variants of Genetic Algorithms (GA) based on Vallada and Ruiz (2011) for solving our network restoration problem.

RESULTS AND DISCUSSION

Our testing was performed on a personal computer equipped with UBUNTU 13.10 OS, intel® i7-4770 3.40 GHz*8 Processors, and 16 GB RAM. Gurobi 6.5.1 version is used for solving integer programs. All algorithms are written in C/C++ language. Based on our computational experiments, we have the following observations: (1) integer programs are usually time-consuming and not suitable for solving large scale networks, (2) M^m_{MCF} has much better performance than $M^m_{B\&C}$, (3) the derived valid inequalities for M^m_{MCF} do shorten

the computational time, yet how to obtain the best combination of valid inequalities is not clear. (4) our proposed heuristics SSH, GT and GCC can calculate very good solutions in short time. To be more specific, SSH may require longer time but usually give better solutions than other heuristics, whereas GT and GCC are very fast but with a slightly larger optimality gaps (within 2%) than SSH. (5) 3 GAs perform similarly but have even larger optimality gaps (about 4% or worse sometimes) than GT and GCC.

CONCLUSION

The contributions of this paper are in 3-folds: first, we might arguably give the first exact IP models to deal with the network restoration problem by multiple restoration teams over general graphs where demands are distributed along arcs, which should be more realistic compared with other related previous studies that assume demands are distributed on nodes; second, we are the first research to test the performance of adding different new valid inequalities, which can provide good suggestions for future research in developing better Integer Programming models; third, we have developed three efficient and effective heuristics (SSH, GT, and GCC) that can obtain solutions of good qualities much faster than the exact methods.

誌 謝

時光荏苒，研究所的日子就這樣在這兩年的時間過去了，Pokemon Go盛行的當下也終於要畢業去面對未來的工作，從四下進來Ilin Lab後，也做了不少事情、結識許多的研究夥伴，感謝有大家的陪伴在這求學之路可以順順遂遂，當然最感謝的便是家人的支持以及女朋友的陪伴。

感謝指導教授 王逸琳教授這幾年的訓練，不管是火車比賽還是課堂的訓練，亦或是英文conference的撰寫，都有著實的練習，知道自己在口說上以及自信心上的不足，也參加了許多比賽或者會議來加強口說報告的能力，而這些能力也會伴隨著到未來的工作裡，有了這些紮實的訓練，不敢說能一路順利，但一定會深深地紮根與發芽。

感謝實驗室同屆的夥伴們，貞泰的視覺化工具與網站設計能力總是讓人歎爲觀止，采緹不管在哪方面都可以很快的駕馭，邏輯方面也跟麒霖拌嘴的日子裡大幅提升，謝謝你們總能讓我這個井底之蛙對世界有更深的體會；現在升上碩二的筱昀以及BK你們也要繼續加油，祝福筱昀可以往自己的夢想邁進、BK可以找尋到自己夢想中的__子；而剛進來實驗室的冠緯以及思涵，記得撐下去就對了；而應該是老師第一個博班的富元，每次看你修課就累得半死，還有一堆事情要弄，雖然做著自己喜歡的事應該也不會太累，但身體還是要顧喔，飲料喝少一點XD；當然還要感謝學長們帶我們這屆宅宅認識更多人，有你們的經驗也讓我們成長得更加快速。

感謝我的朋友們，不管是常常吃大餐去開發台南的餐廳或是踩雷，亦或是玩手機遊戲放鬆心靈，有你們這些事情才會變得有趣起來，之後大家也要各奔南北，希望久久見一次面能夠再度聽到大家在這世界努力生存與開拓的故事。

最後感謝我的女朋友，從一開始因爲太忙而疏於關心，到最後能夠體諒我這個只要開始工作就無法注意周遭變化的工作狂，有你的陪伴讓研究所生活可以從256色變成高彩，也祝福未來研究所的生活可以順利。

謝謝你們，未來珍重！

目 錄

表 目 錄

圖 目 錄

第一章　緒論

1.1　研究背景與動機

與生活息息相關的多種物資(譬如水、電、天然氣、網路等)經常倚賴地下管線系統輸送，而當這些管線一旦遭受災害或人爲的恐怖攻擊而導致毀損時，如何讓使用這些管線的人們能儘快回復正常生活甚爲重要，本研究即是探討如何指揮多組工作隊修復所有毀損管線的最佳排程，以使生活受其影響的居民之總體等待修復時間越短越好。

修復管線需要注意的地方是，如果沒有與物資供給點的網路相連通，該物資就無法送達，所以工作隊如果先去修復需求量最多的管線但卻無法與供給點連通，這些管線上的需求一樣不會被滿足。這好比是開水龍頭流入管線一樣，若管線爲連通，那麼水就可以到達，但不連通的地方，水永遠無法到達。以待修復網路圖1.1爲例，假設節點S爲供給點，虛線表示節線毀損，實線表示未毀損，各節線上的D_{ij}及P_{ij}分別代表節線上的需求量以及損壞時所需的修復時間；此待修復網路圖若先修復具5單位需求量的節線(C, D)，因其修復完畢仍無法與供給點S連通，所以經過4單位修復時間後，5單位需求仍無法被滿足。由此例可知，當僅有一組工作隊時，應依「由外而內」的順序來修復較合理。

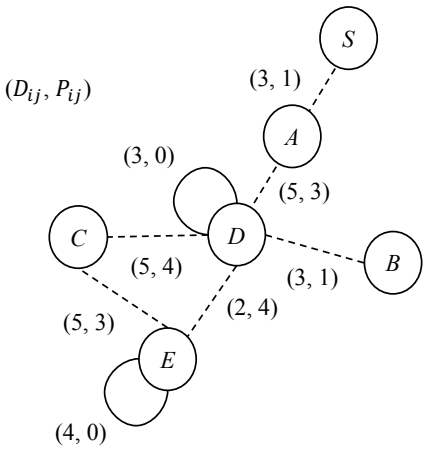


圖 1.1: 待修復網路圖

然而，當有多組工作隊時，因可同步在多處進行修復，不見得必要遵守上述「由外而內」的直覺修復順序，因此其修復管線的排程決策考量更爲複雜。本研究擬將此排程決策分成兩大部分考量，分別是修復路徑的決策以及工作隊的排程規劃，而這也與 Nurre and Sharkey (2014) 裡面提到的派工法則 (Dispatching rule) 類似。希望藉由本研究可以給決策者提供最佳修復方式的理論數學基礎與規劃模式，並設計多種兼具效能與效率的排程演算法，以滿足能於短時間內求解出品質不錯的修復排程決策之現實需求。

1.2 研究目的

本研究的網路修復問題，主要著眼於將物資補給需求能被滿足的總體等待時間極小化，此與文獻中較常探討如何修復網路而使給定時段內能產生最大流量、最小成本流或$s-t$最短路徑的「整合網路設計與排程問題」(Integrated Network Design and Scheduling Problem，INDS)之最佳化目標不甚相同。若將整個網路修復視爲一個專案，個別毀損節線的修復即爲專案中的子任務，而修復工作隊即爲有限的救災資源，則本問題可被視爲一個具有網路架構的特殊「具資源限制下之專案排程問題」(Resource Constrained Project Scheduling Problem，RCPSP)。一般的RCPSP會先給定相鄰子任務間的先後關係，在本問題中子任務間雖無明顯的先後關係規範(在此我們假設可隨時修復任一毀損節線)，但不同節線的修復(子任務)順序卻會對總體等待時間有不同的影響，本研究希望能規劃出工作隊的最佳搶修排程。以下舉圖1.1爲例來說明本問題之決策複雜性：假設現有兩工作能力相同之工作隊欲修復此網路之所有節線，若以圖1.2及圖1.3等兩種不同方式來修復。其中圖1.2及圖1.3內的每一方框代表一毀損節線的修復工作，Y軸代表工作隊編號、X軸代表時刻。若以圖1.2排程來修復，在時刻2時，第一工作隊剛好修復完節線(B, D)，而第二工作隊則正在修復節線(A, D)。

圖 1.2: 第一種修復順序排程

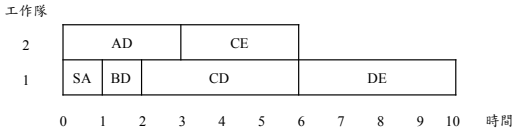


圖 1.3: 第二種修復順序排程

圖1.2的修復方式，在時刻1之前，因尚未滿足所有需求，所以必須將所有節線上的單位需求加總起來 3 (D_{SA}) + 5 (D_{AD}) + 3 (D_{DD}) + 3 (D_{BD}) + 2 (D_{DE}) + 5 (D_{CD}) + 5 (D_{CE}) + 4 (D_{EE}) = 30，到時刻1時，因節線(S, A)已被工作隊1修復且與供給點S連通，所以D_{SA}的3單位需求已被滿足，依此類推，在時刻2、3、6、7、9時，分別會滿足6 ($D_{BD} + D_{DD}$)、5 (D_{AD})、6 ($D_{DE} + D_{EE}$)、5 (D_{CD})、5 (D_{CE})單位需求，如此可得最後的需求總等待時間爲152，同理可以知圖1.3的修復排程其需求總等待時間爲140。此例顯示，雖然圖1.3的修復排程在時刻10才完成，其完工時刻比圖1.2慢一單位時間，但其總等待時間卻比圖1.2短，因此較早完成修復的直覺排程決策不見得會有較短的總體等待時間。

另一個常見的排程直覺是希望針對需求較高的管線先進行搶修，以網路圖1.4爲例。

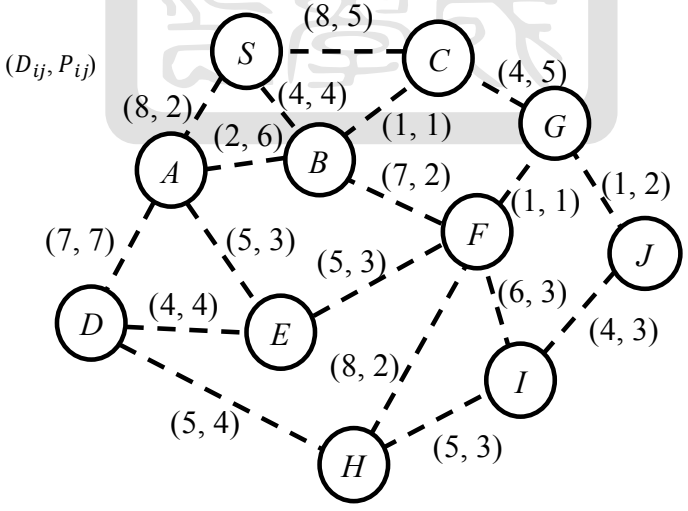


圖 1.4: 待修復網路圖二

若現在有兩個工作隊進行搶修，根據直覺的判斷會先修復與供給點相連且節線需求最高的節線(S, A)與(S, C)，如圖1.5所示，但如果我們觀察整個網路的分佈，可以發

現雖然節線(S, B)所需花費的時間較長需求也較少，但因爲其後續所連接的節線(B, F)及(F, H)皆可以在2單位時間內滿足8單位的需求，因此若使用如圖1.6的修復排程，相較於先考量優先修復需求較大的節線之排程，反而可以得到更短的需求總等待時間，因此優先修復需求較大的節線之直覺排程決策亦不見得會有較短的總體等待修復時間。

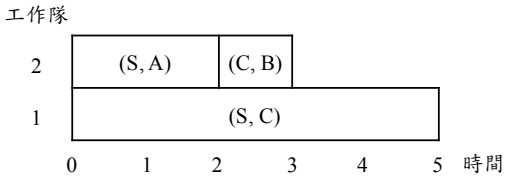


圖 1.5: 先考慮需求之修復順序排程

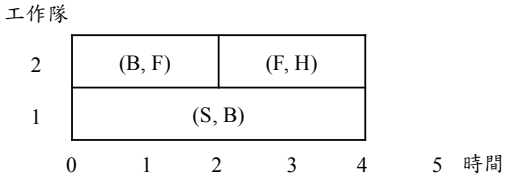


圖 1.6: 觀察網路後之修復順序排程

同理，優先處理修復時間較短的節線之直覺排程決策亦不見得有利。如圖1.7所示，此排程所修復的節線在時刻4之前皆未與供給點連通，亦即不會有任何需求被滿足，在此例看起來也不是一個好選擇。

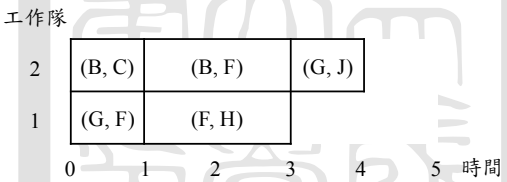


圖 1.7: 先考慮時間較短之修復順序排程

由上述簡例可知，常用的排程直覺在本問題中經常無法導致總體等待需求的時間最短，因此如何有系統地分析本問題、建立嚴謹的數學理論分析模式，並設計出效率與效能兼具的排程演算法，爲本研究主要的目標。

1.3 研究範圍

本研究的研究範圍如下：

1. 建立一個多組工作隊修復管線的搶修排程，目標爲極小化修復所有毀損節線之

總體等待時間，以使全部災民能越快獲得物資供給越好。

2. 考慮工作隊能力相同或不同時，修復數種不同形狀與大小之管線網路，測試不同數學規劃模式與排程演算法的求解效能與效率。

1.4 論文架構

本論文之架構如下：第二章爲文獻回顧，針對INDS以及RCPSP作介紹。第三章將討論問題並提出網路縮減的機制，接著針對單一工作隊的特例，提出基於RCPSP所建立的數學模型，以及針對多組工作隊提出基於INDS所建立的整數規劃模式，及其修正後的整數規劃模式及有效不等式 (valid inequalities)。第四章會先提出一分段式啓發演算法加速模型求解，針對單一工作隊的特例提出演算法，再將其節線排程規劃修改用於指派多組工作隊，以及基因演算法的設計。第五章爲數據測試分析，討論數學規劃模式以及各式排程演算法在效率以及效能上的差異。第六章總結本論文並建議未來研究主題。

第二章　文獻回顧

網路修復問題之相關文獻大致分佈在工業工程、作業研究、災後管理、通訊網路等領域，本章將先介紹整合網路設計與排程問題 (INDS)的相關文獻，說明本研究探討之議題與其相關性，接著回顧具資源限制下之專案排程問題(RCPSP)。

2.1　整合網路設計與排程問題 (INDS)

Nurre and Sharkey (2014) 將問題依其本質分成最大流量、最短路徑以及最小成本流量等三大類，決定如何派遣工作隊去修復因災害造成毀損的節線，以滿足不同設定的目標，統稱此類問題爲「整合網路設計與排程問題」 (Integrated Network Design and Scheduling Problem，INDS)，並證明上述提出的三大類問題皆爲*NP-hard*(將於2.1.1節說明)，其定義如下：

假設有一網路$G_t = (N, A_t, A'_t)$，其中N爲節點集合，包含供給點 (supply nodes) 子集合S ($S \subseteq N$)，以及需求點 (demand nodes) 子集合D ($D \subseteq N$)；A_t爲在時刻t時未毀損的節線集合，A'_t爲在時刻t時仍毀損的節線集合，$A'_t \cup A_t$即是所有節線的集合，而$|A'_t \cup A_t|$爲常數；爲了方便起見，每一條節線皆有其容量上限u_{ij}、成本c_{ij}及其對應的修復時間P_{ij}。假設任一條已毀損之節線$(i,j) \in A'_t$皆可在工作隊維修後，變成一條未毀損之節線$(i,j) \in A_{\bar{t}}$，其中$\bar{t} > t$。評估此網路的好壞即是根據問題目標以一函數$f(G_t)$來決定，主要的影響著重在A_t。若現在共有m個能力相同的工作隊P_m，每個工作隊必須做完手頭工作後才能繼續下個工作，且修復節線(i,j)所需時間爲P_{ij}；一旦開始修復某節線(i,j)，只能持續修復P_{ij}時間，修復中途不得中斷或暫停。INDS問題與傳統排程問題較不同的是，毀損節線間並沒有硬性規定的先後修復順序，每個工作隊可在其閒暇時任選其可修復的節線進行修復任務，且因爲工作隊的移動時間遠小於其修復任一節線所需時間，因此假設節線間的移動時間短到可被忽略不計。

而在目標式部分，通常可以整理成「累計式」(*Cumulative*)和「極值式」($C_{max} - Threshold$)兩大類型，其中*Cumulative*通常代表在一段時間內所需要衡量的目標累積

表現，譬如目標式可用$\sum_{t=1}^{T} w_t f(G_t)$表示，其中$w_t$爲每時刻$t$的權重，再依問題本質來計算極小或極大化該目標值；$C_{max} - Threshold$則假設我們有一個理想的門檻數值$n$，並希望將達到門檻值$n$的時刻$\bar{T}$越早越好。換句話說，欲計算最小的$\bar{T}$值以儘快達成$f(G_{\bar{T}}) \geq n$ (求解極大化目標之問題)或$f(G_{\bar{T}}) \leq n$ (求解極小化目標之問題)。

前述所提及$f(G_t)$會根據不同的問題而有不同的衡量方式，主要分成以下四種類型：

- **最大流量問題(Maximum flow，MF)**：在每條節線皆有其容量限制之下，希望將供給節點到需求節點的流量極大化。
- **最小成本流量問題(Minimum cost flow，MCF)**：在每條節線皆有其成本與容量限制之下，目標即爲極小化成本使供給節點可以提供需求給需求節點。
- **最短路徑問題(Shortest path)**：此類型問題又可以再細分成三種類型：
  - **單一起訖點組合(Single-source, single-destination，SP)**：目標爲找到從單一起點到另一個訖點的最短路徑，問題的轉換也可以是從一起點集合內的任一節點到一訖點集合內的任一節點的最短路徑。
  - **多訖點之最短路徑加總(Sum of shortest path lengths to multiple destinations, $\sum SP$)**：目標爲找到從單一起點到另一訖點集合內的所有節點的距離加總最小。其問題也可以轉變成：從一起點集合內的任一節點到一訖點集合內的所有節點之距離加總最小。
  - **從中心到多訖點之最短路徑(Shortest path centering among multiple destinations，SPC)**：在有眾多訖點集合的情況，極大化起點到訖點集合內的所有節點最短距離加總之後，再極小化該值。因此，通常該起點會位於網路圖的中心。
- **最小展開樹問題(Minimum spanning tree，MST)**：目標爲找到一棵樹狀結構的網路圖使其可以連通所有節點且樹上節線成本加總最小。

結合目標式與問題類型，Graham et al. (1979) 針對排程問題提出一表示方

式，$\alpha|\beta|\gamma$，其中α表示工作隊類型或數量，β表示問題類型或是限制，γ表示目標函式類型。整理上述所整理的問題類型於表2.1，其中P_m表示多個工作隊。

表 2.1: INDS 相關問題整理

$P_m\|MF\|Cumulative$	$P_m\|MF\|C_{max}-Threshold$
$P_m\|MCF\|Cumulative$	$P_m\|MCF\|C_{max}-Threshold$
$P_m\|SP\|Cumulative$	$P_m\|SP\|C_{max}-Threshold$
$P_m\|\sum SP\|Cumulative$	$P_m\|\sum SP\|C_{max}-Threshold$
$P_m\|SPC\|Cumulative$	$P_m\|SPC\|C_{max}-Threshold$
$P_m\|MST\|Cumulative$	$P_m\|MST\|C_{max}-Threshold$

2.1.1 $P_m|\sum SP|C_{max}-Threshold$的NP-hard證明

Nurre and Sharkey (2014) 有針對表2.1所列出的問題證明皆爲*NP-hard*問題，其中$P_m|\sum SP|C_{max}-Threshold$與本篇論文具有相關性故特別提出討論，在3.3小節也會詳加介紹如何將$P_m|\sum SP|C_{max}-Threshold$的INDS問題在多項式時間內轉換成本研究其中一個例子，證明如下：

證明 $P_m|\sum SP|C_{max}-Threshold$ **爲***NP-hard***問題：**

首先，集合覆蓋問題 (Set cover problem)可以經過轉換後變成$P_m|\sum SP|C_{max}-Threshold$。集合覆蓋問題的定義如下：假設有一有限集合群$A_1,\ldots,A_k$以及元素$e_1,\ldots,e_n$，每一個元素可以隸屬於多個集合，而$\cup A_j=\{e_1,\ldots,e_n\}$。集合覆蓋問題希望能夠找到一個最小的子集合群$B_h\subseteq\{A_j\}$，且$\cup B_h=\{e_1,\ldots,e_n\}$，換句話說，$\{B_h\}$涵蓋所有的元素$e_i$且$e_i$至少出現一次。因爲 Karp (1972) 證明集合覆蓋問題爲*NP-hard*問題，若集合覆蓋問題可轉變爲$P_m|\sum SP|C_{max}-Threshold$且$P_m|\sum SP|C_{max}-Threshold\in NP$，那麼$P_m|\sum SP|C_{max}-Threshold$也是*NP-hard*問題。

示例圖如圖2.1，集合A_j和元素e_i皆爲節點，c_{ij}表示節線(i,j)的成本，P_{ij}表示節線(i,j)的修復時間。其中未毀損的節線(A_j,e_i)的成本爲0、修復時間爲0；設立一

個超級節點 (super node) $\bar{S}$，與之連結的毀損節線$(\bar{S},A_j)$的成本爲0、修復時間爲1；而$(\bar{S},e_i)$的成本爲1、修復時間爲0。此$P_m|\sum SP|C_{max}-Threshold$的INDS問題可以想成如下：訂定一個門檻值0，極小化從超級節點$\bar{S}$到所有元素e_i的成本加總到達此門檻值的時間，若無修復毀損節線的話，就必須走$(\bar{S},e_i)$而需負擔1單位的成本。因爲選取毀損節線的修復亦可以想成是選取集合a_j，而a_j會有可以連通的e_i群，如何選擇最少的A_j來覆蓋所有的e_i便是典型的集合覆蓋問題；換句話說，選擇最少的A_j即代表花最少的時間進行修復，而連通所有的e_i也等同於$\bar{S}$到所有的e_i成本加總可爲0。

因爲可以將集合覆蓋問題做轉換，又可以在多項式時間檢查是否有完全覆蓋，亦即$P_m|\sum SP|C_{max}-Threshold \in NP$，故$P_m|\sum SP|C_{max}-Threshold$的INDS問題爲*NP-hard*。

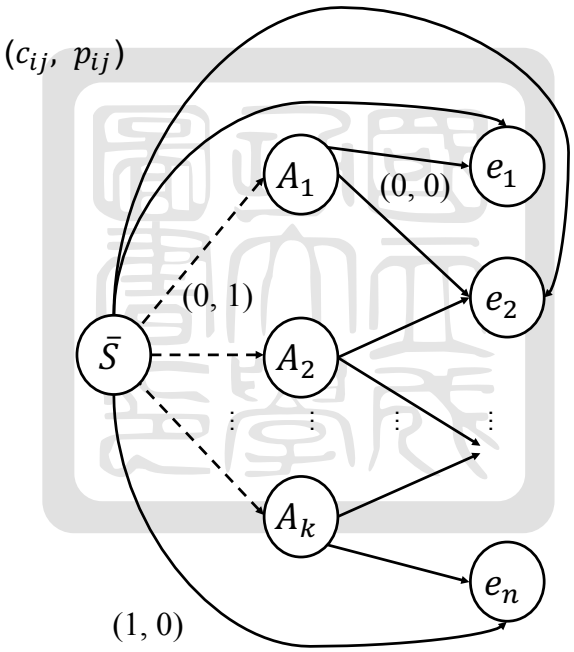


圖 2.1: 待修復網路圖

2.1.2 INDS相關文獻

由前述提及的 Graham et al. (1979) 針對排程問題所提出$\alpha|\beta|\gamma$的表示方式，可以將過去文獻整理如下。

Averbakh (2012)探討$P_m|MCF|Cumulative\ \&\ C_{max}-Threshold$，假設每個工作隊的修復速度已知且固定，並允許多組工作隊合作修復同條節線，合作修復速度爲個別修復速度加總，意即具有可加性。舉例來說，若工作隊A的修復速度爲每小時0.7公里、工作隊B則爲每小時0.8公里，那如果A和B一起修復的話即爲每小時1.5公里；該研究不考慮工作隊的移動時間且規定工作隊只能在已修復網路上移動，考慮的目標則有極小化所有節點的打通時刻總和(即總體等待時間)，以及極小化所有節點皆打通的最晚時刻(即makespan)。此篇論文提出一個動態規劃的方法，但是限定網路結構爲不含迴圈的拓樸架構 (cycle-free topology，亦即連通各個節點的路徑爲唯一)，該動態規劃演算法可以在多項式時間內完成求解；此外，Averbakh (2012)也證明如果各工作隊的修復能力不同的話，該問題將變成*NP-hard*。

Averbakh and Pereira (2012) 探討 $1|Node\ Connectivity|Cumulative$，求解網路重建流量問題 (Flowtime Network Construction Problem, FNCP)，與 Averbakh (2012) 的差別在於其處理的網路沒有特定的結構，並證明即使僅考慮單一工作隊，該問題仍爲*NP-hard*。該研究提出三種混整數規劃模式 (MILP)：模式一利用單一商品流 (single-commodity flow) 的建模方式建立最小展開樹，使每一個節點都會收到一單位的流量以確保與已修復節點保持連通；模式二利用由已修復節點往外連通的展開樹的方式建立數學模式；此兩種模式皆將所有節點的權重視爲相同; 模式三則修改前兩種模式以適用各節點有不同權重之問題。此外，他們亦設計方法推估上界與下界，以使用分支界限法求解。

Kalinowski et al. (2015) 探討 $1|MF|Cumulative$，假設每條節線各有其容量上限，且每一期僅修復一條節線，目標爲當達到最大流量且時刻爲T時，第1到T期的流量總和極大化。此問題簡化修復方式，假設每一期僅能修復一條節線，提出兩種整數規劃模式：模式一的效率與T值的估計精確度息息相關，若T值估算太過不準，則將耗費不少求解時間；模式二則是利用流量的方式建模，大幅改善求解效率。此外，

該研究亦提出三種啓發式演算法：方法一將盡量使每次增加的流量越大越好；方法二先計算要達到一流量值需要修復多少節線，再將欲修復節線數設爲限制來計算最佳的修復排程；方法三則結合方法一和二，確定每一次能夠增加的流量最大值所需修復的節線數，再將欲修復節線數設爲限制來計算最佳的修復排程。

Baxter et al. (2014) 探討 $1|SP|Cumulative$，假設每一期僅修復一條節線，欲求最佳的修復排程，以極小化時限內將各期某$s-t$最短路徑成本的加總。該研究並證明該問題爲*NP-hard*，提出求解演算法，再與整數規劃模式比較。根據測試結果，該演算法求解隨機產生的網路圖時，皆有不錯的表現；然而，若其網路架構或者節線的成本配置有某些特殊設定(譬如具有對稱性)時，該演算法有可能必須花許久時間才能收斂求解。該研究並建議未來可再進一步探討已被修復的節線會隨著時間產生損毀而需要再維護的研究議題。

Matisziw et al. (2010) 探討 $1|MCF|Cumulative\ \&\ cost$，假設節線與節點皆會損壞且各有其修復成本，每一期皆會有節線與節點個別的修復預算上限，利用預算上限的方式限制每個時間點能夠修復的節線與節點個數。該研究提出一個多準則整數規劃模式，綜合考慮起訖點組合 (origin/destination pairs) 間的流量是否能到達，以及整體的系統成本，並以電信通訊爲應用舉例，將路由器視爲節點、連接路由器的骨幹線路視爲節線，如果有多個起訖點組合需要互相連通時，應如何以最小系統成本來修復損壞的路由器以及骨幹以達成起訖點組合間的線路連通。

Xu et al. (2007) 探討 $1|MCF|Cumulative\ \&\ C_{max}-Threshold$，考慮電網在地震過後的修復問題，提出一個整數隨機規劃模式，會根據隨機發生的不同強度地震，以目標爲極小化每個使用者無法使用電的平均時間來決定電網的修復排程以及災害的評估。因爲數學模式的變數具有隨機性，如果將其隨機的可能性變成各種情境組合來求解，將必須求解大量的多階層隨機規劃問題 (multistage stochastic program)，耗時過久，因此他們提出一個基因演算法 (Genetic Algorithm) 求解之。

Cavdaroglu et al. (2013) 探討 $P_m|MCF|Cumulative$。主要考慮不同設施間的相關鏈結，假設一個遭到災害而損毀的城市需要由警察、消防員和醫院等單位一起協助修復，那麼除了通訊管線需要連通以供聯繫之外，如果電力系統沒有完整，一樣會造

成某些地方無法聯繫，故除了滿足設施內部的管線修復之外，設施間的鏈結影響也是重要的考慮點。該研究除了提出一個整數規劃模式之外，也提出一個類似於 Nurre et al. (2012) 提出的派工法則之貪婪演算法，同樣地也是先決定出節線的修復順序，再試圖將各修復任務分派給各工作隊，直到所有的需求節點皆被滿足。該研究特別強調其演算法之效率奇佳，故可供沒有超級電腦的修復單位使用。

Huang (2015) 探討 $P_m|Node\ connectivity|Cumulative\ \&\ C_{max}-Threshold$。主要考慮災後的道路網路修復問題，假設需求分佈於節點上，不需修復所有毀損節線，主要考量修復方式能儘快讓各個節點可連通至供給點。此外，因爲節線即爲道路，因此欲修復某條毀損節線勢必需要能自供給點連通該節線的某端點，而該端點在當下是否能與供給點連通則倚賴其與供給點間是否存在一未毀損(或已完成修復)的路徑，因此某毀損節線是否可開始被修復將與先前的修復決策十分相關，這跟本研究中任一毀損節線可隨時被開始修復的假設極爲不同。該研究說明其目標若爲極小化makespan，則可在多項式時間內求解；然而若目標爲極小化整體等待修復時間之總和，則爲*NP-hard*。此外，該研究有探討雇用單一或無限組工作隊之特例時，可將其轉換成最小展開樹問題或最短路徑樹問題求解。除了提出整數規劃模式之外，該研究亦提出分段式啓發演算法(SSH)及粒子群演算法以因應網路規模變大造成求解緩慢的問題，測試結果顯示SSH可在極短時間得到與最佳解品質相近的解，而粒子群演算法則表現不佳。因此，本研究於第四章將以此SSH爲基礎，來設計適合求解本研究問題之SSH演算法。

表 2.2: INDS 文獻比較

	多工作隊	滿足需求	問題類型	需求
Averbakh (2012)	✓	✓	$P_m\|MCF\|$ $Cumulative\ \&\ C_{max} - Threshold$	節點
Averbakh and Pereira (2012)		✓	$1\|Node\ Connectivity\|$ $Cumulative$	節點
Kalinowski et al. (2015)			$1\|MF\|Cumulative$	節點
Baxter et al. (2014)			$1\|SP\|Cumulative$	節點
Matisziw et al. (2010)		✓	$1\|MCF\|Cumulative\ \&\ cost$	節點
Xu et al. (2007)		✓	$1\|MCF\|$ $Cumulative\ \&\ C_{max} - Threshold$	節點
Cavdaroglu et al. (2013)	✓		$P_m\|MCF\|Cumulative$	節點
Nurre and Sharkey (2014)	✓		$P_m\|MCF,SP,MF\|$ $Cumulative\ \&\ C_{max} - Threshold$	節點
Huang (2015)	✓		$P_m\|Node\ connectivity\|$ $Cumulative\ \&\ C_{max} - Threshold$	節點
本研究	✔	✔	$P_m\|Arc\ connectivity\|$ $Cumulative$	**節線**

表2.2爲相關文獻的整理，幾乎所有文獻皆假設需求分佈在節點而非節線上，並討論諸如最大流量、最小成本流量、最早節點打通時刻等不同類型的修復問題。本研究與過往文獻最主要有兩個不同點：(1)把需求分佈設定在節線而非節點上、(2)必須修復完所有(而非部分)節線以滿足需求，並考慮多工作隊的情況。

2.2 具資源限制下之專案排程問題 (RCPSP)

在「專案排程問題」(Project Scheduling Problem，PSP)中，一個專案通常含有多項子任務(Activity、Task、Process)，而這些子任務之間會有一些先後順序(Precedence Relationship)，若將子任務視爲節點、滿足立即的先後順序之兩節點間關係視爲一條節線，則可畫出一個網路圖(如圖2.2所示)，此類網路圖通稱爲PERT圖，其中PERT爲「計畫評核術」(Program Evaluation and Review Technique，PERT)的簡稱。以圖2.2爲例，工作3必須在工作1和2完成之後才能進行，工作7必須在工作4、5、6皆完成才能進行，其中工作1和2並沒規定何者必須先被執行。由PERT圖可推估專案最少需花多少時間才能做完，然而此類時間估算通常都未考慮實際的資源限制(亦即假設有無限多資源)。將實際執行各子任務會耗費的資源(人力、電、經費等)列入考慮的專案排程問題，稱爲「具資源限制下之專案排程問題」(Resource Constrained Project Scheduling Problem，RCPSP)，本研究所探討的維修排程，其實就是一個具網路架構的RCPSP。

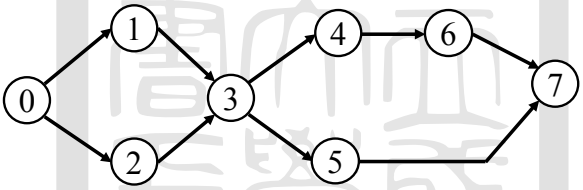


圖 2.2: PERT網路圖

RCPSP可描述如下，如果現在有M個子任務之專案，每個子任務i皆有其執行時間 (processing time) 以及先行子任務 (predecessor activities)，如果將先行子任務視爲一個集合P_i，那麼欲進行子任務i必須將P_i內所有的子任務皆完成後才能進行；且在專案中有K種資源，且每一種資源皆有其上限R_k，每一個子任務i對每一個資源k有需求r_{ik}。故可以簡單定義RCPSP即爲「設法將一種以上的資源隨著時間的進行分配至專案中之子任務，以達到專案的目標」，通常專案的目標可能是極小化專案需求的總等待時間、極小化專案最晚完工時間(makespan)、極大化資源使用率等，Blazewicz et al. (1983) 證明RCPSP爲*NP-hard*問題。

Coelho and Vanhoucke (2011) 參考Talbot (1982)，提出一整數規劃模式以求解具

「多種模式(mode)」之資源限制下的專案排程問題 (multi-mode resource-constrained project scheduling problem，MRCPSP)。其與RCPSP最大的不同在於RCPSP各子任務在任何時間點之執行時間及資源需求量爲已知，而MRCPSP則再進一步考量不同模式的作業方式(譬如雇用不同能力的工作隊，或其考慮合作組合等)將耗費不同程度的資源及執行時間。如果專案中有編號爲1至n的n個子任務，每個子任務有多種作業方式(即模式)，且每一子任務一旦開始執行就不可中斷，專案中之有限資源可以分爲「可更新資源」(譬如人力資源、機械設備等)以及不可更新資源(譬如資金、原物料等)，假設目標則爲極小化總體完工時刻，符號定義與數學模式定義如下。

2.2.1 符號定義

參數

N	各子任務之集合，包含虛擬起始節點 (dummy start node) 以及虛擬結束節點 (dummy finish node) ，編號分別爲0和n+1
A	各節線之集合，每一個節線表示兩兩子任務間之先後順序
R^r	可更新資源之集合
R^n	不可更新資源之集合
a_k^r	每一個時間點可更新資源k之資源使用上限
a_l^n	整個專案下不可更新資源l之資源使用上限
M_i	子任務i之工作方式集合
d_{im}	子任務i在工作方式m下的作業時間
r_{imk}^r	子任務i在工作方式m下對可更新資源k之需求量
r_{iml}^n	子任務i在工作方式m下對不可更新資源l之需求量
T	專案完工時間之上限
es_i	子任務i之最早可執行時刻
ls_i	子任務i之最晚可執行時刻

決策變數

x_{imt}　　若子任務i在時刻t以工作方式m開始執行則爲1，反之爲0

2.2.2 數學模式

因爲目標爲極小化總體完工時刻，故目標式如式2.2.1，其中編號n+1爲虛擬結束節點。

$$\text{Min} \sum_{t=es_{n+1}}^{ls_{n+1}} tx_{(n+1)1t} \tag{2.2.1}$$

限制式2.2.2表示每一個節線(i, j)因爲有工作的先後順序，故子任務j的開始時刻一定會大於或等於子任務i之完工時刻。

$$\sum_{m=1}^{M_i} \sum_{t=es_i}^{ls_i} (t+d_{im})x_{imt} \le \sum_{m=1}^{M_j} \sum_{t=es_j}^{ls_j} tx_{jmt} \qquad \forall (i,j) \in A \tag{2.2.2}$$

限制式2.2.3表示每一個子任務只會被一種工作方式在某一個開始時刻下執行。

$$\sum_{m=1}^{M_i} \sum_{t=es_i}^{ls_i} x_{imt} = 1 \qquad \forall i \in N \tag{2.2.3}$$

限制式2.2.4限制可更新資源的使用量不可超過可更新資源使用上限。

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{imk}^r \sum_{s=max\{t-d_{im},es_i\}}^{min\{t-1,ls_i\}} x_{ims} \le a_k^r \qquad \forall k \in R^r, t = 1,\ldots,T \tag{2.2.4}$$

限制式2.2.5限制不可更新資源的使用量不可超過專案下之不可更新資源的使用上限。

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{iml}^n \sum_{t=es_i}^{ls_i} x_{imt} \le a_l^n \qquad \forall l \in R^n \tag{2.2.5}$$

限制式2.2.6則是定義變數爲0、1變數。

$$x_{imt} \in \{0,1\} \qquad \forall i \in N, m = 1,\ldots,M_i, t = 1,\ldots,T \tag{2.2.6}$$

此整數規劃模式雖然是MRCPSP，但是只要將每個子任務的工作方式設爲一種，即可以變成RCPSP，跟本研究目前的假設較爲接近。而因爲RCPSP有節線修復的先後順序，故此性質會與本研究假設之節線皆可以在任意時刻開始進行修復有所不同，但仍適用於單一工作隊特例，更多細節將會在3.5節說明。

2.3 小結

本章節回顧整合網路設計與排程問題(INDS)並說明該問題在$P_m|\sum SP|C_{max}-Threshold$情況爲*NP-hard*，而在眾多文獻中，討論最大流量、最小成本流量、最短*s-t*路徑等問題時，皆把需求設在節點而非節線上，本研究將主要探討需求在節線，且必須將所有毀損節線完全修復的情況。而在單一工作隊的情況，因爲可以轉換成具資源限制下之專案排程問題(RCPSP)，討論其問題型式之後，也一併了解其建立整數規劃模式的方式，加以運用在本研究中。下章節將針對本研究欲探討的問題提出整數規劃模式。

第三章　網路節線修復問題之整數規劃模式

本章將先描述欲探討的網路修復問題，說明其爲*NP-hard*；再提出一個網路縮減機制以大幅改善數學規劃模式的求解效率。接著針對單一工作隊的特例以RCPSP的角度提出一整數規劃模式M^s；再針對多組工作隊的網路修復問題提出兩個整數規劃模式$M^m_{B\&C}$以及其修正模式M^m_{MCF}，最後再修改模式M^m_{MCF}使其能處理工作隊能力不同的情況。

3.1　問題描述

假設有一網路$G=(N,A_e,A_p)$，N爲所有節點的集合，其中節點S爲供給節點，A_e爲沒有毀損的節線，A_p爲毀損的節線。所有節線皆爲無向，各節線之物資需求量以及修復時間皆爲已知，若節線無法與供給節點連通，則該節線上的物資需求將不會被滿足；假設總共有K組工作隊負責維修管線，各節線的維修工作不得中途暫停(亦即其維修之工作隊一旦開始維修，則必須等到該節線被修復才能休息)，不考慮工作隊的合作(亦即每條節線只會被一個工作隊所修復)，目標爲極小化總體需求的等待時間，也就是將每單位需求量的連通時刻加總越小越好。

以圖3.1爲例說明單一工作隊可能的修復排程：假設實線爲未毀損的節線，虛線爲毀損的節線，每一條節線(i,j)皆有其需求D_{ij}以及修復時間P_{ij}；現在若僅一個工作隊負責維修此網路，若一開始挑選節線(A,B)進行修復，兩單位時間過後，可以得到如圖3.2的網路圖。

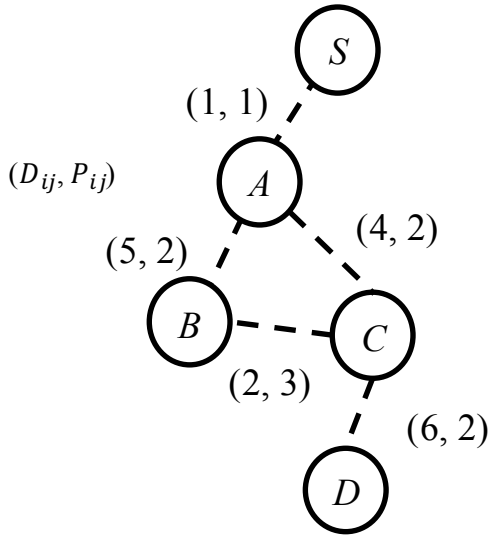


圖 3.1: 待修復之網路圖

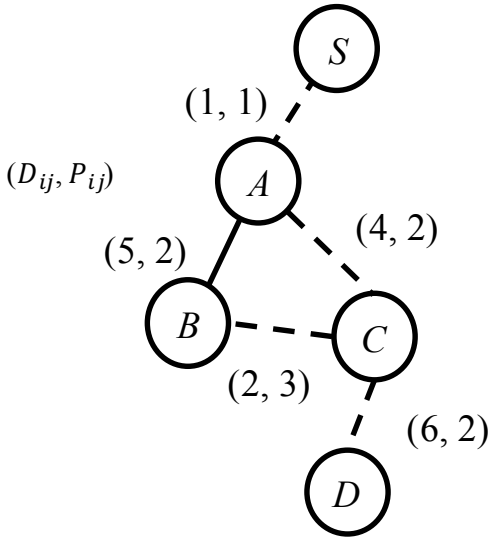


圖 3.2: 修復節線(A,B)後之網路圖

可以發現，因爲本研究探討的問題有如開水龍頭流入管線，所以雖然節線(A,B)被修復，上面的5單位需求因爲仍未與供給點S連通，因此仍然未被滿足。

由上例可以看出，此問題需要決定節線的修復順序，以及如何分配工作隊在什麼時刻去修復哪一條節線，此即爲本研究欲求解的排程決策。

3.2 問題假設

爲了適當地簡化數學模式以及模型的使用限制，本研究有下列基本假設：

1. 假設網路爲connected(即所有節點皆連結在一起)，物資需求量分佈在節線而非節點上，只要管線經過的地方便會涵蓋需求。

2. 假設經過嚴密的檢查以及估算，已經定位出所有的毀損節線，各節線上的需求量、修復各毀損節線所需的資源耗費量(可能以人工小時或特殊機具等表示)、以及各組工作隊的效率、能力與裝備等，使決策單位可以在已知這些資訊之下，整體考量而下正確的排程決策。

3. 一旦管線修復且可與供給點連通，即可立即滿足該管線上的需求量，不會受到

如地形高低差或水壓不夠等問題影響。

4. 假設每組工作隊在進行搶修時，一定會將現在的工作完成才會進行下一個工作，不會有工作到一半去休息或修復其他管線的情況。

5. 每組工作隊的能力不會受到現實情況的影響，皆可以在決策單位訂定的修復時限內完成任務，不需要考慮補充能量、睡覺等情況。

6. 管線間沒有事先規定的修復先後順序(節線之間沒有任何的優先權)，因此工作隊可隨意挑選任一條管線進行修復。

7. 假設工作隊在節線間的移動時間遠小於各個管線的修復時間，因此可忽略之(將工作隊的移動時間視爲零。)

3.3 *NP-hard*說明

2.1.1節Nurre and Sharkey (2014)已證明$P_m|\sum SP|C_{max}-Threshold$爲*NP-hard*。如果能將$P_m|\sum SP|C_{max}-Threshold$的INDS問題在多項式時間內轉換成本研究問題其中一個例子，則代表本研究問題亦爲*NP-hard*。

我們可大致沿用2.1.1節的相關定義，但先將所有互相連通的未毀損節線的需求集合濃縮成一點(即爲元素e_i)，這些e_i之間並未互相連通(否則即可再進一步濃縮)，而與原證明稍稍不同的是$(\bar{S}, e_i)$的成本會是濃縮後的所有成本，定義爲w_i；並且假設所有毀損節線皆無成本，即可以將$P_m|\sum SP|C_{max}-Threshold$的INDS問題轉換爲本問題。

示例如圖3.3所示，方式如2.1.1節，因爲可以將集合覆蓋問題轉換成$P_m|\sum SP|C_{max}-Threshold$，又可以轉換成本問題，故本問題同樣爲*NP-hard*。需要注意的是，因爲本研究探討的問題是無向節線(或可視爲雙向有向節線)，若僅欲表示單向節線(A_j,e_i)，只需設定其反向節線(e_i, A_j)有一個極大的成本，而其餘反向節線皆可以將成本和修復時間設定爲零，即可順利表示之。

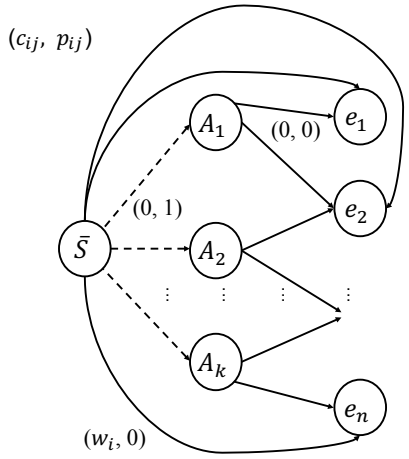


圖 3.3: 本問題轉換集合覆蓋問題

3.4 網路縮減機制

在災難過後，不見得所有的管線皆會毀損，如果可在網路中找到一未毀損且連通的子網路$G'=(N', A')$，則可進一步將G'濃縮成一個具loop的節點，而將那些子網路上未毀損的節線A'上的所有需求集中分佈在該loop上。如圖3.4的節線(A, B)、(A, C)、(C, D)、(B, D)所示，其中未毀損節線以實線表示，毀損以虛線表示。故當把節線(S, A)修復，也等同把節點A、B、C、D所構成的節線打通，所以可以將G'合併爲一點A，而該4條節線的全部10單位需求可加總集中在A的loop上。

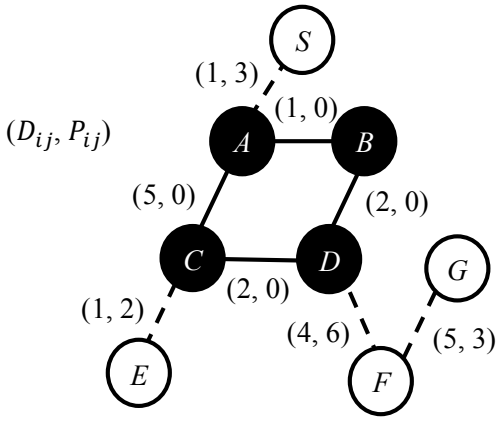


圖 3.4: 已修復網路合併前

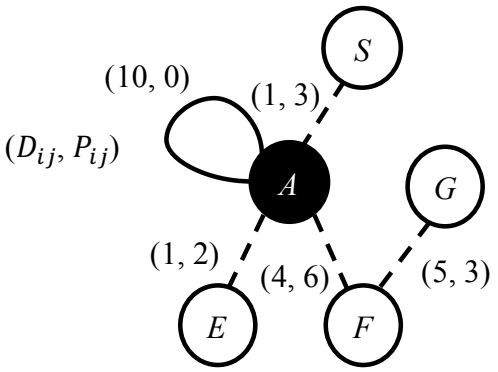


圖 3.5: 已修復網路合併後

網路縮減可以將節點以及節線數減少，有利於模型與演算法的求解效率，以下介紹其步驟：

[步驟一] 檢查是否有未毀損節線(i,j)，且$i \neq j$、$P_{ij}=0$，若有則到步驟二，若無則結束網路縮減。

[步驟二] 將節點i及節點j融合爲一點(假設$i<j$):

如果沒有未毀損loop(i,i)，則建立之且將$D_{ii}:=D_{ij}$，$P_{ii}:=0$。

如果有未毀損loop(j,j)，則將$D_{ii}:=D_{ij}+D_{jj}$，若無則將D_{ij}(及$D_{jj'}$，若有loop$(j,j)'$未修復)加入進去D_{ii}。

接著針對相鄰j的所有節線(k,j)(或(j,k))，建立(k,i)(或(i,k))，並且設定$D_{ki}:=D_{kj}$、$P_{ki}:=P_{kj}$(或$D_{ik}:=D_{jk}$、$P_{ik}:=P_{jk}$)。

[步驟三] 將連向節點j的所有節線移除，並回到步驟一。

依此步驟將圖3.4的網路可以縮減成如圖3.5的網路。此縮減方式也影響到單一工作隊演算法裡的貪婪樹生成法，將會在4.1.2章節介紹。

3.5 單一工作隊特例適用RCPSP之推論

在單一工作隊的情況，因爲每次只能修復一條節線，可以發現最佳解的節線修復排程一定會選擇從已連通網路邊界的毀損節線開始修，故在單一工作隊時可以使用基於RCPSP的數學模式求解，以下我們用反證法推論此排程特性。首先我們先定義「邊界毀損節線」爲僅一端可與供給點連通之毀損節線，而「非邊界之毀損節線」則爲兩端點皆未能與供給點連通之毀損節線。依此定義，若修復一條「邊界毀損節線」，其上的需求量必定可被滿足，並可能進而將其鄰近的「非邊界之毀損節線」轉變成「邊界毀損節線」；反之，若修復一條「邊界毀損節線」，其上的需求量必仍無法被滿足，亦不會影響其鄰近其它「非邊界之毀損節線」的連通狀態。接著我們說明爲何在單一工作隊的最佳修復排程一定是每回合皆選取一條當下的邊界毀損節線來修復，直至所有毀損節線完全修復爲止。

首先，我們解釋最佳排程一定會以修復某條「邊界毀損節線」做結尾，否則若最後修復的節線爲「非邊界之毀損節線」(i,j)的話，那節點i與j即使在修復後仍無法連通至供給點，代表該網路並非connected，與我們的問題假設矛盾。因此，無論如何，修復過程中至少會以某條「邊界毀損節線」做結尾，亦即若修復過程中曾選取某些「非邊界之毀損節線」來修復的話，在其後一定至少仍會選取某條「邊界毀損節線」來修復。以下我們就針對修復排程中可能出現的先修復「非邊界之毀損節線」a_i、再緊接著修復「邊界毀損節線」a_{i+1}的情況來討論。

假設現有一修復m條毀損節線之最佳修復排程，其修復的毀損節線編號依序列成$L=(a_1,a_2,...,a_i,...,a_m)$。假設其中$a_i$爲「非邊界之毀損節線」，但緊接其後的$a_{i+1}$爲「邊界毀損節線」(上一段已解釋若曾修復某「非邊界之毀損節線」的話，必可找到一組相鄰的a_i與a_{i+1}之組合)。倘若時刻t時修復完a_{i-1}，原先排程L的節線a_i在時刻t'才能連通至供給點，則$t' \geq t+P_{a_i}+P_{a_{i+1}}$，然而節線$a_{i+1}$在時刻$t+P_{a_i}+P_{a_{i+1}}$必能連通至供給點。如果我們將排程$L$中的$a_{i+1}$與$a_i$順序對調、其餘順序不變，可形成另一修復排程順序$L'=(a_1,a_2,...,a_{i-1},a_{i+1},a_i,a_{i+2},...,a_m)$。此時排程$L'$的節線$a_{i+1}$在時刻$t+P_{a_{i+1}}$必能連通至供給點，而節線$a_i$亦在時刻$t'$才能連通至供給點。

比對此兩個排程的總體需求等待時間，可知在修復$a_1,...,a_{i-1}$以及$a_{i+2},...,a_m$的部分因完全一樣故可抵消，僅需比對前者排程L剩餘的需求等待時間 $d_{a_i}t'+d_{a_{i+1}}(t+P_{a_i}+P_{a_{i+1}})$ 以及後者排程L'剩餘的需求等待時間 $d_{a_i}t'+d_{a_{i+1}}(t+P_{a_{i+1}})$，會發現前者會比後者多出$d_{a_{i+1}}*P_{a_i}$，亦即排程$L'$反而有比最佳排程$L$更短的總體需求等待時間，違反原先排程$L$爲最佳的假設，因此可反證原先的最佳排程不可能先修復「非邊界之毀損節線」，所以最佳排程一定是每回合皆會選取某條「邊界毀損節線」來修復，這印證了先前由一端連通供給點的毀損節線開始修復的排程直覺。

3.6 單一工作隊整數規劃模式 (M^s)

上一節推論在單一工作隊的特例可以適用RCPSP的數學規劃模式，本研究即以RCPSP的建構方式提出一整數規劃模式 M^s(其中上標s代表單一工作隊single之

意)，其想法便是使用由已與供給點連通之區域向毀損區域修復的方式，3.6.1節介紹符號定義，3.6.2節則說明數學模式。

3.6.1 符號定義

參數

A_p 毀損的節線集合

P_a 毀損節線a所需的修復時間

R_a 修復節線a所需的工作隊數 (單一工作隊時$R_a = 1$)

$\bar{R}$ 全部的工作隊總數 (單一工作隊時$\bar{R} = 1$)

T 完工時刻上限

D_a 節線a上的需求量

δ_{at} $\begin{cases} 1: & t \geq P_a \text{ and } P_a \neq 0 \\ 0: & otherwise \end{cases}$

決策變數

x_{at} 若節線$a = (i, j)$在時刻t開始修復則爲1；反之爲0

$x_{\bar{a}t}$ 若節線$\bar{a} = (j, i)$在時刻t開始修復則爲1；反之爲0

y_a 若毀損節線a已被修復則爲1，反之爲0，此變數也可用於定義維修方向。

z_{it} 若節點i在時刻t之前打通則爲1；反之爲0

f_a^A 節線a的完工時刻

f_i^N 節點i的打通(即可連通至供給點)時刻

3.6.2 數學模式

目標式爲希望將所有需求的總體等待時間加總極小化，如式(3.6.1)，毀損節線的完工時刻即爲該節線每單位需求的等待時間，且因爲是無向圖，所以節線(i, j)被修復

等同於節線(j, i)被修復，只是修復方向不一樣，而因爲RCPSP皆是從已與供給點連通之子網路往仍毀損之子網路來修復，所以不會有無法連通的問題，而M爲一個大的常數值。

$$\text{Min} \sum_{a \in A_p} D_a (f_a^A - M(1 - y_a)) \tag{3.6.1}$$

限制式部分，限制式(3.6.2)爲定義各節線的完工時刻，如果某節線尚未修復，其完工時刻將爲一大值M。

$$f_a^A = \sum_{t=0}^{T} (t + P_a) x_{at} + M(1 - y_a) \qquad \forall a \in A_p \tag{3.6.2}$$

限制式(3.6.3)代表節點i一定要先被打通，才能沿(i, j)來修復。

$$\sum_{t=0}^{T} t x_{at} \geq f_{tail(a)}^N - M(1 - y_a) \qquad \forall a \in A_p \tag{3.6.3}$$

限制式(3.6.4)爲各節線之修復方向((i,j)或(j,i))及其開始修復時刻爲唯一。

$$\sum_{t=0}^{T} (x_{at} + x_{\bar{a}t}) = 1 \qquad \forall a \in A_p \tag{3.6.4}$$

限制式(3.6.5)爲定義修復方向((i,j)或(j,i))，每一條節線只會有一種修復方向。

$$y_a = \sum_{t=0}^{T} x_{at} \qquad \forall a \in A_p \tag{3.6.5}$$

限制式(3.6.6)以及限制式(3.6.7)定義節點與供給點的連通狀態，若節點i在時刻t-1已可連通至供給點，則時刻t及其後將仍保持與供給點連通。此外，若在時刻$t - P_a$之前曾(或未曾)開始修復某條(或所有)節線$a = (j,i)$的話，那時刻t時節點i應已能(或仍無法)與供給點連通。

$$z_{it-1} \leq z_{it} = \sum_{\substack{a \in A_p, \\ head(a) = i}} \delta_{at} \sum_{s=0}^{t-P_a} x_{as} \qquad \forall i \in N, i \neq S, t = 1, \ldots, T \tag{3.6.6}$$

$$z_{it-1} \leq z_{it} \qquad \forall i \in S, t = 1, \ldots, T \tag{3.6.7}$$

限制式(3.6.8)則表示若時刻t開始修復節線(i, j)，則時刻$t+P_a$將打通節點j。

$$x_{at} \leq z_{head(a)min\{T,t+P_a\}} \qquad \forall a \in A_p, t = 0, \ldots, T \tag{3.6.8}$$

限制式(3.6.9)則規定每個節點最後一定要可連通至供給點。

$$\sum_{t=0}^{T} z_{it} \geq 1 \qquad \forall i \in N \tag{3.6.9}$$

限制式(3.6.10)則可夾擠出節點i的打通時刻。

$$t(z_{it} - z_{it-1}) \leq f_i^N \leq t + M(1 - (z_{it} - z_{it-1})) \qquad \forall i \in N, t = 1, \ldots, T \tag{3.6.10}$$

限制式(3.6.11)與限制式(3.6.12)則定義各節點的初始狀態(可與供給節點連通與否)。

$$z_{i0} = 1 \qquad \forall i \in S \tag{3.6.11}$$

$$z_{i0} = 0 \qquad \forall i \in N, i \neq S \tag{3.6.12}$$

限制式(3.6.13)限制每個時刻的工作隊數上限。

$$\sum_{a \in A_p} R_a \sum_{s=max\{t-P_a+1,0\}}^{t} x_{as} \leq \bar{R} \qquad \forall t = 0, \ldots, T \tag{3.6.13}$$

利用此整數規劃模式可以求得在單一工作隊情況下的最佳解，若是變成多組工作隊(即R_a與$\bar{R}$可大於或等於1)，此建模方式因爲限定必須從已與供給點連通之子網路往仍未連通之子網路的方向來修復，若在工作隊數超過邊界的毀損節線個數時，勢將忽略了可同時指揮多餘工作隊去修復那些未在邊界的毀損節線的可能性，所以求出的最佳解未必是原問題之最佳解。舉例來說，假設現在已連通供給點之子網路與未連通之子網路僅以兩條毀損節線相連，但當下有三個閒置的工作隊，在此整數規劃模式只會派兩組工作隊來修復，而其中一個工作隊只能閒置，明顯不符合可以派遣工作隊到任意節線修復的規則。

3.7 使用Branch-and-Cut (B&C) 之多工作隊整數規劃模式 ($M^m_{B\&C}$)

在多組工作隊的情況，本研究提出以INDS爲基礎架構的整數規劃模式$M^m_{B\&C}$(其中m代表多組工作隊，$B\&C$代表Branch-and-Cut)，針對多工作隊的情況，探討修復所有毀損管線的最佳排程，因爲多組工作隊不是單純地由已連通供給點之子網路向外修，在修復完節線計算需求總體等待時間時，還需考量是否與外界連通的問題。此整數規劃模式便是利用找尋一條從供給點出發連通至各已連通之節點的路徑以確保連通性。以圖3.6爲例，節點S、C、H、I、G所構成的節線路徑因與供給點連通，流量流通所以需求可以被滿足，但是像是節點A、B、D所構成的節線集合(構成子迴圈subtour)因未與供給點連通，但仍滿足本模式的的流量守恆限制式，所以會產生子迴圈的流量，在計算需求時需將該子迴圈流量排除。以下及爲此整數規劃模式之介紹，3.7.1節介紹符號定義，3.7.2節說明數學模式。

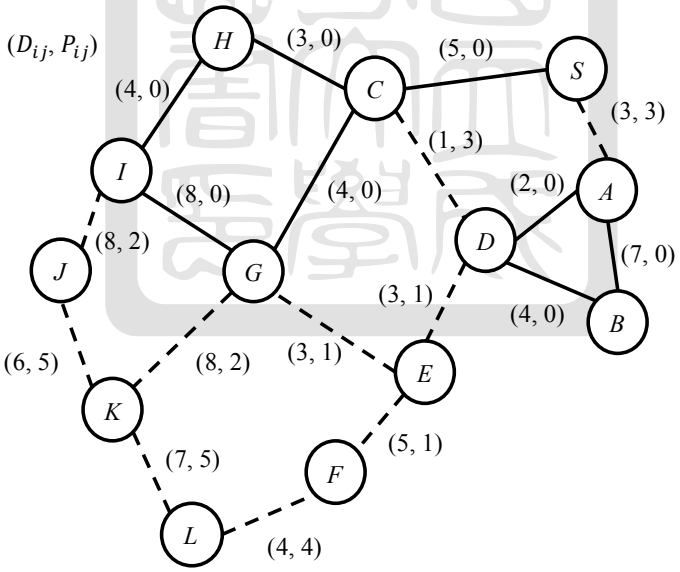


圖 3.6: 使用B&C之多組工作隊整數規劃模式想法示意圖

3.7.1 符號定義

參數

A_p	毀損的節線集合
A_e	沒有毀損的節線集合
P_a	毀損節線a的修復時間
$\bar{R}$	修復節線工作隊數上限
T	完工時間上限
D_a	節線a上的需求量

決策變數

x_{at}	在時刻t節線a是否有流量，有則爲1；反之爲0
s_{atr}	工作隊r是否在時刻t開始修復節線a，有則爲1；反之爲0
α_{atr}	工作隊r是否在時刻t完成修復節線a，有則爲1；反之爲0
z_{at}	在時刻t節線a是否已被修復，有則爲1；反之爲0

3.7.2 數學模式

目標爲希望在時限T內全部需求之總等待時間越小越好，故目標式如式(3.7.1)。

$$\text{Min} \sum_{t=0}^{T} \sum_{a \in A_e \cup A_p} \frac{D_a}{2}(1 - x_{at}) \tag{3.7.1}$$

限制式部分，限制式(3.7.2)爲流量守恆。

$$\sum_{\substack{a \in A_e \cup A_p, \\ tail(a)=i}} x_{at} - \sum_{\substack{a \in A_e \cup A_p, \\ head(a)=i}} x_{at} = 0 \qquad \forall i \in N, t = 0, \ldots, T \tag{3.7.2}$$

限制式(3.7.3)爲待修復節線需修復才能有流量。

$$x_{at} \leq z_{at} \qquad \forall a \in A_p, t = 0, \ldots, T \tag{3.7.3}$$

限制式(3.7.4)爲某方向的節線a的連通狀態(打通與否)，應與其反向節線$\bar{a}$的連通

狀態相同。

$$z_{at} = z_{\bar{a}t} \qquad \forall a \in A_p, t = 0, \ldots, T \tag{3.7.4}$$

限制式(3.7.5)限制各毀損節線a一定要在期限$T - P_a$前被某一組工作隊開始修復。

$$\sum_{r=1}^{\bar{R}} \sum_{t=0}^{T-P_a+1} s_{atr} = 1 \qquad \forall a \in A_p \tag{3.7.5}$$

限制式(3.7.6)限制各工作隊一旦開始修復某毀損節線，不可中途暫停再去修別條毀損節線。

$$\sum_{a \in A_p} \sum_{u=t}^{min\{T, t+P_a-1\}} \alpha_{aur} \leq 1 \qquad \forall t = 0, \ldots, T, r = 1, \ldots, \bar{R} \tag{3.7.6}$$

限制式(3.7.7)規定了各毀損節線的開始修復及其完成修復時刻。

$$\sum_{t=0}^{T-P_a} (t + P_a) s_{atr} = \sum_{t=0}^{T} t \alpha_{atr} \qquad \forall a \in A_p, r = 1, \ldots, \bar{R} \tag{3.7.7}$$

限制式(3.7.8)定義了毀損節線完成修復時刻之相關變數的關係。

$$z_{at} - \sum_{u=0}^{t} \sum_{r=1}^{\bar{R}} (\alpha_{aur} + \alpha_{\bar{a}ur}) \leq 0 \qquad \forall a \in A_p, t = 0, \ldots, T \tag{3.7.8}$$

限制式(3.7.9)規定待修復節線不可能在時刻0完工。

$$\alpha_{a0r} = 0 \qquad \forall a \in A_p, r = 1, \ldots, \bar{R} \tag{3.7.9}$$

限制式(3.7.10)限制各毀損節線一定要在期限T前被某一組工作隊完成修復。

$$\sum_{r=1}^{\bar{R}} \sum_{t=0}^{T} \alpha_{atr} = 1 \qquad \forall a \in A_p \tag{3.7.10}$$

限制式(3.7.11)爲「排除獨立子迴圈」 (isolated subtour elimination) 限制式，由Fisher and Jaikumar (1981)改變而成，其中P爲內部迴圈的節點集合，$|P|$爲集合內的節點個數。

$$\sum_{\substack{tail(a), head(a) \in P, \\ tail(a) \neq head(a)}} x_{at} - M\Big(\sum_{\substack{tail(a) \in P, \\ head(a) \in \bar{P}}} x_{at} + x_{\bar{a}t} \Big) \leq |P| - 1 \qquad \forall t = 0, \ldots, T \tag{3.7.11}$$

需利用 Padberg and Rinaldi (1991) 所提出的B&C技巧如演算法1所示，在每次求解過程得到可行解時，檢查是否有子迴圈未與供給點連通，若有此情況再加入限制式(3.7.11)。

Algorithm 1 B&C by Gurobi procedure

```
set up the environment
set up the variable
add the constraints 式(3.7.2) ∼ 式(3.7.10)
add the objective 式(3.7.1)
do
    solving problem
    if an feasible solution contains any subtour then
        stop model
        add subtour-elimination constraint 式(3.7.11) to model and update model
    end if
while Time limits or obtain an optimal solution
```

可以利用此整數規劃模式求得在多組工作隊時，如何派遣工作隊並決定修復節線順序，極小化需求總體等待時間。其中T值對此整數規劃模式的影響甚巨，T值越大將導致變數與限制式個數增加，求解模式將耗時甚久。在單一工作隊的情況，因爲只有一個工作隊在修復，所以T值即爲全部毀損管線的修復時間加總；但在多組工作隊時，T值則會因爲派遣工作隊的方式不同而有所變化，如果能夠預測準確的T值上下界即可大幅降低求解時間。T值如果設定太小，雖然可以增加求解速度，但可能會導致數學模式無可行解(即有節線未完成修復)；反之T值若設定太大將導致變數與限制式過多、求解時間變長。

3.7.3 有關排除獨立(未與外界連通)子迴圈機制之進階說明

車輛途程問題 (Vehicle Routing Problem，VRP) 相關文獻之數學模式經常會探討排除子迴圈的限制式，式(3.7.12)即爲 Fisher and Jaikumar (1981) 所提出之排除子迴圈

限制式 (subtour elimination constraints)，其中P爲某些不包含供給節點S之節點集合。

$$\sum_{i,j\in P, i\neq j} x_{ij} \leq |P|-1 \qquad P\subseteq\{1,\ldots,n\}/S, 2\leq|P|\leq n-1 \tag{3.7.12}$$

該限制式主要是將節點集合P限制住，使子迴圈完全不會出現在求出的解當中。如果以一台車的旅行員推銷問題 (Travelling salesman problem) 來看，求出的路徑只能如圖3.7所表示的單一迴圈，而不可出現如圖3.8的兩連結迴圈；然而就本問題而言，其實是如圖3.8的兩連結迴圈是可被允許的。因此我們不能逕行使用傳統VRP常用的排除子迴圈限制式，以免其過度限制解空間的範圍。Wu (2010)針對式(3.7.12)提出改變方式，將式(3.7.12)改成式(3.7.13)並將其命名爲「排除獨立子迴圈限制式」(isolated subtour elimination constraints)，即加入$M(\sum_{i\in P, j\in\bar{P}}(x_{ij}+x_{ji}))$項次，其中$\bar{P}$爲$P$之互補節點集合(即$P\cup\bar{P}=N$)。

$$\sum_{i,j\in P, i\neq j} x_{ij} - M(\sum_{i\in P, j\in\bar{P}}(x_{ij}+x_{ji})) \leq |P|-1 \qquad P\subseteq\{1,\ldots,n\}/S, 2\leq|P|\leq n-1 \tag{3.7.13}$$

當求解過程中存在有其他節線能夠連結至此子迴圈時，則限制式將會被放寬，使求得之解如圖3.8的情況亦可以被接受。

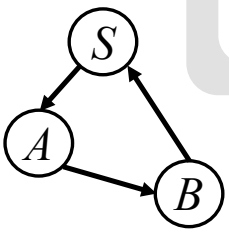


圖 3.7: 排除子迴圈限制式之可行解

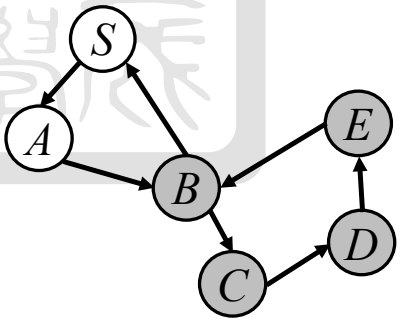


圖 3.8: 排除獨立子迴圈限制式之可行解

換句話說，此限制式有別於 Fisher and Jaikumar (1981) 所提出的限制式，並不是將所有可能出現的子迴圈皆消除，而是將無法與供給點連通的子迴圈給予限制，即此限制式只會排除獨立子迴圈 (isolated subtour)。舉例來說，若圖3.8中子迴圈的節點集合爲P，即節點B、C、D、E所構成的集合，而子迴圈以外的節點爲$\bar{P}$，若使用

式(3.7.12)，則子迴圈含四條節線及四個節點，可以得到$4 > 4 - 1 = 3$的結果，並不符合式(3.7.12)，所以在求出的解中會把此種可能消除，而求得如圖3.7的解，即不包含子迴圈P之網路；若以式(3.7.13)來解的話，因爲有一條節線(1, 2)連進P，一條節線(2, S)連出P，假設M爲1000，可以求得$4 - 1000(1 + 1) = -1996 \leq 3$，可以發現此限制合理，故可以得到如圖3.8與供給點可以相連通的解。而此排除獨立子迴圈的限制式符和本研究可以找到一條路徑走過所有已修復節線的條件，故在基於INDS的整數規劃模式使用。

3.8 使用多商品網路流量 (Multi-commodity Network Flow，MCF) 之多工作隊整數規劃模式 (M^m_{MCF})

在3.7節的多工作隊整數規劃模式，因爲需要使用B&C逐一加入限制式的方法，所以在模型求解的效率上非常不佳，本節提出使用MCF的建模方式，利用各節點的單位需求是否可被供給點滿足，以用來判斷該節點與供給點的連通性，以圖3.9爲例，由供給點提供12單位需求，各節點的-1需求如果被滿足的話，即表示該節點可與外界(即供給點)連通。譬如節點C及H的需求皆有被滿足，但即使修復節線(K,L)，節點K及L的需求仍無法被滿足；而在需求考慮上，擬將因網路縮減所造成的自體迴圈上的需求視同爲該濃縮節點的需求。以下即爲修正後的整數規劃模式M^m_{MCF}。3.8.1節定義數學模式符號，而3.8.2節則說明整數規劃模式，3.8.3節再提出四種有效不等式 (valid inequalities)，以加速求解整數規劃模式。

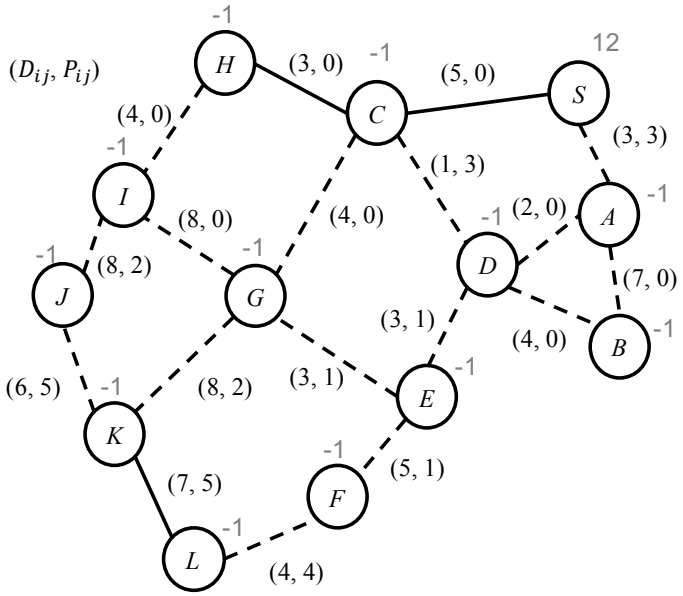


圖 3.9: 使用MCF之多工作隊整數規劃模式想法示意圖

3.8.1 符號定義

參數

A_p	毀損的節線集合
A_e	未毀損的節線集合
P_a	節線a的修復時間
$\bar{R}$	修復節線工作隊數上限
T	總體完工時刻上限
D_a	節線a上的需求量
D_i^{loop}	節點i上的需求量
IND_i	節點i的向內臨邊數 (in-degree)

決策變數

x_{at}	時刻t節線a的流量大小
α_{atr}	工作隊r是否在時刻t修復完節線a，有則爲1；反之爲0
z_{at}	時刻t節線$a \in A_p$是否已被修復，有則爲1；反之爲0 (針對$a \in A_e$，一律設定$z_{at} = 1$)

y_{at}　　時刻t節線a是否與外界連通，有則爲1；反之爲0

v_{it}　　時刻t節點i是否與外界(即供給點)連通，有連通則爲1；反之爲0

3.8.2 數學模式

目標即爲極小化時刻T內的總體需求等待時間，目標式如式(3.8.1)所示，其中$\bar{a}$表示a的反向節線。

$$\text{Min} \sum_{t=0}^{T} \sum_{\substack{a\in A_e\cup A_p,\\ tail(a)<head(a)}} D_a(1-y_{at}-y_{\bar{a}t}) + \sum_{t=0}^{T}\sum_{i\in N} D_i^{loop} v_{it} \tag{3.8.1}$$

限制式部分，限制式(3.8.2)規範了需求節點的個數上限，而(3.8.3)式爲流量守恆，與外界連接的節點會有至多一單位流量的需求，確保連通性。

$$\sum_{\substack{a\in A_e\cup A_p,\\ tail(a)=S}} x_{at} - \sum_{\substack{a\in A_e\cup A_p,\\ head(a)=S}} x_{at} \le |N|-1 \qquad \forall t=0,\ldots,T \tag{3.8.2}$$

$$\sum_{\substack{a\in A_e\cup A_p,\\ tail(a)=i}} x_{at} - \sum_{\substack{a\in A_e\cup A_p,\\ head(a)=i}} x_{at} = -v_{it} \qquad \forall i\in N, t=0,\ldots,T \tag{3.8.3}$$

限制式(3.8.4)爲待修復節線需修復才能有流量。

$$0 \le x_{at} \le (|N|-1) z_{at} \qquad \forall a\in A_p, t=0,\ldots,T \tag{3.8.4}$$

在時刻t時，限制式(3.8.5)代表若某原未毀損節線$a\in A_e$或某已被修復($z_{at}=1$)的毀損節線$a\in A_p$的tail節點已可連通外界，則該節線當下必可連通外界(if $z_{at}=1$, then $v_{tail(a)t}=1 \Rightarrow y_{at}=1$)；限制式(3.8.6)代表若某毀損節線$a\in A_p$尚未被修復，該節線必尚未連通外界($z_{at}=0 \Rightarrow y_{at}=0$)；限制式(3.8.7)代表若某節線$a$(包括原未毀損，或原毀損但已修復)已連通外界，則其tail節點必也可連通外界($y_{at}=1 \Rightarrow v_{tail(a)t}=1$)。總之，限制式(3.8.5)、(3.8.6)、(3.8.7)欲規範(y_{at}, v_{it}, z_{at})滿足以下四種可能性：(1, 1,

1),(0, 0, 1),(0, 1, 0),(0, 0, 0) (其它的四種可能組合則不用在意)。

$$y_{at} \geq v_{tail(a)t} + z_{at} - 1 \qquad \forall a \in A_e \cup A_p, t = 0, \ldots, T \tag{3.8.5}$$

$$y_{at} \leq z_{at} \qquad \forall a \in A_e \cup A_p, t = 0, \ldots, T \tag{3.8.6}$$

$$y_{at} \leq v_{tail(a)t} \qquad \forall a \in A_e \cup A_p, t = 0, \ldots, T \tag{3.8.7}$$

限制式(3.8.8)限制節線之修復方向爲唯一((i,j)或(j,i))。

$$y_{at} + y_{\bar{a}t} \leq 1 \qquad \forall a \in A_p, t = 0, \ldots, T \tag{3.8.8}$$

限制式(3.8.9)爲每條節線a剛好只會被某個工作隊在某時刻修復。

$$\sum_{a \in A_p} \sum_{u=t}^{min\{T, t+P_a-1\}} \alpha_{aur} \leq 1 \qquad \forall r \in 1, \ldots, R, t = 0, \ldots, T \tag{3.8.9}$$

限制式(3.8.10)爲若節線a在時刻t下修復(或仍未修復)，表示有一組(或完全沒有)工作隊在時刻t已完成修復該節線。

$$z_{at} - \sum_{u=0}^{t} \sum_{r=1}^{R} \alpha_{aur} = 0 \qquad \forall a \in A_p, t = 0, \ldots, T \tag{3.8.10}$$

限制式(3.8.11)和(3.8.12)規範毀損節線a在時刻P_a之前不可能被修復完成。

$$\sum_{t=0}^{P_a-1} z_{at} = 0 \qquad \forall a \in A_p \tag{3.8.11}$$

$$\sum_{r=1}^{R} \sum_{t=0}^{P_a-1} \alpha_{atr} = 0 \qquad \forall a \in A_p \tag{3.8.12}$$

3.8.3 有效不等式

以上即爲修正後的多組工作隊之基本數學模式M_{MCF}^{m}。以下提出幾組有效不等式，藉由增加有效的限制式以限縮求解區域，加速數學模式的求解收斂過程。

限制式(3.8.13)代表在時刻t時，若某(j,i)方向的節線可與供給點連通，則i在當下必亦可與供給點連通。

$$\sum_{\substack{a \in A_e \cup A_p, \\ head(a)=i}} y_{at} \leq IND_i \cdot v_{it} \qquad \forall i \in N, i \neq S, t = 0, \ldots, T \tag{3.8.13}$$

限制式(3.8.14)代表在時刻t時若節線a已被修復，則該節線在之後亦當然已被修復。

$$z_{at} \leq z_{at+1} \qquad \forall a \in A_p, t = 0, \ldots, T-1 \tag{3.8.14}$$

限制式(3.8.15)代表在時刻t時若節線a已與供給點連通，則該節線在之後亦會繼續保持其連通性。

$$y_{at} \leq y_{at+1} \qquad \forall a \in A_e \cup A_p, t = 0, \ldots, T-1 \tag{3.8.15}$$

限制式(3.8.16)代表在時刻t時若節點i已與供給點連通，則該節點在之後亦會繼續保持其連通性。

$$v_{it} \leq v_{it+1} \qquad \forall i \in N, t = 0, \ldots, T-1 \tag{3.8.16}$$

3.9 考量工作隊能力不同之模式修改方式

上述之數學模式皆假設所有工作隊的能力皆相同，因此不同工作隊在時程不衝突下可互換其執行任務。然而，現實情況有可能每組工作隊各有不同的能力或效率，因此不同工作隊不可任意互換任務，本節將針對此情境修改先前3.8.1小節的P_a，以及修改數學模式之限制式(3.8.9)、(3.8.11)、(3.8.12)如下。

參數修改

P_a^r　　工作隊r修復節線a所需的修復時間

限制式修改

將先前限制式(3.8.9)修改成限制式(3.9.1)，爲每條節線a至多只會被一個工作隊在某時刻被修復，需考慮每組工作隊針對每條節線的能力不一樣。

$$\sum_{a \in A_p} \sum_{u=t}^{min\{T, t+P_a^r-1\}} \alpha_{aur} \leq 1 \qquad \forall r \in 1, \ldots, R, t = 0, \ldots, T \tag{3.9.1}$$

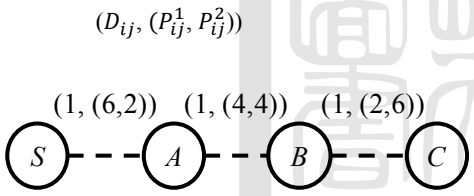
再將限制式(3.8.11)及(3.8.12)修改成限制式(3.9.2)和(3.9.3)

$$\sum_{t=0}^{\min_{r\in R} P_a^r - 1} z_{at} = 0 \qquad \forall a \in A_p \tag{3.9.2}$$

$$\sum_{r=1}^{R} \sum_{t=0}^{P_a^r - 1} \alpha_{atr} = 0 \qquad \forall a \in A_p \tag{3.9.3}$$

範例說明

假設現在有一網路如圖3.10左，而工作隊針對每一條節線的修復時間如圖3.10右。可能形成的兩種工作分配如圖3.11及3.12，可以發現其總等待修復時間可能會有很大的差別，其中圖3.11爲此圖的最佳排程，我們可以發現節線(B, C)其實在時刻2時就可以交給工作隊2進行修復，但因爲工作隊2在節線(B, C)修復需要花費的時間超過等待工作隊1修復完節線(A, B)再修復節線(B, C)的時間，因而可能造成延遲。



節線	修復時間	
	P_{ij}^1	P_{ij}^2
(S, A)	6	2
(A, B)	4	4
(B, C)	2	6

圖 3.10: 工作隊能力不同之修復網路

由此例可知在工作隊能力不同時，有時讓某些工作隊(通常是能力較差的工作隊)稍事休息而非一味工作(現實常用的人工排程原則)，說不定會有更好的排程結果。

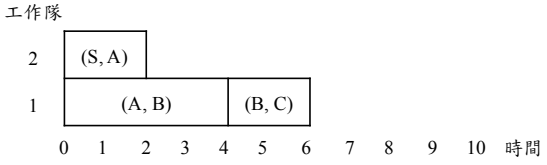


圖 3.11: 不同工作隊能力排法一

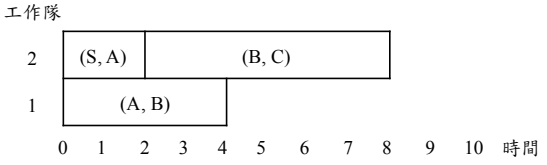


圖 3.12: 不同工作隊能力排法二

3.10 小結

本章節先討論問題的定義以及假設並說明其爲*NP-hard*後，接著提出一個網路縮減機制，將原問題進一步簡化成較小但僅包含毀損節線的等效網路。接著說明單一工作隊時，其最佳排程應遵守優先修復「邊界毀損節線」的「由外而內」之排程直覺，並以RCPSP數學模式爲基礎提出一個整數規劃模式 M^s，再參考INDS的建模方式提出多組工作隊之整數規劃模式 $M^m_{B\&C}$，該模式必須利用B&C技巧逐一加入新的限制式以排除子迴圈，而該方法常因必須持續加入的限制式數量過多而導致求解效率不佳。因此，我們又提出另一種整數規劃模式M^m_{MCF}，利用偵測各節點的單位需求是否被滿足以判斷其與供給點的連通性，應可比模式 $M^m_{B\&C}$的求解效率更好。最後，我們針對工作隊能力不同的情況，提出模式M^m_{MCF}的修正方式。下一章將會針對單一及多組工作隊的網路修復排程問題提出較整數規劃模式更有效率的數種求解演算法。

第四章　節線修復排程演算法之設計

在第三章提出的整數規劃模式在處理較大網路時，會耗時甚久，因此本章將提出數種求解演算法，期能使用較短時間得到品質不錯的排程，較符合現實需求。一開始在4.1節會先介紹將時間切段，再分別針對較小的時間區段求解整數規劃模式的分段式啓發演算法；再針對單一工作隊的特例提出窮舉法(4.2節)，以及兩種貪婪式演算法(4.3節)，再將其轉換成多組工作隊的排程；最後於4.4節設計數種基因演算法。

4.1　分段式啓發演算法 (Sequential Segment Heuristic，SSH)

分段式啓發演算法 (SSH) 爲Huang (2015)所提出的一種啓發式求解演算法，藉由將時間切成小區段，利用此較短的時間計算從供給點可以到達的網路範圍，以此爲基礎逐漸地由近而遠往外修復，直到所有節點皆被搶通爲止。圖4.1爲此演算法的基本想法示意圖，第一次求解的網路包含節點S、A、B、C、D及其相關節線，第二次求解的網路包含節點H、G、E、F及其相關節線，以此類推。而在切時段方面，求解的總時段T可以比預期規劃的時段還要長，藉此可以觀測到更大的網路，以得到更準確的結果。本研究亦承襲此一想法，將其修改爲修復節線以使其更適合於本研究，以下4.1.1節說明此分段式啓發演算法。

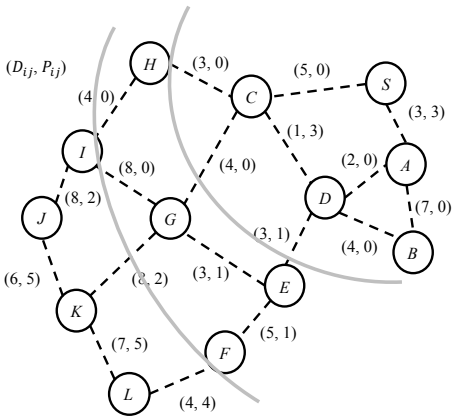


圖 4.1: SSH演算法想法示意圖

4.1.1 演算法步驟

假設有一網路$G=(N,A)$，共有R組工作隊進行搶修且工作隊能力一樣，若訂定n_{seg}爲期望的時間區段個數 (亦即SSH一共迭代n_{seg}次，每次求解該時間區段內的排程)，若將n_{seg}值變大，迭代次數變多也會使每次求解的網路變小，預期會使求解時間變快。T_{total}設定爲所有節線修復的時間加總，即$\sum_{a\in A_p} P_a$，則每一區段的時間T'可由$\lceil \frac{T_{total}}{n_{seg}*R} \rceil$計算得出，但此值需大於最長的節線修復時間，意即$T' \geq \max_{a\in A_p} P_a$，若發現$T' < \max_{a\in A_p} P_a$，則必須將$n_{seg}$的值調小。接著設定$\alpha \geq 1$，利用$\lceil T'*\alpha \rceil$設定每次迭代數學模式的$T$值，也就是該次迭代的總修復時間，藉由此調整可以將非原時間區段內的節線也考慮進來，而節線的修復排程只有考慮到原區段T'。舉例來說，若$\alpha = 1.5$，表示我們將$1.5T'$時間內可修復的節線列入考慮來求解整數規劃模式，然而其最佳解我們只取那些在T'時間內的排程決策。因此，若我們將α調整變大時，在每次求解中考慮到的時間區段變大會導致列入排程考量的網路也相對變大(亦即「看得比較遠」)，將可以使排程規劃得到更好的結果，但其求解過程將更耗時；反之，若使用較小的α，其求解過程較快，但也因「看得不夠遠」，所以其排程品質應較差。如何拿捏合適的α值，必須自更多測試經驗來嘗試而得。

以圖4.2爲例，可以想像原先只有考慮線段L1所切的範圍網路，但若α值調大，可能可以考慮線段L2內的範圍網路，亦即所考慮的網路更大，但最後的節線修復排程還是只有考慮線段L1內的部分。

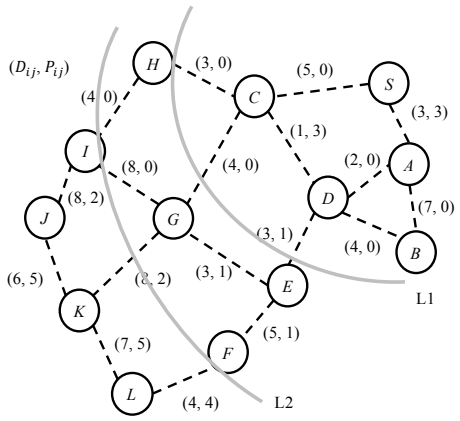


圖 4.2: SSH演算法α示意圖

接著設定β_r爲每個工作隊r的空閒釋放時刻，$A_{connect}$記錄每個節線被打通的時刻，並設定一個初始設爲$false$值的布林參數A_{repair}用以記錄節線是否被修復，$currentTime$爲該次迭代的初始全域時刻，β_a^A表示各節線在每一個區段最早打通時刻。演算法2即爲*SSH procedure*演算法流程。

Algorithm 2 SSH procedure

Require: $G, R, n_{seg}, \alpha, \beta_r, T_{total}, A_{connect}$

1: **function** SSHPROCEDURE($G, R, n_{seg}, \alpha, \beta_r, T_{total}, A_{connect}$)
2: **local variables:** $currentTime, T, T', G'$ ▷ G' is temporary graph
3: $\beta_r, currentTime \leftarrow 0$
4: $A_{connect} \leftarrow$ inf
5: $T' \leftarrow \lceil \frac{T_{total}}{n_{seg} * R} \rceil$
6: $T \leftarrow T' * \alpha$
7: **while** *some* $A_{connect} =$ inf **do**
8: $Dijkstra(G)$ ▷ (1) construct shortest path
9: $G' = setArc(G, T)$ ▷ (2) set graph to optimize
10: $optimizeModel(G', T, T', currentTime)$ ▷ (3) optimize the model
11: update β_r, $A_{connect}$, $currentTime$ ▷ (4) update argument's time
12: $combineArc(G)$ ▷ (5) combine the repaired arc
13: **end while**
14: **end function**

[步驟一] 計算從供給點S到達每一個節點的最早連通時刻。

此部分可以使用各種一對多的最短路徑演算法實作，將修復時間視同於距離，本研究則是使用Dijkstra演算法實作。

[步驟二] 選擇需要求解的節線集合。

初始一暫存的網路G'，查看所有$a \in A$，若其兩端點的最早可連通時刻皆小於或等於T，則將該節線a加入G'中，此G'即爲求解該次迭代的縮減網路。

[步驟三] 求解縮減網路的整數規劃模式。

此部分的求解模型使用3.8小節提出的修正後整數規劃模式M_{MCF}^m，而因爲每次迭

代的總修復時間內並不一定能修復所有節線，所以需修改限制式(3.8.11)爲限制式(4.1.1)。每次求解的數學模式中，每個工作隊的釋放時刻β_r皆不一樣，其表示工作隊需在時刻大於其釋放時刻時才能進行工作，所以在模型的求解上也需修改限制式(3.8.12)爲限制式(4.1.2)，定義各工作隊可以開始修復的時刻。

$$\sum_{t=0}^{min\{T,P_a-1\}} z_{at} = 0 \qquad \forall a \in A_p \tag{4.1.1}$$

$$\sum_{r=1}^{R} \sum_{t=0}^{\beta_r+P_a-1} \alpha_{atr} = 0 \qquad \forall a \in A_p \tag{4.1.2}$$

限制式(4.1.1)與(3.8.11)的差別在於因爲T不一定能修復該縮減網路的所有節線，故取其min值規範。限制式(4.1.2)與(3.8.12)的差別則是延長工作隊的釋放時刻β_r，限制每個工作隊需在釋放時刻之後才能進行工作。

爲了避免節線的最早打通時刻因爲時間切段而造成下一個時段提早連通，故需在前一期判斷節點i是否已連通，並更新由i往外連出的所有節線a，使β_a^A的時刻需晚於節點i的連通時刻，並在下一期加入限制式(4.1.3)。而爲了避免該期修復完卻仍未連通的情況，確認該期被修復的節線a並更新A_{repair}爲$true$，並在下一期加入限制式(4.1.4)。

$$\sum_{t=0}^{\beta_a^A-1} y_{at} = 0 \qquad \forall a \in A_p \tag{4.1.3}$$

$$z_{a0} >= 1 \qquad \forall a \in A_{repair}, A_{repair} = true \tag{4.1.4}$$

[步驟四] 更新β_r、$A_{connect}$、$currentTime$。

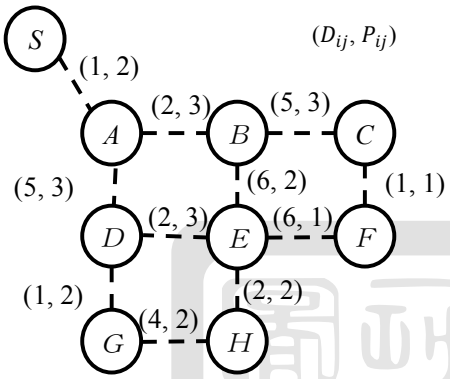
計算所有工作隊最後的釋放時刻β_r'，也就是每個工作隊最後一條節線的連通時刻，並更新β_r，更新方式爲 $\beta_r \leftarrow \beta_r' - \min_{r \in R} \beta_r'$；更新$A_{connect}$，將數學模式求得的節線連通時刻加上目前的$currentTime$；最後更新$currentTime$，更新方式爲$currentTime \leftarrow currentTime + \min_{r \in R} \beta_r'$。

[步驟五] 將已修復的節線濃縮爲供給點。

查看所有節線$a \in A$，若已連通則將a從A刪除；若一端點已連通、另一端點還未連通，則將未連通的點與供給點S以一條新節線互相連通，並將節線a從A刪除，若兩端點皆未連通則保留。

4.1.2 範例說明

假設有一網路如圖4.3左，且爲了將接下來的解說可以對應程式實作，將每條節線的對應編號顯示如圖4.3右。將初始參數n_{seg}設定爲2、工作隊數量R爲2，T_{total}爲所有需修復節線的時間加總可得24，經計算後得T'爲$\lceil \frac{24}{2*2} \rceil = 6$，並設定$\alpha$爲1.1，故可得$T$爲7。其中必須檢查是否$T' \geq \max_{a \in A_p} P_a$，因$6 \geq 3$，故此處合理。



節線	對應編號
(S, A)	1
(A, B)	2
(B, C)	3
(A, D)	4
(B, E)	5
(C, F)	6
(D, E)	7
(E, F)	8
(D, G)	9
(E, H)	10
(G, H)	11

圖 4.3: SSH範例說明網路圖

第一次迭代

[步驟一] 經過Dijkstra計算後可得從供給點到各個節點的最早連通時刻如圖4.4左。

[步驟二] 查看所有節線，若其兩端點的最早連通時刻皆小於等於T，則將節線加入一暫存網路G'中，此G'即爲該次迭代所要求解的網路，如圖4.4右所示，其中深色的節點爲該次求解G'的節點，各深色節點之間的連線即爲G'的節線。

[步驟三] 求解整數規劃模式，先設定時刻上限爲$T = 7$，但最後僅選取$T' = 6$以內的排程決策。而因爲是第一次進行求解，故β_r皆爲0，且不用檢查β_a^A。求解完的工作隊分配如圖4.5所示。

[步驟四] 計算各工作隊的最後釋放時刻β_r'，如圖4.5所示，兩工作隊釋放時刻皆

爲5。

節點	最早連通時刻
A	2
B	5
C	8
D	5
E	7
F	8
G	7
H	9

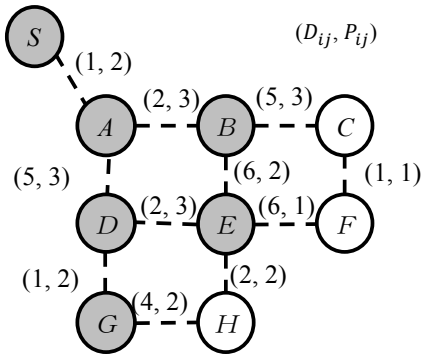


圖 4.4: SSH第一次迭代節點選擇

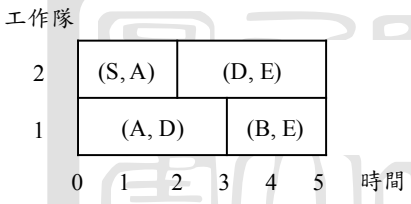


圖 4.5: SSH第一次迭代工作隊分配

並利用更新公式$\beta_r \leftarrow \beta'_r - \min_{r\in R} \beta'_r$可得$\beta_r$如下所示。

$\beta_1 \leftarrow 5-5=0$

$\beta_2 \leftarrow 5-5=0$

接著更新$A_{connect}$及$currentTime$，設定目前的$currentTime=0$，與求得各節線的打通時刻相加可得$A_{connect}$，接著更新$currentTime \leftarrow currentTime + \min_{r\in R} \beta'_r$。這裡陣列的數值爲圖4.3右節線對應之數值。

$A_{connect}[1] \leftarrow 2+0=2$

$A_{connect}[4] \leftarrow 3+0=3$

$A_{connect}[5] \leftarrow 5+0=5$

$A_{connect}[7] \leftarrow 5+0=5$

$currentTime \leftarrow 0+5=5$。

[步驟五] 依據訂定的規則將已修復的節線濃縮爲供給點，新的網路圖如圖4.6所示。

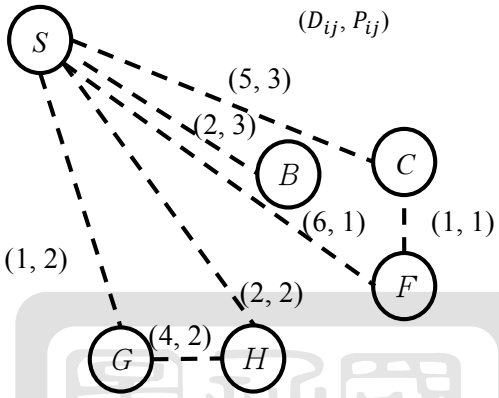


圖 4.6: SSH範例說明網路圖第二階段

第二次迭代

第二次迭代的**[步驟一]**與**[步驟二]**依循上述作法進行，可得如圖4.7所示。

節點	最早連通時刻
B	3
C	2
F	1
G	2
H	2

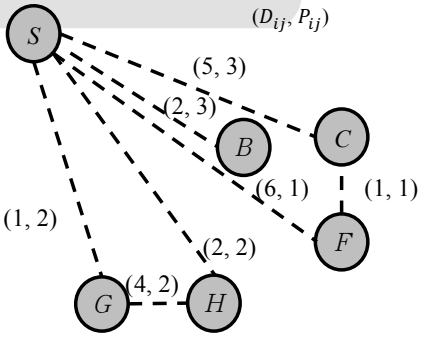


圖 4.7: SSH第二次迭代節點選擇

[步驟三] 求解整數規劃模式。需注意前一次β_r的值，此處皆爲0，並需檢查β_a^A，檢查方式爲查看各節點之連通時刻，以E點爲例，其連通時刻爲5，與其相連的節線β_a^A皆爲$5-currentTime=0$，並更新A_{repair}中的節線1、4、5、7爲$true$後，加入限制式(4.1.3)及限制式(4.1.4)。求解完的工作隊分配如圖4.8所示。

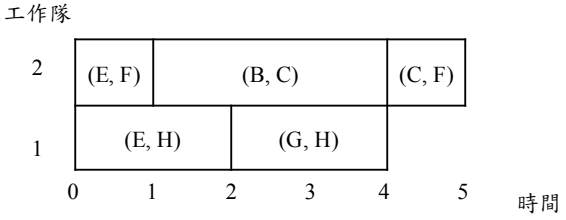


圖 4.8: SSH第二次迭代工作隊分配

[步驟四] 更新β_r如下所示。

$\beta_1 \leftarrow 4-4=0$

$\beta_2 \leftarrow 5-4=1$

更新$A_{connect}$及$currentTime$如下所示。

$A_{connect}[3] \leftarrow 4+5=9$

$A_{connect}[6] \leftarrow 5+5=10$

$A_{connect}[8] \leftarrow 1+5=6$

$A_{connect}[10] \leftarrow 2+5=7$

$A_{connect}[11] \leftarrow 4+5=9$

$currentTime \leftarrow 0+4=4$。

[步驟五] 更新網路圖。

依此步驟持續做完之後，可以得到最終的工作隊分配以及各節線的連通時刻如圖4.9及圖4.10所示。利用此節線運輸時刻可以得到最後的總修復等待時間爲234單位，其值與未分段的原整數規劃模式求得的226單位相比，約有3.54%的誤差；若α值設定爲1，亦即考慮的節線較少，則得到237單位的總體等待時間，約

爲4.87%的誤差。

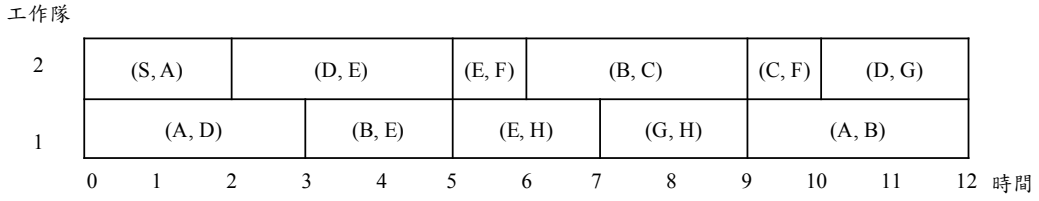


圖 4.9: SSH最終工作隊分配

節線	連通時刻	需求	節線	連通時刻	需求
(S, A)	2	1	(D, E)	5	2
(A, B)	12	2	(E, F)	6	6
(B, C)	9	5	(D, G)	12	1
(A, D)	3	5	(E, H)	7	2
(B, E)	5	6	(G, H)	9	4
(C, F)	10	1			

圖 4.10: SSH求出之各節線連通時刻

4.2 窮舉法 (Brute Force method，BF)

多工作隊的排程規劃可以藉由先求得單一工作隊特例的節線修復排程之後，再進行多組工作隊的排程組合以得到結果，本研究藉由窮舉法可獲得單一工作隊的修復排程，在4.2.1節說明，並在4.2.2節說明多組工作隊的排程組合方式。

4.2.1 方法說明

窮舉法 (Brute force method，BF)會將各個節線的修復順序組合皆試過並進行求解，如有三條節線A、B、C要修復，可以利用分支界限法將ABC、ACB、BAC、BCA、CAB、CBA此六種組合皆試過，由其中選出結果最好的答案以確保得到最佳解。初始定義如下：

現有一網路$G=(N,A)$，設定初始最小總需求等待時間$minDemandLost$爲一極大值，$demandLost$爲目前時間下所累積的總需求等待時間，$repairedNum$爲需要修復的節線總數，$depth$爲目前選取的節線數，$currentT$爲目前連通的時刻，並利用$bruteForce(A,0,0,0)$進行求解。方法如演算法3所示。

Algorithm 3 bruteForce

Require: $A, depth, demandLost, currentT$

1: **function** BRUTEFORCE($A, depth, demandLost, currentT$)
2: **if** $depth == repairedNum$ **then**
3: update the *minDemandLost* by *demandLost*
4: **return**
5: **end if**
6: **for** each arc a which is not repaired **do**
7: add a to the order of repaired arc and update $curentT$ to $newCurrentT$
8: $newLost$ = $demandLost$ + CHECKWAITINGTIME($A, a.time$)
9: mark a as repaired
10: BRUTEFORCE($A, depth+1, newLost, newCurrentT$)
11: mark a as unrepaired
12: **end for**
13: **end function**

$bruteForce(A,0,0,0)$爲遞迴本體，$checkWaitingTime(A,addTime)$則是計算需求等待時間的方法，用來計算修復一條節線時，仍未被滿足的需求當下已累積的等待時間總和。此方法可以保證找到最佳解，但是由複雜度分析可以得知，因爲需要遍歷

所有未修復節線的組合，所以複雜度爲$O(n!)$。舉例來說，若需要修復的節線數量爲12時，則需要測試12!種組合，也就是近5億種組合後才能找到最佳解，雖然可以利用加設條件的方式將一些明顯不需嘗試的組合在演算法的過程中取消掉(譬如節線還沒修完，所累積的總需求等待時間已超過記錄中的最小需求等待時間等)。但即使可利用一些方法來減少需被檢查的組合，仍需要許多運算量，而導致運算時間過久無法快速求解，僅能當成確認小型網路的答案是否爲最佳解使用。

4.2.2 多組工作隊之修復任務分配方式

在求得單工作隊的節線排程規劃後，皆可以將其排程結果再重新分配給多組工作隊。此分配方式立即得到不錯的可行解，譬如使用FCFS (First-come, First-Served)分配法則。將單工作隊所決定的最佳路徑規劃排程，直接派發給多工作隊執行，只要任一工作隊完成修復某節線即可馬上再承接下一份修復子任務，也就是說工作隊在子任務與子任務之間不會閒置，以表4.1爲例，此爲單一工作隊所得到的節線修復排程，若現在有三個工作隊利用FCFS的方式排入工作，其分配方式將如圖4.11所示：一開始工作隊1沒有任何工作故排入節線3，接著將節線5及2排給工作隊2及3，在時刻4時，因爲工作隊3已完成工作，故可繼續將節線1排給工作隊3，依此類推，一旦有工作隊完工即可馬上再承接新的子任務，可得最終修復時刻爲10，此方法不會讓工作隊有閒置時間，可得到不錯的結果且容易實作。

表 4.1: FCFS排程範例之個別修復工作資訊

執行順序	1	2	3	4	5	6	7
節線編號	3	5	2	7	1	4	6
修復時間	1	5	4	9	2	2	3

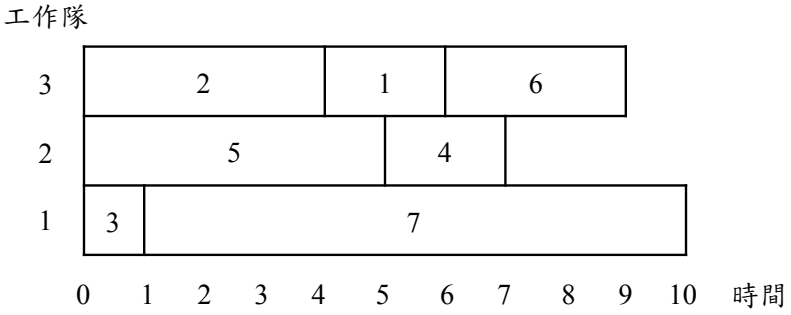


圖 4.11: 依單一工作隊排程結果以FCFS指派給多組工作隊之修復排程範例

4.3 貪婪式演算法

此部分將會介紹兩種貪婪式演算法，分別爲貪婪樹生成法以及貪婪連通分量生成法，此兩種方法主要用於單一工作隊的節線修復排程，此兩種方法的主要的不同點爲前者較適用在樹狀網路結構，而後者則可用非特定結構的一般網路。先求出單一工作隊排程後，再利用4.2.2節提出的FCFS方法，將其轉換成多工作隊節線修復排程。

4.3.1 貪婪樹生成法 (Greedy Tree method，GT)

Nurre et al. (2012) 本節提出的派工法則利用需求量以及修復時間的權重比值(即需求量/修復時間，此比值越大的節線越值得早點修復)的方式，來將節線加入序列裡面，利用當下時間閒置的工作隊去修復，以達到最大流、最小成本流或者最短路徑等問題的解，而取其權重比值的大小當作判斷基準，可以想成現實問題中，希望越短時間修復管線使其滿足越多人越好，本研究權重計算方式的想法基準由此而來。貪婪樹生成法 (Greedy tree method, GT) 最大的不同點是每次的修復不是決定修復完整路徑，而是一條一條節線慢慢決定，而因爲在網路縮減機制會造成loop(自環)，所以其權重的計算必須稍加修改。此方法雖然在執行速度上與 Nurre et al. (2012) 所提的派工法則效率相比相對較不佳，然而仍可以在極短時間內求得不錯解，且其解之

品質還更高。初始定義如下：

現有一樹狀網路$G = (N, A)$，設定$demandLost$爲目前時刻下所累積的總需求等待時間，利用$treeHeuristic(A, 0, 0)$求解，方法如演算法4所示。

Algorithm 4 treeHeuristic

Require: $A, demandLost$

```
1:  function TREEHEURISTIC(A, demandLost)
2:      while some arc unrepaired do
3:          for all n ∈ N do
4:              find the path P from n to a source node
5:              calculate the time and demand from P
6:              update the largest weight and finalP
7:          end for
8:          tmpArc ← first arc of finalP is adjacent to a source node
9:          demandLost += CHECKWAITINGTIME(A, tmpArc.time)
10:         mark tmpArc become repaired
11:     end while
12: end function
```

此演算法會先檢查每一個節點到已修復網路的路徑，並計算其中的需求總合和修復時間總和，並取其比值當作權重值，以此做爲判斷基準，越大表示這條路徑的重要性越大。找出最大的權重值後，並不是把整條路徑依序修復，而是只有修復從已修復網路往外連接的第一條節線，這邊只修復第一條節線的原因如圖4.12所示，現在假設所有節線皆需要修復，一開始各節點的權重值分別爲節點A的1 / 3 = 0.33，節點B的(1 + 2) / (3 + 1) = 0.75，節點C的(1 + 5) / (3 + 3) = 1，若是修完整路徑的話，修繕路徑將會依序修復節線(S, A)、節線(A, C)、節線(A, B)，得到的總需求延遲時間爲(1 + 2 + 5) * 3 + (2 + 5) * 3 + 2 * 1 = 47；若是使用貪婪樹生成法的邏輯只修復第一條節線，經過計算後，修繕路徑將會依序修復節線(S, A)、節線(A, B)、節線(A, C)，得到的總需求延遲時間則爲(1 + 2 + 5) * 3 + (2 + 5) * 1 + 5 * 3 = 46，可以發現46比47好。從圖中可以發現，因爲修繕整條路徑可能會導致修復完一條節線後，權重值經重新

計算後會有更好的選擇出現，造成路線的變動。

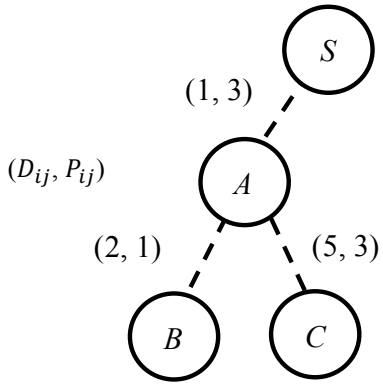


圖 4.12: 只修復第一條節線的例子

在3.4小節提到網路縮減的部分將會影響此演算法求解，如圖4.13所示，若已修復節線(S, A)，可以發現若網路沒有縮減，節點C的權重值最高，所以將會選擇節線(A, C)修復；但是觀察後可以發現若修復節線(A, B)其實更好，因修復完還可以順帶打通節線(B, F)與節線(B, G)，應把其需求也一併算入，但因爲沒有合併而忽略修復節線(A, B)後續帶來的效益，故網路縮減將會影響此演算法的結果。

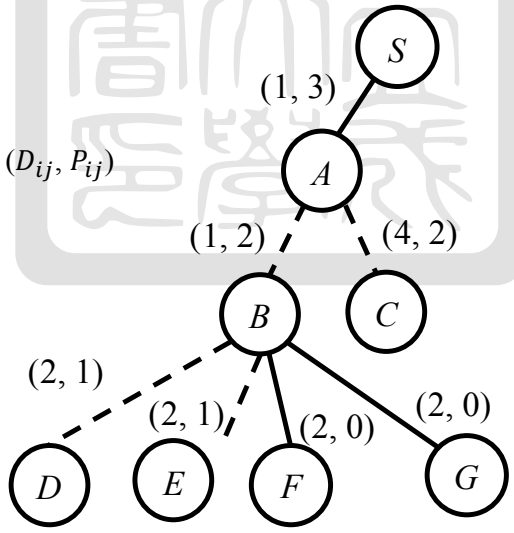


圖 4.13: 修復節線順序例子

貪婪樹生成法適合運用在樹狀結構的原因如下：每次決定修復節線時，需計算出每個節點連向已修復網路的路徑，若將已修復網路結合成一個點來看的話，其實就是從已修復網路連通到未打通節點的最大權重樹，而樹狀結構可以保證連通路徑只會有一種，故適用此方法。

而在一般網路結構的情況雖然也可以使用貪婪樹生成法，但求解結果會較差，以圖4.14爲例，此網路在修復時，因爲節線(B, C)較少的需求以及較長的修復時間，在每次的最大權重樹選擇時，皆不會被考慮在路徑上，而在演算法結束時，我們可以得到如圖4.15的網路圖，可以發現節線(B, C)皆未被修復，爲了解決此情況，可以在演算法最後，把還未修復的節線依據本身的需求除以修復時間之權重值大小來排序作修復，而因爲這些未修復的節線一定會跟已修復網路連通，所以只需個別計算本身的權重值，便可以解決一般網路結構的情形。

雖然Brucker (2007)也有提及$1|tree|\sum w_j C_j$的演算法，其計算權重的方式稍作修改後也可以用於求解本研究在樹狀結構的問題，並且證明可以在單工作隊情況的樹狀網路得到全域最佳解，但本研究所求解的問題並非全是樹狀網路結構，針對貪婪樹生成法也有提出適用於一般網路的修改方式，故不考慮使用該演算法，而藉由貪婪樹生成法得到單一工作隊的節線修復排程後，再進行多組工作隊的排程配置可得到多組工作隊之排程結果。

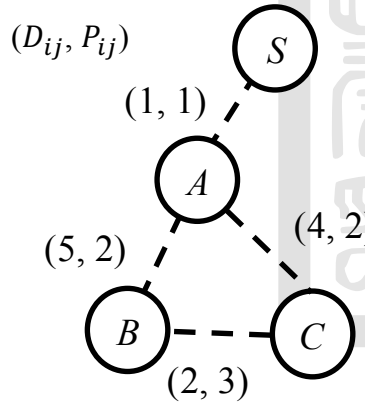


圖 4.14: 一般結構貪婪樹生成法修復前

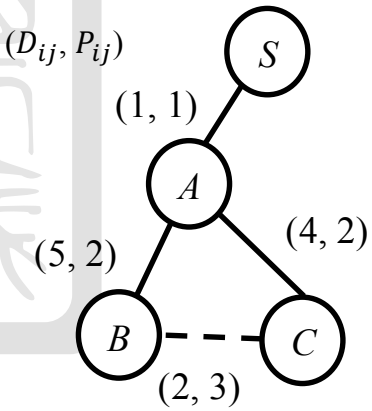


圖 4.15: 一般結構貪婪樹生成法修復後

4.3.2 貪婪連通分量生成法 (Greedy Connected Component Method，GCC)

貪婪連通分量生成法 (Greedy Connected Component Method，GCC) 找尋與外界連的節線，並計算其連通子圖的需求量以及修復時間的權重比值，利用此一方式選

擇欲修復的節線。初始定義如下：現有一網路，設定$demandLost$爲目前時間下所累積的總需求等待時間，利用$componentHeur(A,0)$求解，方法如演算法5所示。

Algorithm 5 Component Heuristic

Require: $G, demandLost$

```
1: function COMPONENTHEUR(G, demandLost)
2:     while some a ∈ A is unrepaired do
3:         LINKA ← set of arcs connected to a source node
4:         for tmpArc ∈ LINKA do
5:             find the connected component C connected to tmpArc, and C ∪ LINKA = tmpArc
6:             calculate the time and demand from C
7:             update the largest weight and finalArc
8:         end for
9:         demandLost ← demandLost + checkWaitingTime(A, tmpArc.time)
10:        mark tmpArc as repaired
11:    end while
12: end function
```

貪婪連通分量生成法會先將已與供給點連通之網路往外連接的「邊界毀損節線」集合設爲$LINKA$，以圖4.16爲例，如果節線(S, A)、(A, B)、(B, C)、(C, D)、(A, D)爲目前已修復的節線，其構成的子網路稱其爲G'，與G'連接的節線(A, E)、(D, F)、(C, G)、(B, H)所構成的集合即爲$LINKA$。接著會分別計算$LINKA$裡的節線其涵蓋的連通分量且不包含其他$LINKA$節線的權重值，以節線(A, E)爲例，會計算出節線(A, E)、(E, F)的需求加總以及修復時間加總再取其比值，得到(2 + 3) / (1 + 7) = 0.625；以節線(B, H)爲例，會計算出節線(B, H)、(H, G)的需求加總以及修復時間加總再取其比值，得到(1 + 3) / (4 + 9) ≅ 0.38；$LINKA$裡的節線皆計算過後，選擇比值最大的節線來修復，依此步驟直到所有節線皆被修復。

藉由貪婪連通分量生成法得到單一工作隊的節線修復排程後，再進行多組工作隊的排程配置即可得到多組工作隊之排程結果。

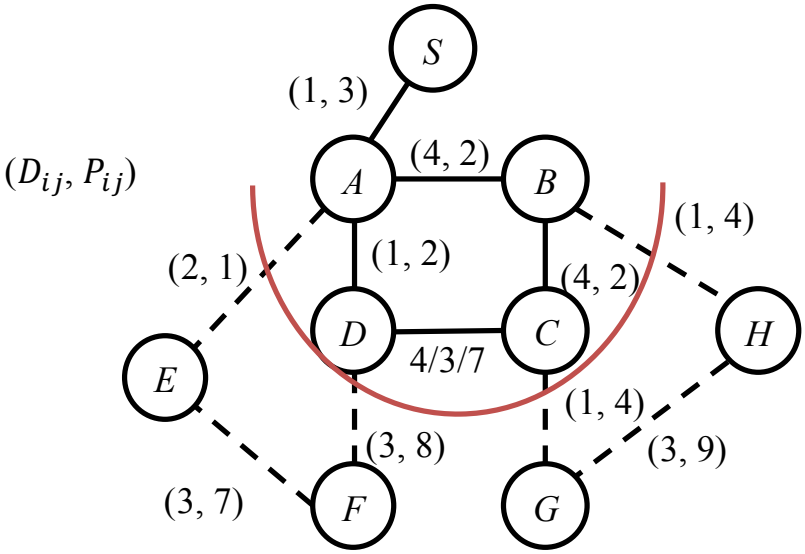


圖 4.16: *LINKA*示意圖

4.4 基因演算法 (Genetic Algorithm，GA)

在演算法設計上，除了貪婪式演算法之外，排程相關問題經常會使用GA進行求解，本研究爲了比較其與貪婪式演算法的差別，故以Vallada and Ruiz (2011)爲原型，建立GA_1，並將其交配策略做修改設計出GA_2以及GA_3使其更符合本研究的問題，以下4.4.1節將介紹基於Vallada and Ruiz (2011)所提出的原型所設計的方法，4.4.2節則提出本研究針對交配策略所進行的修改。

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

Vallada and Ruiz (2011)針對多平行機台的排程問題提出基因演算法來求解，且經實驗證明其方法可以得到不錯的結果。本研究在假設工作隊能力一樣時，可以視同多平行機台的排程問題加入網路結構，故本研究根據其所提出的編碼方式以及策略進行修改，設計出GA_1，以下介紹方法。

染色體的編碼方式

每一條染色體皆可以表示成一組解，我們可以想成每個工作隊皆有其需負責進行修復的節線編號，以一陣列儲存之。若現在共有三個工作隊且九條節線需進行修復，可將一條染色體以圖4.17表示之，以此例而言，工作隊1會依序修復節線1、6、7、2。以此類推，藉由此方式可以方便表示一組解，不需再利用傳統的0、1組合來進行編碼。

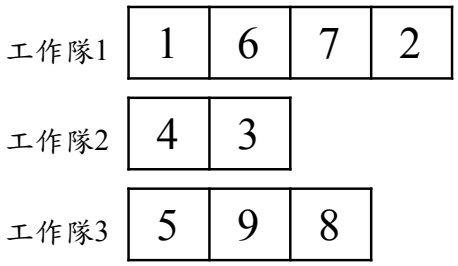


圖 4.17: 染色體示意圖

適應函式

透過適應函式我們可以求出該條染色體的目標值，因本研究的問題爲最小化總等待時間，故其值越小表示越接近最佳解，藉由小節的編碼方式，在工作隊依序修復完節線後，可以利用4.2節所提到的*checkWaitingTime*函式計算該多組工作隊修復組合的需求等待時間加總。藉由接下來提到的交配以及突變策略，使求出的目標值越小。

挑選策略 (selection operator)

在每次的迭代的一開始都會進行染色體的挑選，本研究的挑選策略則是使用輪盤法的方式，亦即如果該解越好，其能夠被選進這次迭代的候選染色體的機率就越高，以下介紹其方法。

假設目前有m個染色體，其適應值分別爲$E(C_1), E(C_2), \ldots, E(C_m)$。

1. 將所有適應值減掉當中最小者後再加一並取其倒數：$E(C_i) = \frac{1}{E(C_i) - \min_{j=1,\ldots,m} E(C_j)+1}$ $\forall i = 1,\ldots,m$。

2. 計算所有新適應值的加總：$E_{total} = \sum_{i=1}^{m} E(C_i)$。

3. 計算各個染色體被選到的機率：$P_i = E(C_i)/E_{total}$。

4. 利用累積機率的方式，每個染色體的累積機率爲Q_i。

5. 最後產生隨機亂數x，若x介於$Q_i \leq x \leq Q_{i+1}$，則選擇染色體i。

在步驟1的計算方式之所以要取倒數是因爲本研究的目標問題爲最小化問題，爲了讓越小值的解機率越大故以倒數表示；而減掉最小值的原因，是因本研究的適應值皆非常相近，所以如果直接取倒數會導致每個解的機率差不多，等同於隨機挑選染色體，故利用此平移方式將較佳解的機率值提高。

從步驟中可以看出，在經過挑選策略後，有較佳適應值的解會有較高機率被挑選，而較差的解也不會因此直接排除，因其在後續的迭代中也有可能會找到較佳解。

交配策略 (crossover operator)

在交配策略部分，因爲每條節線只會被一組工作隊修復，所以利用以下所表示的方法確保交配出來的爲可行解。一開始先隨機挑選兩條染色體當作父母，如圖4.18所示，並針對第一條染色體的每個工作隊挑選一切割點，圖中工作隊1即以第二和第三個工作爲切割點，工作隊2則是以第一和第二工作爲切割點。

接著針對每個工作隊，將分割點的兩端分別加入新產生的染色體中，如4.19所示，並複製兩個第二條染色體，並使其對應新的兩個染色體，將重複的予以刪除。

最後再將複製出的第二條染色體餘下的工作，依序插入新染色體對應的工作隊中，如圖4.20所示。以上便可以藉由交換得到新的可行解，而在文獻中還利用區域搜尋 (local search) 的方式，在插入餘下工作的部分進行插入位置的選擇，但因爲本研究在目標值的計算較爲複雜，所以不考慮加入區域搜尋。

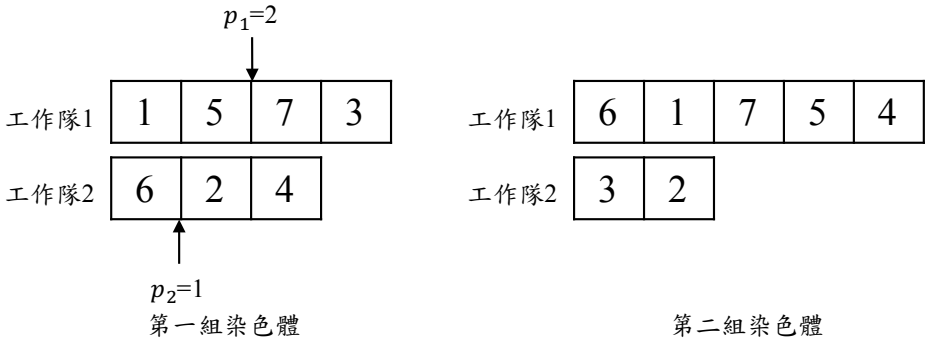


圖 4.18: 選擇父母及交配的分割點

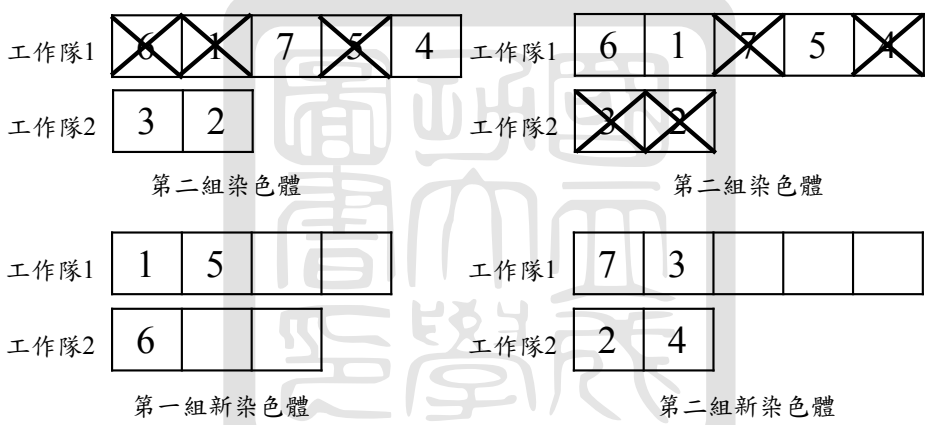


圖 4.19: 進行複製與交換

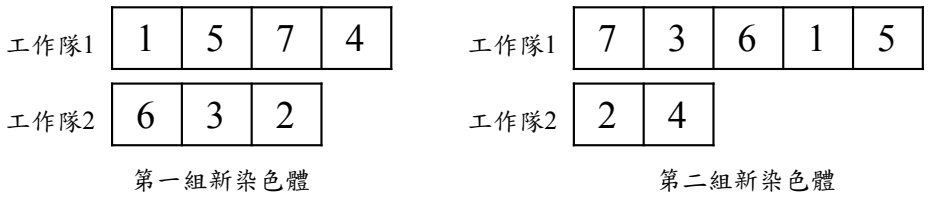


圖 4.20: 最後小孩結果

突變策略 (mutation operator)

突變策略則是爲了避免基因演算法落入區域最佳解，所以使產生出來的解有機率性的改變，改變的方式則是有機率的挑選現在餘下的染色體，再隨機挑選任意一組工作隊並交換其兩個工作，如圖4.21即是交換工作隊2的第一和第三個工作(即6,2互換位置)。

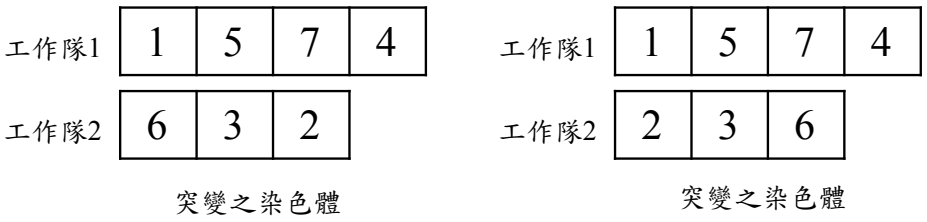


圖 4.21: 突變策略

綜合以上可將基因演算法表示如演算法6，其中M爲每個回合初始的染色體數量，P_c爲迭代時進行交配的機率，P_m則爲突變的機率。

Algorithm 6 Genetic Algorithm

1: **function** GA(M, P_c, P_m)
2: Generate initial population (M random chromosome)
3: **while** stopping criteria not satisfied **do**
4: evaluate each chromosome in population
5: select M chromosome and copy them to new population
6: probability P_c to do crossover and copy to population
7: probabilty P_m for each chromosome to mutate
8: **end while**
9: **end function**

4.4.2 修改交配策略

爲了使基因演算法的設計可以更貼近本研究的問題，我們將其交配策略進行修改提出GA_2。在進行複製與交換這個部分，我們不直接排入，而是先計算每一條欲排

入的節線的權重值，計算方式爲先計算從供給點到各個節點的一對多最短路徑，每條節線的長度則是利用修復時間除以需求的比值，我們希望此比值越外越好。藉由此計算過後，我們取每條節線的兩端點較小的值再加上節線自身的修復時間除以需求爲其權重值，越小的值表示其較接近供給點且需求較多可以先修，故我們利用類似選擇策略的輪盤法，在每個工作隊排入工作時，隨機挑選要先修的節線。

以圖4.19爲例，在第一組新染色體中的工作隊1，原先會先固定工作1、5後再排入工作7和4，但在GA_2中，因爲會先計算每一條節線的權重值，所以我們將工作1、4、5及7進行輪盤法選擇，以此類推最後可能得到的結果如圖4.22所示，其表示可能工作1、2及4有較高的權重值，故有較高的機率先被選中。

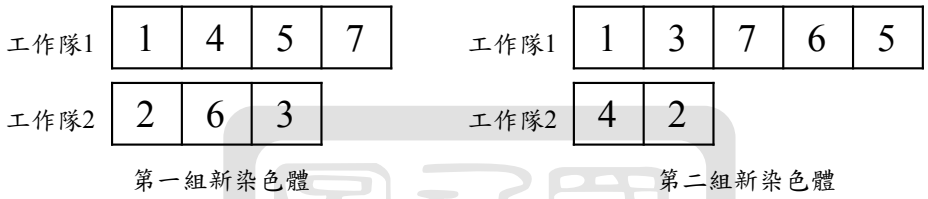


圖 4.22: GA_2可能結果

而GA_3的設計則是考量可能因爲決定每組工作隊的第一個工作後，其權重計算會隨著網路的節線修復而有所改變(譬如修復完一條毀損節線後，會改變部分毀損節線至供給點最短路徑的長度)，故在每個工作隊的第一個工作排定後，將已修復節線的修復時間改爲0後再重新計算權重值，再將後續的工作排入。以圖4.23爲例，經過第一輪輪盤法決定每個工作隊的第一個工作後，再重新計算一次權重值，再將剩餘工作依據新的權重值計算排入。

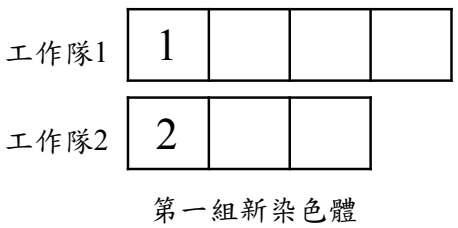


圖 4.23: GA_3過程

4.4.3 基因演算法範例說明

假設欲修復的節線共有6條節線，分別爲[1, 2, …, 6]，共有兩組工作隊進行維修，且每次迭代所保留的染色體數M爲4條，交配機率P_c及突變機率P_m分別爲0.6及0.2。

[步驟一] 迭代開始前，隨機產生初始的染色體，可以使用平均分配工作的方式給各個工作隊。

$$C_1 = [1, 6, 2], [4, 3, 5]$$
$$C_2 = [5, 4, 2], [3, 6, 1]$$
$$C_3 = [2, 1, 4], [3, 5, 6]$$
$$C_4 = [3, 4, 6], [5, 1, 2]$$

[步驟二] 進行選擇策略。

一開始先計算各個染色體的適應值。

$$E(C_1) = 134$$
$$E(C_2) = 130$$
$$E(C_3) = 135$$
$$E(C_4) = 140$$

接著將原適應值轉換成機率，因本研究的問題爲最小化，所以將數字取倒數後，可以使較低適應值的有較高的機率被選中，而爲了避免因適應值取倒數後皆很接近，故先減掉最小值再加一此平移計算後，再取其倒數成新適應值如下所示。

$$E(C_1) = \frac{1}{(134-130)+1} = 0.2000$$
$$E(C_2) = \frac{1}{(130-130)+1} = 1.0000$$
$$E(C_3) = \frac{1}{(135-130)+1} = 0.1667$$
$$E(C_4) = \frac{1}{(140-130)+1} = 0.0909$$

將其加總後可以得到0.2+1+0.1667+0.0909=1.4576。

計算各機率如下。

$$P_1 = \frac{0.2}{1.4576}=0.1372$$
$$P_2 = \frac{1}{1.4576}=0.6861$$
$$P_3 = \frac{0.1667}{1.4576}=0.1144$$
$$P_4 = \frac{0.0909}{1.4576}=0.0623$$

計算累積機率如下。

$$Q_1 = 0.1372$$
$$Q_2 = 0.1372+0.6861=0.8233$$
$$Q_3 = 0.8233+0.1144=0.9377$$
$$Q_4 = 0.9377+0.0623=1$$

因爲M爲4，所以隨機產生四個亂數介於0到1之間，假設產生出來的數字分別爲0.3214, 0.6413, 0.8432, 0.0456，以0.3214爲例，因$Q_1 \leq 0.3214 \leq Q_2$，所以將C_2加入新的迭代中，以此類推，最後新產生的群體爲[C_2,C_2,C_3, C_1]。

[步驟三] 進行交配策略。此部分將會進行四次，每次會先產生一個介於0到1之間的亂數，若其數字小於等於P_c，便會隨機挑選兩染色體。假設被挑選的染色體分別爲C_2及C_3。

C_2 = [5, 4, 2], [3, 6, 1]　C_3 = [2, 1, 4], [3, 5, 6]

隨機挑選$C2$各工作隊的切割點，假設爲1及2。此時新產生出來的C_5及C_6分別如下。

C_5 = [5], [3, 6]　C_6 = [4, 2], [1]

接著複製兩個C_3並使其對應C_5及C_6，將重複的刪除可得到如下。

C_3' = [2, 1, 4], []　C_3'' = [], [3, 5, 6]

最後結合C_3'、C_5及C_3''、C_6便可以得到最終結果。

C_5 = [5, 2, 1, 4], [3, 6]　C_6 = [4, 2], [1, 3, 5, 6]

藉此交換得到新染色體後，在此階段全結束時，將新染色體加入染色體群中。

[步驟四] 進行突變策略。每個解在這個階段都有突變的可能，產生一個亂數介於0到1之間，若其小於等於P_m便會產生突變，突變方式爲隨機挑選任一工作隊的任兩不同工作進行交換。假設現在C_5發生突變，隨機選到工作隊2的第一個工作和第二個工作交換，便會得到結果如下。

C_5 = [5, 2, 1, 4], [3, 6] ⇒ [5, 2, 1, 4], [6, 3]

以上便是整個基因演算法的流程。

4.5　小結

本研究利用數學規劃求解軟體Gurobi求解整數規劃模式，通常難以在短時間內處理較大規模的網路修復問題。本章節提出兩種方式，第一種爲以修復時間切段的分段式啓發演算法，先求解較小規模的問題，再以此爲基礎由近而遠往外修復，直到所有節線皆被修復；第二種則是針對單一工作隊修復所有毀損管線的特例，先以窮舉法、貪婪樹生成法、貪婪連通分量生成法等三種求解演算法進行求解，其中窮舉法可找出任一管線修復排程之最佳解，可供其它演算法比較其求解品質，但可能耗時甚久；而貪婪樹生成法則以貪婪法則來產生修復排程，該演算法可以在大量的樹狀結構例子中得到最佳解，但較不適用於非樹狀結構的情況；貪婪連通分量生成法一樣是利用貪婪法則來產生修復排程，但是不受限在樹狀結構，其方法是利用連通子圖的方式，也可以得到不錯的可行解。最後，利用單一工作隊的節線修復排程進行多工作隊轉換的方式得到結果。而除了貪婪演算法，我們也針對本研究的問題設計三種基因演算法，第一種原型爲單純地進行交配突變，第二種和第三種則是根據本研究網路的性質，將離供給點較近的節線給予較高權重，使其在交配策略時，有更高的機率先被排進工作隊中。

第五章　模式與演算法之數值測試及分析

此章節將會針對本研究所建立的整數規劃模式、有效不等式及演算法，分別測試樹狀網路結構 (tree graph) 以及一般網路結構 (general graph) 並進行驗證與分析，最後會測試三個與現實管線情境中大小較接近的例子。測試環境爲UBUNTU 13.10，搭配Intel® Core™ i7-4770，3.40GHz*8處理器與16G記憶體，並利用最佳化求解軟體Gurobi，版本爲6.5.1進行求解，求解時間設定爲最多10小時，也就是36000秒。資料已經經過網路縮減，以下說明測試網路資料的設定方式，並分別針對不同狀況分析數據。

5.1　測試網路資料

在現實的資料中，因爲管線網路通常沒有特殊結構，所以在網路圖的測試使用隨機產生的方式爲主，但在管線末端分送到各家庭裡則會有樹狀的分支結構，所以分別測試樹狀以及一般任意結構等兩種形式。本研究使用可以基於python開發的NetworkX套件(https://networkx.github.io)來產生隨機網路圖，此套件提供gnm_random_graph可以根據給定的節點與節線數來隨機產生網路圖，但必須自行檢查整個網路圖的連通性，亦即還必須確保是否該網路所有節點皆可互相連通(即只有一個連通分量(Component))；樹狀網路結構方面則是使用nrandom_powerlaw_tree藉由給定節點數來隨機產生。函式所使用的演算法皆可以在網際網路上找到。

以下針對樹狀網路與一般網路結構分別測試小型與中型的網路規模，且每個規模皆有5個例子，詳細資訊如表5.1。在修復時間以及需求的設定上，皆考慮在3到5單位之間。

本研究提出之方法，爲了因應不同網路結構及工作隊數量有分別適用的情境，故整理於表5.2，其中$M^{m}_{B\&C}$及BF因爲求解時間過長，在接下來的驗證與分析就不予考慮。

表 5.1: 網路結構資訊

網路名稱	節點數	節線數	節線修復時間	需求大小
$Tree_s$	[16, 20]	[15, 19]	[3, 5]	[3, 5]
$Tree_m$	[22, 30]	[21, 29]	[3, 5]	[3, 5]
$General_s$	[16, 20]	[31, 41]	[3, 5]	[3, 5]
$General_m$	[25, 29]	[35, 47]	[3, 5]	[3, 5]

表 5.2: 各情境適用之方法

	M^s	$M^m_{B\&C}$	M^m_{MCF}	*SSH*	*BF*	*GT*	*GCC*
樹狀網路	✓	✓	✓	✓	✓	✓	✓
一般網路	✓	✓	✓	✓	✓		✓
單工作隊	✓	✓	✓	✓	✓	✓	✓
多工作隊		✓	✓	✓	✓(FCFS)	✓(FCFS)	✓(FCFS)

5.2 單工作隊網路測試

在單工作隊的情境下比較前述提出的整數規劃模式，並利用較有效率的整數規劃模式與演算法比較其求解時間以及可行解的品質。

5.2.1 整數規劃模式比較

分別測試M^s、M^m_{MCF}在單一工作隊的表現，其中M^s只能求解單一工作隊的問題，表5.3爲結果。其中表格中時間代表求解時間，爲建置時間與運算時間的加總；Gap的計算則是若該組合解爲A、數學模式求得之最佳解爲B，利用$(A-B)/B$而得。結果可以發現M^s相對於M^m_{MCF}在求解時間上有較佳的表現，尤其當網路規模變大時，其差距會明顯拉大，此也說明M^s在單一工作隊的問題適合使用M^s求解。而一般網路因較樹狀網路有更多節線，因此求解將更爲耗時，因此M^s在求解一般網路時將較M^m_{MCF}更有效率。

表 5.3: 單一工作隊數學模式測試

網路結構	工作隊個數	測資	M^s		M^m_{MCF}	
			時間 (s)	Gap (%)	時間 (s)	Gap (%)
$Tree_s$	1	5	7	0.00	92.4	0.00
$Tree_m$	1	5	78.8	0.00	1055.2	0.00
$General_s$	1	5	331.8	0.00	357	0.00
$General_m$	1	5	3794.8	0.00	12050.4	0.00

5.2.2 演算法測試

利用M^s、*GT*與*GCC*比較，*GT*主要使用在樹狀結構，結果如表5.4所示。從表中可以發現，求解樹狀結構問題時，*GT*的確可以得到不錯的近似最佳解，其求解品質也優於*GCC*；而*GCC*的優點在於其可求解任何網路結構，兩演算法的的求解時間皆遠小於M^s。

表 5.4: 單一工作隊演算法測試

網路結構	工作隊個數	測資	M^s		*GT*		*GCC*	
			時間 (s)	Gap (%)	時間 (s)	Gap (%)	時間 (s)	Gap (%)
$Tree_s$	1	5	7	0.00	0	0.02	0	0.06
$Tree_m$	1	5	78.8	0.00	0	0.01	0	0.04
$General_s$	1	5	331.8	0.00	-	-	0.2	0.64
$General_m$	1	5	3794.8	0.00	-	-	0.8	1.23

圖5.1及圖5.2顯示在單工作隊時，樹狀結構的求解時間與Gap差異，圖5.3及圖5.4則是一般網路結構的求解時間與Gap差異。從圖中可以更清楚的看到求解時間在M^s中會有較明顯的變化，而對於演算法的差距則非常微小；在Gap方面差距則較小，但*GCC*在一般網路結構的表現，則相對樹狀網路結構來說表現較差，因爲樹狀結構在演算法中需考慮的部分較爲單純，分支與分支之間不會連接，在權重值的計算相對準確很多。

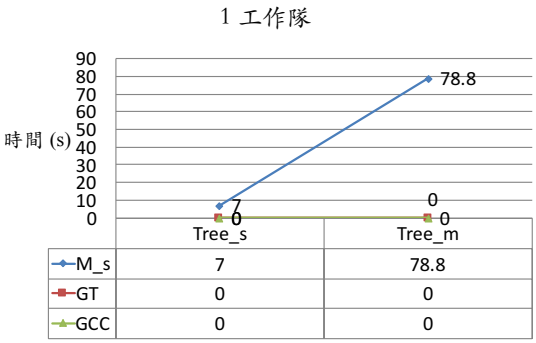


圖 5.1: 單一工作隊樹狀結構問題之求解時間比較

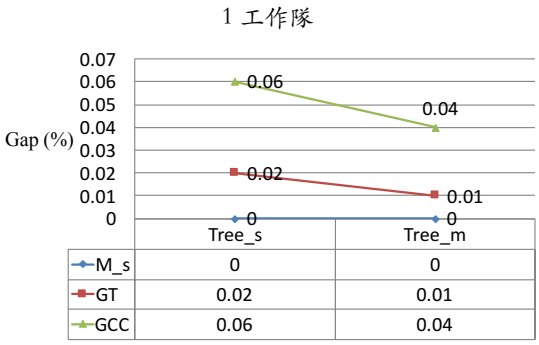


圖 5.2: 單一工作隊樹狀結構問題之Optimality Gap比較

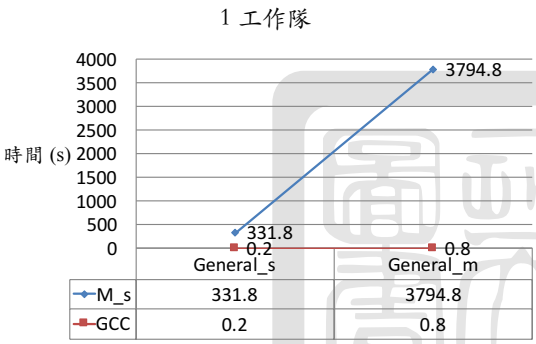


圖 5.3: 單一工作隊一般結構問題之求解時間比較

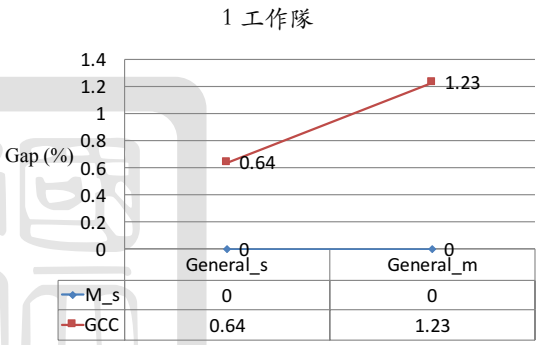


圖 5.4: 單一工作隊一般結構問題之Optimality Gap比較

5.3 多組工作隊網路測試

在多組工作隊的情境下討論M^m_{MCF}有效不等式加入、工作隊數量不同的測試，並與演算法比較。

5.3.1 有效不等式效益測試

在3.8.3節提出M_{MCF}^{m}可新增的四個有效不等式，分別測試16種組合，情境皆設定爲2組工作隊並在$General_s$的第一筆測資進行測試，可得到表5.5。表中列出在未加入有效不等式的求解時間，而剩餘四項有效不等式其求解時間分別爲16種組合中，有出現該有效不等式的資料平均，藉此可以看出該有效不等式的加入對求解時間的影響。爲了測試其效率，此測試只進行兩小時，若未求得最佳解則該次時間不予考慮，最佳解次數最大值爲8。從表中可以發現加入限制式(3.8.13)對整體影響較佳，該限制式表示任何點*i*的向內臨邊只要有一條爲連通外界，則該點*i*則必定亦連通外界，此限制式的確可有效地限縮將求解區域以改善求解效率；而限制式(3.8.14)則表現較不佳，有一個組合無法得到最佳解。

表 5.5: 有效不等式測試

有效不等式	$General_s$_1	
	求解時間(s)	**最佳解次數**
未加入	4090	-
限制式(3.8.13)	336	8
限制式(3.8.14)	2738	7
限制式(3.8.15)	2446	8
限制式(3.8.16)	2730	8

從結果中可以得知限制式(3.8.14)的結果是最差的，不僅平均花費時間較久，還有一種組合無法在兩小時內求得最佳解。因爲在測試過程中我們也發現每個有效不等式彼此之間可能會有綜效(亦即若同時加入，可能效率提升更多)，所以以下測試限制式固定加入限制式(3.8.13)，再分別測試其與限制式(3.8.15)、限制式(3.8.16)的組合表現，再與將所有有效不等式一併加入的情況一併測試比較。

表 5.6: M^m_{MCF}有效不等式組合測試

網路結構	工作隊個數	測資	有效不等式全加入	(3.8.13) + (3.8.15)	(3.8.13) + (3.8.16)
			時間 (s)	**時間 (s)**	**時間 (s)**
$Tree_s$	1	5	92.4	94.2	124.4
	2	5	70.4	59.6	49.8
	4	5	24.6	23.4	22.2
$Tree_m$	1	5	1055.2	610.4	1144
	2	5	594	765.2	405.4
	4	5	119.4	205.8	346.2
$General_s$	1	5	357	262	208.2
	2	5	417.8	194	179
	4	5	252.8	150	126.8
$General_m$	1	5	12050.4	7874.6	7959.4
	2	5	5597.4	5041.6	3568.2
	4	5	2985.2	3456.2	1398.2

由表5.6中可以看出，將四條有效不等式皆加入不一定都能夠得到較好的結果。在小型樹狀網路的情況，平均來看全部有效不等式皆加入可以得到較好的表現，但因爲求解時間都非常快速，所以較看不出表現差異；中型的樹狀網路則是沒有太大的差別。普通網路則有相對大的差別，限制式成長速度會因爲節線以及節點數的增加而上升，所以在限制式(3.8.13)與限制式(3.8.15)及限制式(3.8.16)的組合，因爲限制式較少又能有效刪減求解區域，某些例子甚至只需原先花費時間的二分之一的求解時間即可有與原先設定類似的表現。

5.3.2 工作隊數量不同測試

在多組工作隊的測試中，分別測試1、2、4組工作隊進行修復時，最佳解以及求解時間的差異，其中因爲$M^m_{B\&C}$幾乎無法在有效時間內求解完畢，所以不列出。參數

設定部分，M^m_{MCF}的有效不等式爲全部加入；*SSH*演算法的n_{seg}、$alpha$參數分別爲3以及1.3，可得到實驗結果如表5.7所示。

從結果中可以發現在整數規劃模式方面，當網路規模稍微變大時，求解時間將會有大幅度的成長，但是演算法的求解時間則不受網路規模變動有太大的影響，以目前的網路規模皆可在時限內求得最佳解；演算法方面，*SSH*和*GCC*相比，前者普遍上可以得出較佳的求解品質，但卻花費較久時間，而後者則相反。兩演算法皆可在求解時間以及品質上取捨，不過兩演算法求出的可行解Gap值約小於等於1%，其值近似於最佳解。

表 5.7: 多組工作隊測試

網路結構	工作隊個數	測資	M^m_{MCF}		*SSH*		*GCC*	
			時間 (s)	Gap (%)	時間 (s)	Gap (%)	時間 (s)	Gap (%)
$Tree_s$	1	5	92.4	0.00	7.4	0.00	0	0.06
	2	5	70.4	0.00	0.4	0.00	0	0.12
	4	5	24.6	0.00	0	0.77	0	0.88
$Tree_m$	1	5	1055.2	0.00	89.2	0.00	0.2	0.05
	2	5	594	0.00	4.6	0.05	0	0.12
	4	5	119.4	0.00	0.2	0.73	0	0.56
$General_s$	1	5	357	0.00	50.8	0.00	0.2	0.62
	2	5	417.8	0.00	21	0.04	0.2	0.57
	4	5	252.8	0.00	6.4	0.16	0	0.63
$General_m$	1	5	12050.4	0.00	602.6	0.00	0.6	1.23
	2	5	5597.4	0.00	154	0.02	0.2	1.18
	4	5	2985.2	0.00	9.8	0.77	0.6	1.22

圖5.5及圖5.6顯示在工作隊數爲2時，樹狀結構問題之求解時間與Optimality Gap的差異，在時間差異上相對於M^m_{MCF}而言，兩演算法皆遠小於M^m_{MCF}的求解時間；求解品質方面，*GCC*則稍差，但都算蠻接近最佳解了。

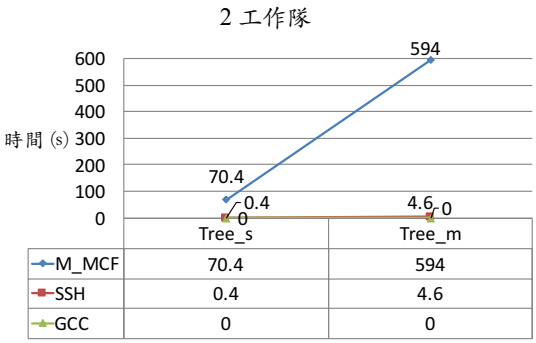


圖 5.5: 多組工作隊樹狀結構問題之求解時間比較

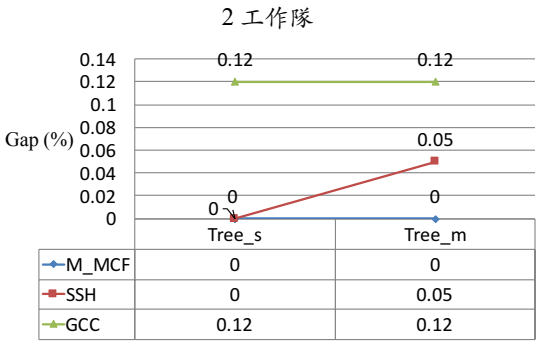


圖 5.6: 多組工作隊樹狀結構問題之Optimality Gap比較

圖5.7及圖5.8顯示在工作隊數爲2時，一般結構問題之求解時間與Optimality Gap的差異，整體趨勢與求解樹狀結構問題時大致上雷同，但卻有較長的求解時間。這可能肇因於一般結構問題的節線與節點數較多且有較複雜的連結方式。演算法在求解時間以及Optimality Gap上仍皆有不錯的表現。

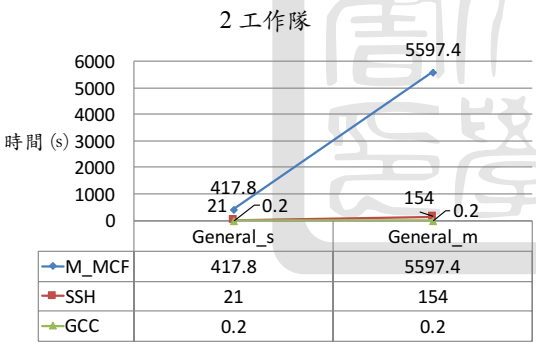


圖 5.7: 多組工作隊一般結構問題之求解時間比較

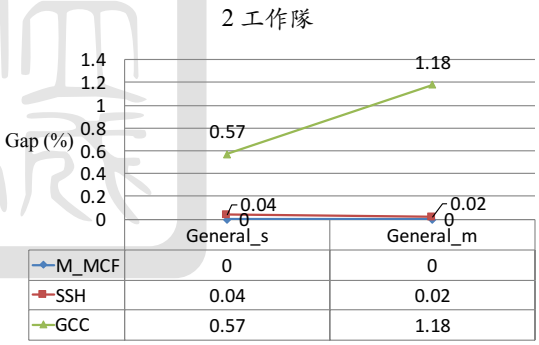
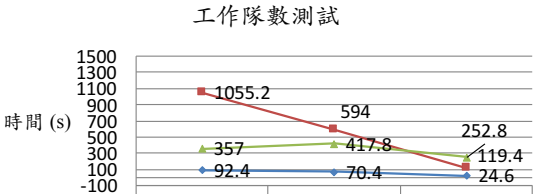


圖 5.8: 多組工作隊一般結構問題之Optimality Gap比較

圖5.9比較不同工作隊數求解時間上的差異，爲了顯現明顯的趨勢，$General_m$的資料只列出在下方，而因爲演算法對於工作隊的差別較無影響所以使用M^m_{MCF}進行分析。從圖中可以發現雖然工作隊數增加，但是普遍上求解時間都有下降的趨勢，對於$Tree_m$網路結構甚至在單一工作隊以及四個工作隊的比較差了將近十倍，可見工作

隊數雖然變多，反而可以加速求解時間。



	1 team	2 teams	4 teams
Tree_s	92.4	70.4	24.6
Tree_m	1055.2	594	119.4
General_s	357	417.8	252.8
General_m	12050.4	5597.4	2985.2

圖 5.9: 不同工作隊數比較

5.4 分段式啓發演算法測試

分段式啓發演算法中，時間區段個數n_{seg}及α值可以進行調整，以下情境設定工作隊數量皆爲2組工作隊並在$General_s$的第一筆測資進行測試。

5.4.1 n_{seg}數固定

若將n_{seg}數固定爲4並測試α值在1、1.3、1.5、1.7、2的表現，可得表5.8。可以發現當α值上升，表示每一次求解時會觀測到更大的網路，所以在求解時間上會花費較長的時間，但是也因爲考慮到更多種可能，所以Gap值也會下降。

表 5.8: n_{seg}數固定

α	時間 (s)	Gap (%)
1	9	0.28
1.3	14	0.05
1.5	18	0.02
1.7	16	0.02
2	26	0.02

5.4.2 α固定

若將α固定爲1，也就是求解的時限與實際決策的時限相同(取消「看得更遠」的機制)，並測試n_{seg}值在1、2、3、4、5的表現，可得表5.9。在n_{seg}爲1時，表示未使用時間切割，所以求得的結果將會與M^m_{MCF}相同。隨著n_{seg}數上升，可以發現因爲將每一回合的網路切得更小，可以使加快求解速度，但又因觀測到的網路較小，考慮較不夠周全則會使Optimality Gap值變大，導致求解效能較差。

表 5.9: α固定

n_{seg}	**時間 (s)**	**Gap (%)**
1	98	0.00
2	22	0.32
3	14	0.05
4	9	0.28
5	3	0.35

5.5 基因演算法測試

基因演算法的測試中，每個回合初始的染色體數量M設定爲8，交配的機率P_c設定爲0.6，突變的機率P_m設定爲0.2。此部分若將突變機率提高，會越接近於亂數隨機決定，所以設定0.2使其有機會可跳出區域最佳解、卻又不會太隨機。針對迭代次數的設定測試，圖5.10到圖5.15分別測試三種方法在中小型的樹狀網路與普通網路的第一個測資下，1、2、4組工作隊求解一萬次迭代的結果。可以發現，整體來看幾乎是在迭代到300次時就可差不多收斂，較難再找到更好的解。所以之後的測試皆使用300次的迭代次數求解中小型的樹狀網路，結果如表5.10所示。

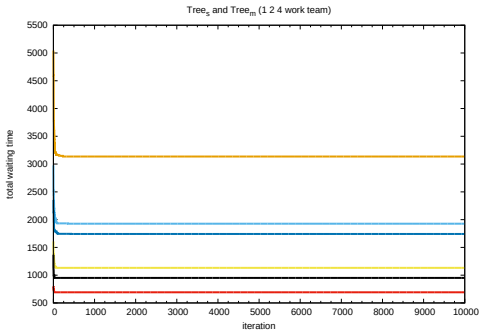


圖 5.10: GA_1之樹狀網路圖測試

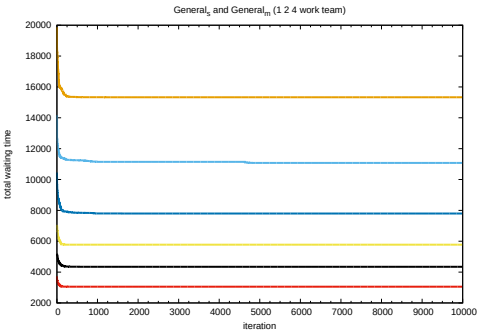


圖 5.11: GA_1之普通網路圖測試

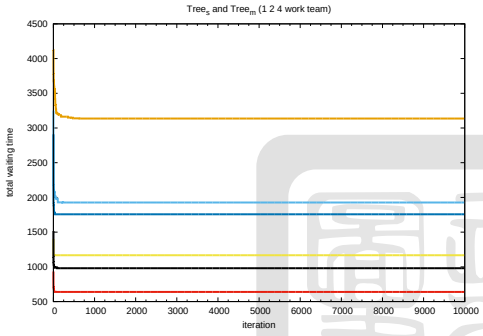


圖 5.12: GA_2之樹狀網路圖測試

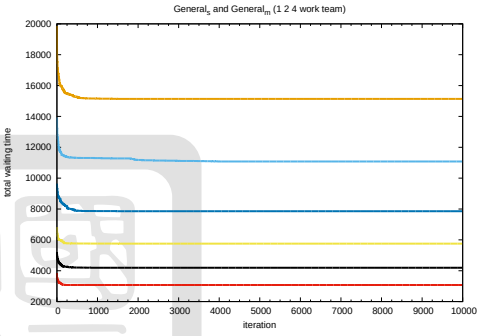


圖 5.13: GA_2之普通網路圖測試

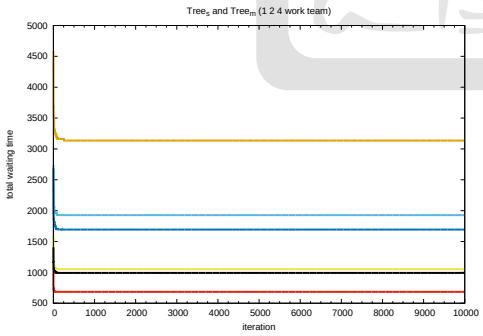


圖 5.14: GA_3之樹狀網路圖測試

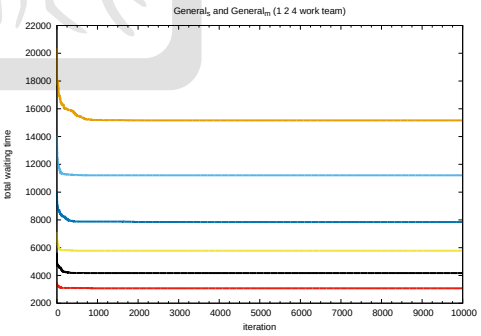


圖 5.15: GA_3之普通網路圖測試

表 5.10: 基因演算法測試(300次迭代)

網路結構	工作隊個數	測資	GA_1		GA_2		GA_3	
			時間 (s)	Gap (%)	時間 (s)	Gap (%)	時間 (s)	Gap (%)
$Tree_s$	1	5	2.2	0.00	2.4	0.18	2	0.01
	2	5	1.6	3.86	1.8	2.80	2	5.38
	4	5	1.4	12.94	1.4	14.86	2	16.44
$Tree_m$	1	5	4	0.28	4.2	0.41	4.8	0.74
	2	5	3.6	8.26	3.6	7.59	3.4	3.89
	4	5	3.2	8.94	3.4	8.99	3	12.32
$General_s$	1	5	5.8	1.44	6	0.75	6.4	1.68
	2	5	5	1.88	5.6	1.81	5.6	1.15
	4	5	4.4	3.06	4.2	2.68	4.4	3.51
$General_m$	1	5	8.8	1.28	8.8	2.32	9.2	2.69
	2	5	7.8	2.63	7.6	2.96	8	2.76
	4	5	6	2.90	6.4	4.41	6.4	4.31

因爲迭代次數只有300次，所以可以發現求解時間皆非常短，而三種方法在Optimality Gap值的表現皆相差不多，推測是因爲基因演算法的隨機找解過程本來就是以較隨機的方式進行，雖然在GA_2及GA_3我們有加入較接近供給點的節線會被選到機率較高的條件，但可能因爲隨機性的影響較大，所以其結果並未明顯優於GA_1。

以上測試採取隨機方式產生初始解，如果考慮初始解在每次隨機產生後，再進行順序調整，以使每組工作隊分配的工作都是由外界向內修的話，結果如表5.11所示，整體來說這種改善初始解的方式仍未能大幅改善其求解表現。

表 5.11: 利用由外界向內修產生初始解之基因演算法測試(300次迭代)

網路結構	工作隊個數	測資	GA_1		GA_2		GA_3	
			時間 (s)	Gap (%)	時間 (s)	Gap (%)	時間 (s)	Gap (%)
$Tree_s$	1	5	2	0.10	2.2	0.01	2.2	0.21
	2	5	1.8	5.92	1.8	5.64	1.8	4.41
	4	5	1.6	18.44	1.4	17.59	1.8	17.03
$Tree_m$	1	5	4.2	0.14	4.2	0.56	4.4	0.66
	2	5	3.6	5.67	3.8	3.33	3.8	3.24
	4	5	2.6	8.14	3.2	13.23	3.2	14.51
$General_s$	1	5	6	0.66	6.2	1.23	6.4	0.65
	2	5	5.2	2.93	5.2	2.38	5.8	1.46
	4	5	4.2	3.01	4.6	2.83	4.2	3.73
$General_m$	1	5	9	2.07	9	1.64	9.4	1.93
	2	5	8	2.85	8	2.97	8.2	3.72
	4	5	6	5.26	6.4	4.65	6.8	4.87

5.6 現實大小之網路測試

在現實生活中，網路的大小不可能只有寥寥無幾的十幾個節點以及節線，爲了驗證本研究提出的方法可以在現實生活中使用，我們分別測試三個大型網路，其網路結構資訊如表5.12所示，其中$General_{l1}$呈現如圖5.16。因爲測試網路爲普通網路，所以分別使用(1)M^m_{MCF}並加入有效不等式3.8.13及3.8.16，(2)*SSH*演算法並將n_{seg}分別設定爲8、11、15、22，其中數字越小表示每次求解的時間區段T越長，且α設定爲1.3，以及(3)*GCC*演算法，並在八組工作隊的情境下進行求解。

表 5.12: 大型網路結構資訊

網路名稱	節點數	節線數	節線修復時間	需求大小
$General_{l1}$	80	238	[3, 5]	[3, 5]
$General_{l2}$	100	450	[3, 5]	[3, 5]
$General_{l3}$	120	670	[3, 5]	[3, 5]

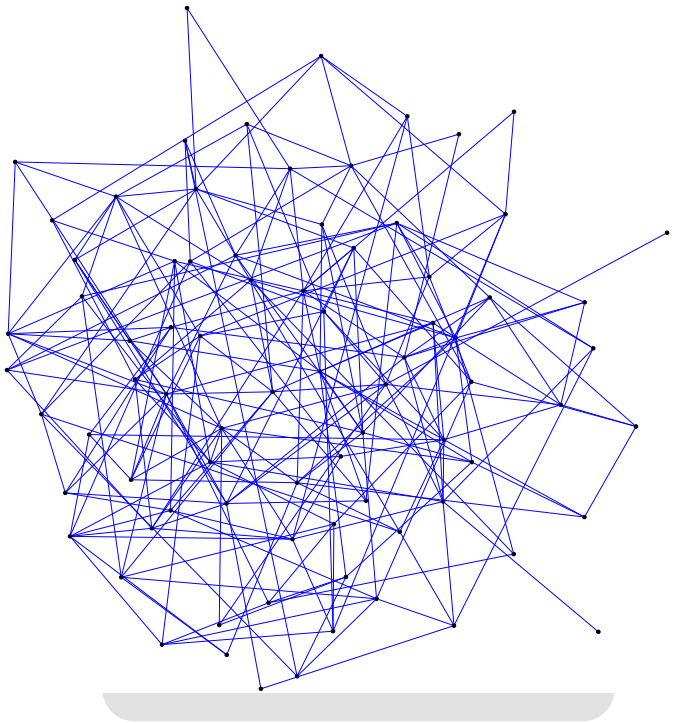


圖 5.16: $General_{l1}$網路圖

結果顯示如下，因爲*SSH*測試多個時間區段參數，故整理於表5.13，並將$n_{seg} = 11$的測試與其他方法進行比較如表5.14。M^m_{MCF}因爲求解的問題過大，16G的記憶體也無法進行模式的求解，即使是$General_{l1}$最小的網路圖，在工作隊設定爲4時其限制式就有3130978條、決策變數有3200456個，皆約三百多萬，工作隊爲8時，求解軟體Gurobi甚至在建立模型還未求解的情況就被迫停止，所以gap的計算是利用求解方法中目標值最小的演算法當作基準。演算法的部分，表5.13可以明顯看出，當n_{seg}值越大時，表示求解時間區段越少，所以每次求解都是針對較小型的網路進行求解，故可以在較短的時間求解完畢，但結果較差。

表 5.13: *SSH*大型網路多組工作隊測試

網路名稱	工作隊個數	n_{seg}	*SSH*	
			時間 (s)	Gap (%)
$General_{l1}$	8	8	273	0.00
		11	98	0.02
		15	53	0.00
		22	32	0.33
$General_{l2}$	8	8	8089	0.00
		11	2194	0.04
		15	585	0.15
		22	361	0.19
$General_{l3}$	8	8	-	-
		11	8929	0.00
		15	5566	0.01
		22	1896	0.10

*gap的計算是利用求解方法中目標值最小的*SSH*演算法當作基準。

表 5.14: 大型網路多組工作隊測試

網路名稱	工作隊個數	M^m_{MCF}		$SSH(n_{seg}=11)$		*GCC*	
		時間 (s)	Gap (%)	時間 (s)	Gap (%)	時間 (s)	Gap (%)
$General_{l1}$	8	-	-	98	0.02	16	0.74
$General_{l2}$	8	-	-	2194	0.04	119	1.83
$General_{l3}$	8	-	-	8929	0.00	477	1.14

*gap的計算是利用求解方法中目標值最小的*SSH*演算法當作基準。

而從表5.14可以發現，其整體表現與小型網路的結果大致相同，亦即*GCC*演算法可以在較短的時間求得可行解，*SSH*則是花費時間較長但可以得到較佳的解。較特別的是在$General_{l1}$的網路測試中，*SSH*在$n_{seg}=15$、22時甚至可以比*GCC*花更短的時

間求得較佳解。整體時間上可以明顯的看出求解較耗時，但是對於如此大的網路，本研究提出的演算法皆可以約在一小時內得到品質不錯的解，且由小型網路可推得其結果應是近似最佳解，此結果顯示本研究提出的演算法確實能夠處理現實大小的網路修復排程問題。

而在初始亂數解的基因演算法測試中，我們分別記錄其在執行1000次及3000次的GAP值，可以發現其解雖然隨著迭代數上升會有所改善，但仍無法與*SSH*以及*GCC*匹敵。

表 5.15: 大型網路基因演算法測試

網路名稱	GA_1 (1000 / 3000 itr.)		GA_2 (1000 / 3000 itr.)		GA_3 (1000 / 3000 itr.)	
	時間(s)	GAP(%)	時間(s)	GAP(%)	時間(s)	GAP(%)
$General_{l1}$	379 / 1137	3.00 / 2.01	362 / 1086	3.24 / 1.62	375 / 1125	3.84 / 2.03
$General_{l2}$	1121 / 3363	3.59 / 0.88	1098 / 3294	5.34 / 1.37	1129 / 3389	4.45 / 1.47
$General_{l3}$	2199 / 6598	6.26 / 1.00	2197 / 6592	8.72 / 2.09	2272 / 6818	7.37 / 2.22

*gap的計算是利用求解方法中目標值最小的*SSH*演算法當作基準。

而在利用「由外界向內修」的方式產生初始解之基因演算法測試中，我們仍然測試並記錄1000次及3000次的GAP值，發現其表現與使用隨機亂數產生初始解的求解表現相差不多，其差異可能只來自於交配與突變時的亂數改變而有些微的不同。

表 5.16: 大型網路基因演算法測試(利用由外界向內修復的順序來產生初始解的方式)

網路名稱	GA_1 (1000 / 3000 itr.)		GA_2 (1000 / 3000 itr.)		GA_3 (1000 / 3000 itr.)	
	時間(s)	GAP(%)	時間(s)	GAP(%)	時間(s)	GAP(%)
$General_{l1}$	388 / 1163	3.10 / 2.28	362 / 1087	3.48 / 1.47	377 / 1131	3.37 / 2.00
$General_{l2}$	1125 / 3376	3.89 / 0.85	1111 / 3333	5.64 / 1.64	1121 / 3363	5.25 / 1.28
$General_{l3}$	2222 / 6665	5.84 / 1.35	2236 / 6707	8.08 / 2.02	2259 / 6776	8.03 / 2.01

*gap的計算是利用求解方法中目標值最小的*SSH*演算法當作基準。

5.7 小結

本章節一開始先介紹測試網路的規模以及參數設定，並將各方法適合的情境列出，再分別針對單一工作隊、多組工作隊在樹狀網路以及一般網路結構進行測試。在單一工作隊方面，M^s的建模方式雖然只適用在單一工作隊，但相對於可以適用在多工作隊的M^m_{MCF}模式，其求解時間相對有效率；而*GT*演算法雖然只適用在樹狀結構，但其結果在演算法的比較中是最接近最佳解的方法。在多組工作隊方面，M^m_{MCF}解決$M^m_{B\&C}$因爲B&C造成求解效率緩慢的問題，但在網路規模稍微變大時，求解時間便會大幅增加。本研究提出的四個有效不等式經過測試後，發現確實能降低求解時間，尤其以限制式(3.8.13)表現最佳；在有效不等式的組合測試中，也發現將全部的有效不等式一併加入未必能夠得到較好的結果。多組工作隊演算法部分，測試的結果皆能在極短的時間內獲得近似最佳解，其中Optimality Gap值皆約略在1%左右。接著調整*SSH*演算法的參數，並觀察每回合求解時段的長度以及其往後觀測幅度的變化結果，可以發現α上升時可以使求解品質上升但需要花費更長的時間，此乃因爲在每回合會考慮更大的網路；n_{seg}變大時則可以使求解時間下降但是其結果會變得相對較差，因爲考慮的時間區段太小，導致查看的網路不夠大，造成修復排程效果較差。在基因演算法的測試中，先在其中一個網路圖中經由一萬次迭代的測試後，發現中小型網路差不多迭代約300次就會收斂，實際測試後發現三種基因演算法的結果皆差距不大，在求解十秒左右的效能Optimality Gap值約在3到10%左右。最後測試一個較符合現實的大規模網路，可以發現本研究提出的演算法仍可以在有效時間內求得不錯的可行解，而GA表現也都相對較差。

根據實驗結果，本研究提出的M^m_{MCF}整數規劃模式可以在多組工作隊且各種網路結構中使用，搭配*SSH*演算法架構及貪婪演算法皆能在適合的情境得到近似最佳解。

第六章　結論與未來研究議題建議

本研究探討當輸送水、電、天然氣、通訊網路等物資的管線受到天災或人禍的破壞而毀損時，該如何派遣工作隊修復所有毀損的節線，以使各毀損節線上的災民需求能儘快被滿足。以往的修復排程決策方式，經常藉由決策者自身的經驗與接收到的部分需求及管線毀損資訊來擬定，未有系統化的科學理論基礎，因此往往讓災民枯等許久才能自修好的管線接收物資。本研究經由數學理論分析推導，建立數種整數規劃模式以求解理論上的最佳修復排程，由於求解理論最佳排程通常耗時甚久，因此我們亦設計多種兼具效率與效能的演算法以在較短時間內可求得品質不錯的管線修復排程。本章將整理並臚列本研究的貢獻與結論，並建議數個未來可再繼續探討的相關議題。

6.1　結論

本研究探討一網路修復問題，在日常生活中有許多的管線埋藏在地底下形成一個互相連接的網路，其節線上的需求大小可以想成是該管線使用的居民人口多寡的程度。如果遭遇重大事故則可能造成損壞，導致該節線無法連通供給點，此與過往文獻大都將需求設定在節點上更爲實際，而本研究的目標即是希望探討如何派遣多組工作隊進行修復，使需求總等待時間最小。

在第一章中，我們先提出供水網路來舉例，若有一供給點能夠提供所有管線上的需求，但必須要與其能夠相連接，如果無法經由已修復好的節線連接到供給點，那麼該節線上的需求就不會被滿足。在假設工作隊能力一樣的情況，修復順序的不同，其導致等待救援的需求總等待時間也會不一樣，且經由實驗證明若還有節線未修復，工作隊一旦空閒即會進行剩下節線的修復，不會有閒置等待的情況；但在假設工作隊能力不一樣時，則可能會閒置而不工作，等待另外一組對該節線有較高效率的工作隊進行修復。

第二章整理與本研究相關的網路修復問題文獻，主要分成兩大類型，分別爲

「整合網路設計與排程問題」以及「具資源限制下之專案排程問題」。在整合網路設計與排程問題中，Nurre and Sharkey (2014) 證明皆爲*NP-hard*問題，本研究利用$P_m|\sum SP|C_{max}-Threshold$可以將本問題簡化成集合覆蓋問題來證明其爲*NP-hard*的方式，在第三章證明本研究也爲*NP-hard*問題。接著將相關排程文獻依四大特性加以整理：考慮是否爲多組工作隊合作、是否滿足需求、目標爲修復造成的流量、及需求分佈情況(在節點或者節線上)，本研究與文獻最大的差別在於先前文獻大都將需求設定在節點而非節線上，且本研究考慮的網路沒有限制任何特殊架構，且必須修復所有的毀損節線。具資源限制下之專案排程問題則是爲了處理PERT網路圖通常使用資源無上限的不合理假設，而該建模方式所考慮的假設與方法，可以將其運用在本研究的單一工作隊模型中。

第三章詳細描述本研究的問題以及假設，解釋爲何此問題爲*NP-hard*，接著利用網路縮減的方式，在不影響結果的情況下將求解的網路縮小以提高求解的效率。在單一工作隊的特殊情況下，提出基於具資源限制下之專案排程問題的M^s整數規劃模式，其利用由供給點往外修復的限制方式，能有效提升M^s的求解效率；在多組工作隊的情況先提出$M^m_{B\&C}$，需利用B&C方式來檢查是否有獨立子迴圈的產生，獨立子迴圈爲未與供給點連通的迴圈，以$M^m_{B\&C}$求解較大規模的網路時，將會耗時甚久，故本研究提出另一種數學模式M^m_{MCF}，利用節點的需求是否被滿足來判斷其與供給點的連通性，並將網路縮減所造成的自體迴圈上的需求，視同節點上的需求建構整數規劃模式。由第五章的數值測試結果可看出M^m_{MCF}較$M^m_{B\&C}$有更佳的求解效率。此外，針對M^m_{MCF}我們也提出四個有效不等式，經過第五章的驗證也可以發現這些有效不等式個別的影響與其綜效。最後針對工作隊能力不同的情況提出數學模式的修改方式，並發現當工作隊能力不一樣時，最佳的排程決策有可能讓某些工作隊偶爾休息，這也顛覆了傳統決策者希望所有工作隊在結束前不得休息的錯誤排程直覺。

第四章考慮本研究問題一旦在網路規模變大、節點數以及節線數上升的情況會造成求解無效率，提出數個求解演算法。其中，分段式發演算法從整個網路中，利用總修復時間切段的方式，將考慮的節線排程規劃縮小，從供給點開始由近而遠地求解整個網路修復排程，藉由時間區段個數n_{seg}以及可以查看更大網路的α值的調整，可知較大的α可「看得較遠」因此其求解品質較佳，但也需要求解較久；較大

的n_{seg}(切成較多段，因此每段時間較短)求解較快但其排程品質會相對較差。而利用單一工作隊修復所有毀損管線的特例，本研究分別提出窮舉法、貪婪樹生成法及貪婪連通分量生成法等三種演算法，其中，窮舉法可用來檢查小型網路結果是否正確，因其方法是將所有可能展開，所以不適合求解大型網路；後兩演算法皆是用需求對修復時間的比值當節線權重值，優先選取較大權重值的節線來建構排程決策；其中，貪婪樹生成法主要針對樹狀網路結構，經過實驗證明可以得到非常近似最佳解的結果；貪婪連通分量生成法則可處理任何網路結構的排程，經由實驗證明發現也可以得到近似最佳解。此種利用單一工作隊特例得到的節線修復排程，可再藉由FCFS的方式將各節線分配給多組工作隊修復。除了貪婪演算法，本研究也提出在排程領域廣爲使用的基因演算法，並設計符合本研究網路問題的交配、突變、與初始解產生策略，或許因爲尋找解的過程較隨機，測試結果顯示此3種基因演算法表現差異不大，且其排程品質普遍劣於貪婪演算法的結果。

綜合以上結論，本研究具體貢獻如下：

1. **將需求設定在節線上：** 過去文獻皆是將需求設定在節點上，本研究將需求設定在節線上，因此更符合現實之考量，發展一系列相關的數學模式以及演算法進行求解。(Section 3-1)

2. **提出「單一工作隊的整數規劃模式」M^s：** 利用由供給點往外修復的限制方式，能有效提升單一工作隊的求解效率。(Section 3-6)

3. **提出「使用Branch-and-Cut整數規劃模式」$M^m_{B\&C}$：** 利用Branch-and-Cut的方式，當求解時有獨立子迴圈產生時加入限制式限制之以判斷其與供給點的連通性，但此方法會隨著網路規模變大而大幅增加求解時間。 (Section 3-7)

4. **提出「使用多商品網路流量之整數規劃模式」M^m_{MCF}：** 利用多商品網路流量的建模方式，查看節點的需求是否被滿足來判斷其與供給點的連通性，大幅縮減求解時間。(Section 3-8)

5. **提出「有效不等式的設計與測試」：** 針對「使用多商品網路流量之整數規劃模式」提出四項有效不等式，再進一步加速求解，並測試其個別效益與綜

效。(Section 3-8)

6. **提出「可處理能力不同的工作隊修復排程之整數規劃模式」：** 爲相關研究中首次將個別工作隊的不同能力、效率列入建模考慮。(Section 3-9)

7. **客製化「分段式啓發演算法」：** 修改文獻的解法以適用本研究問題，在一開始決定要切割的時間長度後，逐步地針對時間區段進行管線修復以及工作隊分配。(Section 4-1)

8. **提出兩種貪婪演算法：** 提出(1)「貪婪樹生成法」，利用展開樹的概念，計算節線的權重(需求與修復時間之比值)來決定節線的修復排程；(2)「貪婪連通分量生成法」，利用連通分量(component)的概念計算節線的權重。此兩種方法皆可以在短時間內得到近似最佳解。(Section 4-3)

9. **客製化「基因演算法」：** 提出3種基因演算法：(1)最原始的多平行機台基因演算法套用在本研究問題；(2)先計算各節線的權重並在交配策略時，給予有較高權重的節線有先被排到的可能；(3)同第二種，但在固定每個工作隊的第一項工作後，重新計算權重值。經實驗證明此三種基因演算法表現雷同，但也能在較短時間內估得近似解。(Section 4-4)

10. **測資設計與演算法效率與效能實證：** 經測試後本研究提出的演算法不僅可以在小型網路求得近似最佳解，在現實情況大小的大型網路，亦即節點及節線數皆爲上百個的情況下，也能夠在一小時內甚至更短求得近似最佳解，說明本研究提出的演算法具有其效率性，與Nurre and Sharkey (2014)所測試的通訊網路相比，我們曾測試過多達670條毀損節線的大型網路，超過Nurre and Sharkey (2014) 所測過最大僅548條節線(包含毀損與未毀損)的網路。SSH與GCC演算法的Optimality Gap大都可在2%以內，基因演算法的表現略差，但Optimality Gap亦大多可在5%以內。因此，在精確最佳解數學模式部分，我們推薦使用結合有效不等式的M_{MCF}^{m}，而求解演算法部分則推薦SSH及GCC (Chapter 5)

6.2 建議之未來研究議題

本研究針對管線的修復討論網路修復問題，但尚有許多可以更進一步探討的方向，以下列出未來可以繼續延伸之議題：

1. **繼續精進發展本問題之數學規劃模式與演算法研究：**

   雖說我們已盡力發展多種數學規劃模式及求解演算法，然而這方面可能還有不少進步空間。特別是在求解演算法的部分，譬如基因演算法或許可嘗試不同的染色體編碼方式等。

2. **整合節線需求及修復時間之即時偵測與修復排程模式研究：**

   在本研究的問題，目前是假設修復單位對於欲修復的節線需求以及修復時間爲已知，但在現實情況經常需要經過調查後才能知道這些資訊，在相關議題上可以討論，如何即時(on-line)兼顧修復以及偵測，並安排哪些工作隊先去偵測哪一些區域，可對於整體修復工作花費較少時間並偵測及滿足較高需求，以達到高效率的修復。在作法上或許可以將其分成不同類型的工作隊如偵測組及修復組，便可以利用此兩種工作隊進行建模，演算法的部分則必須同時考量兩種工作隊的更新來進行方法的修改。

3. **整合管線修復與供水車服務排程模式研究：**

   在管線修復時，爲了滿足民眾無水可使用的情況，常常會派出供水車提供民用水的需求，以使商店或者自用水需求可滿足，對於一個區域而言，要派多少供水車且需如何巡繞需供水的區域，可以想像成是一個車輛途程問題(Vehicle Routing Problem，VRP)，結合修復節線去滿足供水需求，最小化整體的總等待用水，便是可以討論的議題。

4. **整合管線修復與連通節線水壓調控排程模式研究：**

   在本研究的問題，一旦節線修復且與供給點連通，供水即可到達，此假設沒有考慮水壓問題，因在現實生活中可能會遇到情形是爲了將水能夠先送到地勢較高的區域，而必需暫時停止地勢較低的供水，使水壓夠強足以向上輸送。一地勢較高的區域需求非常大且管線皆完好，這時就會希望水能夠到達以減少需求

總等待時間，如果考慮水壓問題就必須停止供水給地勢較低且需求較少的幾條管線，以達到可以輸水到高處的目的。而在此種因水壓而造成可能的問題可以將其流量的限制加入數學模式內，且在管線需求是否滿足的判斷上，就不能單純以是否連通爲條件，而必須再加入連通但水壓是否足夠的限制，比如說水壓大於多少才能夠供水；啓發式演算法方面，節線權重值的衡量必須減少因水壓而可能到達不了的區域，且在方法設計上修復與供水可能需要分開但同時考量。

5. **整合管線修復排程與明管建置決策模式研究：**

現實情況有時因爲節線需花費較長的時間進行修復，但該區域的需求卻又非常大時，會先使用明管的方式來應急供水，而明管與暗管的差別在於暗管是埋在地底下，較不會影響人民的生活，若考慮明管的配置，因爲是額外的管線配置，且在地面上會造成生活機能的影響，勢必會對整體修復成本造成變動，但若又需提供水的需求，修復單位該如何拿捏其成本的配置便是一個重要的問題，在模型的建置上或許可以考量若要安裝明管的話，可以想成是節點之間有額外的節線連接且會有額外的成本，目標式便除了滿足需求的總等待時間之外，還需將此造成的額外成本考慮進來。

6. **多層管路修復排程決策模式研究**：

目前問題只有考量供水一種管路，若將天然氣、電力等管線一併考慮進來的話，在修復開挖的過程中，勢必會有開了在挖再掩蓋又再開挖的可能性，在這種多階層的管線修復問題，要如何進行修復排程才能讓各種需求的總等待時間最小。此在設計上或許可以考量不同管線及不同的修復成本，而不同種類的管線之間可以利用虛擬節線來進行連接，亦即如果有三種類型的管線，網路圖看起來就會像是三層網路之間有兩層虛擬節線的網路，利用此網路的產生方式可以來模擬並建立模型，而重複開挖會不會對管線造成損壞、有沒有先後順序的問題便可以再考量。

參考文獻

Averbakh, I. (2012). Emergency path restoration problems. *Discrete Optimization*, *9*(1), 58–64.

Averbakh, I., & Pereira, J. (2012). The flowtime network construction problem. *IIE Transactions*, *44*(8), 681–694.

Baxter, M., Elgindy, T., Ernst, A. T., Kalinowski, T., & Savelsbergh, M. W. (2014). Incremental network design with shortest paths. *European Journal of Operational Research*, *238*(3), 675–684.

Blazewicz, J., Lenstra, J. K., & Kan, A. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11–24.

Brucker, P. (2007). *Scheduling algorithms* (Vol. 3). Springer.

Cavdaroglu, B., Hammel, E., Mitchell, J. E., Sharkey, T. C., & Wallace, W. A. (2013). Integrating restoration and scheduling decisions for disrupted interdependent infrastructure systems. *Annals of Operations Research*, *203*(1), 279–294.

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.

Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, *11*(2), 109–124.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, *5*, 287–326.

Huang, T. M. (2015). *A multi-mode network restoration problem in post-disaster humanitarian logistics management* (Master's thesis). National Cheng Kung University, Tainan, Taiwan.

Kalinowski, T., Matsypura, D., & Savelsbergh, M. W. (2015). Incremental network design with maximum flows. *European Journal of Operational Research*, *242*(1), 51–62.

Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of computer computations*, 85–103.

Matisziw, T. C., Murray, A. T., & Grubesic, T. H. (2010). Strategic network restoration. *Networks and Spatial Economics*, *10*(3), 345–361.

Nurre, S. G., Cavdaroglu, B., Mitchell, J. E., Sharkey, T. C., & Wallace, 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. (2014). Integrated network design and scheduling problems with parallel identical machines: Complexity results and dispatching rules. *Networks*, *63*(4), 306–326.

Padberg, M., & Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, *33*(1), 60–100.

Talbot, F. B. (1982). Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, *28*(10), 1197–1210.

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.

Wu, M. C. (2010). *Biking route planning based on target calorie comsumption* (Master's thesis). National Cheng Kung University, Tainan, Taiwan.

Xu, N., Guikema, S. D., Davidson, R. A., Nozick, L. K., Çağnan, Z., & Vaziri, K. (2007). Optimizing scheduling of post-earthquake electric power restoration tasks. *Earthquake engineering & structural dynamics*, *36*(2), 265–284.