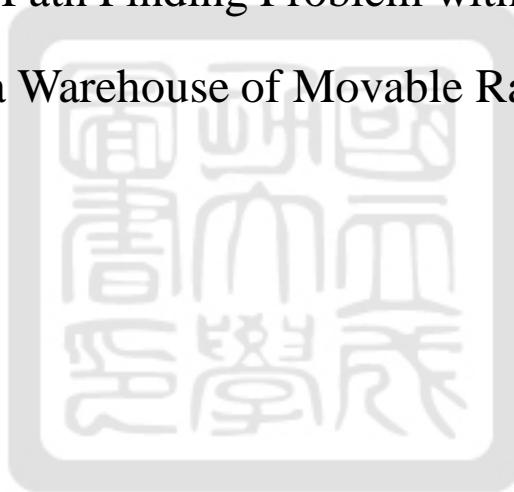


國立成功大學
工業與資訊管理學系 碩士班
碩士論文

考量移動式貨架選擇機制之多台倉儲機器人
最佳揀貨作業路徑規劃研究

A Multi-Agent Path Finding Problem with Rack Selection
in a Warehouse of Movable Racks



研究生：王宗瀚
指導教授：王逸琳 博士
中華民國一百零九年五月

國立成功大學

碩士論文

考量移動式貨架選擇機制之多台倉儲機器人最佳揀貨
作業路徑規劃研究

A Multi-Agent Path Finding Problem with Rack
Selection in a Warehouse of Movable Racks

研究生：王宗瀚

本論文業經審查及口試合格特此證明

論文考試委員： 王逸琳

丁慶榮

解翠萍

指導教授： 王逸琳

系(所)主管：

惠嘉

中 華 民 國 109 年 5 月 29 日

摘要

隨著線上購物風氣盛行，物流需求量與配送頻率亦日益增長。新型態的物流倉儲系統捨棄傳統整面的固架式儲位，以眾多可移動式的單一立體貨架來儲存產品，各貨架均可被一台小型機器人自底部頂起而在倉庫內移動。此類倉儲系統可充分彈性運用閒置之貨架與倉庫空間，不似傳統固架式倉儲系統常陷於不當的儲位規劃以及狹窄之揀貨通道，進而導致多餘、繁忙又不順暢之揀貨作業。過去相關研究大多將倉儲內之訂單揀貨流程分為(1)貨架選擇問題與(2)多機器人路徑規劃問題(Multi-Agent Path Finding, MAPF)等兩階段流程來個別探討。然而，現實中此兩階段流程的決策息息相關，理應要一併考量才能得到更好的整體物流效率。其中第一階段的相關文獻大多考量選取較少貨架，卻忽略各貨架之實際位置，導致其第二階段可能會走較久的路線；而第二階段決策的相關文獻大多假設每貨架皆配置一台專屬機器人，亦即機器人與貨架有相同數量，將機器人數量遠少於貨架的現實大幅簡化，導致規劃而得之移動路線效能不佳。此外，針對結束揀貨作業的貨架該被移至何處停留，亦無文獻加以著墨。

本研究試圖將上述兩階段物流決策及其衍生的相關問題一併處理，假設各台機器人與各貨架之初始位置、貨架上之貨物品項與數量、各站訂單需求等資訊皆為已知，探討如何指揮調度機器人去移動貨架，期以最快方式完成規劃期間所有訂單之揀貨、包裝、出口等作業。上述之揀貨作業包含指揮機器人移動至合適的貨架，將該貨架移至倉庫內的某揀貨站點，待駐守該站點的揀貨人員完成包裝作業後，再將該貨架移至合宜的暫存位置等數個步驟。因同時有多台機器人與貨架在倉庫內移動，如何避免彼此碰撞、阻塞而能以最快方式完成揀貨任務的路線規劃問題，其本質是一個 NP-hard 的多元商品網路流量問題。由於本議題十分新穎，除結合過去的兩階段倉儲內物流路線規劃外，更考量了機器人少於貨架個數的現實狀況，同時規劃機器人與貨架在兩階段流程中的最佳移動路徑。本研究使用「層空網路」圖的架構，以整數規劃模式處理機器人與貨架的指派以及避撞路徑之規劃，以在最短的時間內完成所有的訂單出貨；本研究也提出貪婪演算法，以及一滾動式求解演算法，將問題依時段切分成數個較小規模的問題再依序求解，以加速整體的求解過程；數值測試結果顯示本研究設計之求解演算法可在短時間內得到不錯的可行解，可供相關系統在其實務應用中參考使用。

關鍵詞：多機器人路徑規劃；移動式貨架選擇；移動式貨架倉儲；多元商品網路流量

A Multi-Agent Path Finding Problem with Rack Selection in a Warehouse of Movable Racks

Tsung-Han Wang

I-Lin Wang

Department of Industrial and Information Management

SUMMARY

We investigate a realistic problem from a Robotic Movable Fulfillment System (RMFS), where racks containing items are carried by robots to workstations for finishing orders. For a planning horizon, we seek efficient and effective ways to route robots and racks with minimum makespan or total waiting time for collecting all items in given orders. Related literature usually simplifies this complex problem to two easier subproblems: (1) movable rack selection to assign racks to workstations, and (2) multi-agent path finding (MAPF) to plan collision-free routings for racks (as robots). To the best of our knowledge, this thesis may arguably be the first that deals with both subproblems at the same time. Also, we further route a just-off-duty rack from a workstation to its destined storage region for avoiding congestion near the workstation. This problem is NP-hard since its MAPF subproblem is already NP-hard. We propose a mixed integer programming (MIP) formulation similar to a multi-commodity network flow model on a level-space network. The MIP calculates an exact optimal solution but is time-consuming in practice. We introduce a rolling-horizon solution mechanism that divides the planning horizon into several shorter periods and solve the same MIP over a few consecutive periods in an iterative fashion. We propose a greedy algorithm to calculate a good quick solution. We also design three order sorting heuristics to improve the solution quality. Computational experiments indicate the total waiting time a better objective to optimize than the makespan. Some intriguing cases are reported where our MIP solution suggests carryover operations between robots that could never be considered by any MAPF-like heuristics. For such cases, the optimality gap by efficient heuristics may be up to 30% or more, which shows the advantages of the proposed MIP formulation.

Key words: Multi-Agent Path Finding, Movable Rack Selection, Robotic Movable Fulfillment System, Multi-Commodity Network Flow

Introduction

In the past, racks storing items in a warehouse are usually made of heavy and unmovable steel frames. After receiving orders, an agent (staff) visits racks to collect items and returns workstation to finish orders. Such a storage and order picking system lacks flexibility. It requires proper planning in many aspects, including the layout design, storage allocation, item deployment, and pick-up routing plans. As the technologies in automation and sensing become more sophisticated, the Robotic Movable Fulfillment System (RMFS) has become prevalent in modern warehouses. In an RMFS, small racks are movable by robots. Due to this flexibility, the optimization in the layout design, storage allocation, and item deployment can be achieved with more ease. Staff training and recruitment costs can also be saved. The challenges of RMFS involve four subproblems: (1) which rack (for collecting items) to select and picking station (for packing) to visit for a selected rack, (2) which robot to carry which rack, (3) how to plan a collision-free path for an on-duty rack destined to a workstation, and (4) how to plan a collision-free path for a just-off-duty rack from a workstation back to storage regions. Related RMFS literature mainly focuses on part of the subproblems (1) and (3), but not (2) and (4) at all. To the best of our knowledge, this thesis is arguably the first research that considers all four subproblems in an integrated way.

In particular, the subproblem (1), the movable rack selection problem, ignores the robot-rack and rack-workstation transportation time and route congestion. The subproblem (3), the Multi-Agent Path Finding (MAPF), assumes each rack is a robot, and skips relocating those just-off-duty racks/robots. In practice, racks are many more than robots. The MAPF should route some robots to reach those to-be-carried racks. Moreover, a route for the just-off-duty rack from a workstation to a storage region should also be planned.

MATERIALS AND METHODS

This thesis deals with orders in a planning horizon.. Assume all the orders to be handled, the items and location for each rack, and the location for each robot are given in the beginning of the period. We seek the best robot-rack and the rack-workstation assignments and routings, as well as the rack-storage routings to minimize the makespan or total waiting time (i.e., the sum of all the order finishing times) without any collision.

We convert the movable space in a warehouse as a grid-like network, where each node represents a space for an agent (i.e., rack or robot) to occupy, and a unit-length edge (horizontal or vertical) between adjacent nodes represents a possible movement (of the same distance) that allows only one agent at one time, too. To record the movement of an agent

over time, we include the time dimension with the space network to construct an acyclic level-space network. A path for an agent in this network represents its routing over time. We propose a mix integer programming (MIP) model M_0 to calculate the best agent path with minimum makespan or total waiting time. For larger-scale cases, we propose an iterative solution mechanism M_{Iter} that solves part of the orders at a time in an iterative fashion. For example, we may solve three orders each time, and then fix the planned agent paths for the top two orders of the earliest finishing times. We secure the planned paths over time and repeat the same steps for the next three orders. Since this mechanism is still time-consuming for larger cases, we propose a greedy algorithm that calculates a feasible solution quickly. We further propose three order sorting heuristics applicable for improving the solution quality by the proposed M_{Iter} and greedy algorithm.

RESULTS AND DISCUSSION

Our testings are performed on a personal computer with Windows 10, Intel Core i7-87000 3.20 GHz*6 Processors, and 8GB RAM. All the solution methods are implemented in Python language. Gurobi 9.0.2 version is used for solving MIP. We have tested grid networks containing 9, 16, 25, 36, 49, and 64 nodes with randomly generated orders, items, racks, and robots. For the 36-node cases, the MIP could not find a feasible solution within 1 hour, whereas M_{Iter} could solve the 64-node case within 10 minutes. The greedy algorithm calculates a feasible solution within a few seconds, but the solution gaps range could vary between 0% to 40%, with the majority of 10%~20% gaps.

CONCLUSIONS

In this thesis, we have investigated a path planning problem with rack selection in an RMFS warehouse. We consider a more complete order picking process that solves the best robot-rack, rack-workstation, rack-storage assignments and routings. We extend the rack selection problem to deal with multiple workstations. We construct a level-space network and use it to formulate an MIP, arguably the first exact solution method, to deal with the four subproblems in one shot. We propose an iterative solution mechanism that takes advantage of MIP while speeding up the solution process. Our proposed greedy algorithm can quickly calculate a good feasible solution. Our computational experiments show that our greedy algorithm is very efficient, although its effectiveness is hard to guarantee. We suggest investigating speed-up techniques for exact solution methods, or new metaheuristics that can encode, decode, and converge solutions in some more effective way.

致謝

感謝我的指導老師王逸琳老師，這些日子以來在學業上的指導與生活上的關心。在研究所期間透過參加許多競賽來磨練我，讓我從中成長，也因此獲得了出國見識與報告的機會，透過種種扎實的訓練，使我在這兩年間有了明顯的成長，也有了更多的自信。也因為老師的指導，我才能完成題目設定龐大的論文。老師對於研究的態度與投入，也令我學習到很多。

感謝實驗室的學長姊，BK、思涵、冠緯、嘉豪、昀軒、雪湄，在我剛進實驗室什麼都還不熟悉的時候給予我很多幫助，特別感謝嘉豪都會跟我分享有趣的事或是實用的程式小技巧，以及給了我許多求職的建議。

感謝同屆的彥瑋以及晏慈，從碩0開始一起參與了各種競賽，還有一起出國真的是很難得的經驗，與你們一起討論論文與未來工作的事也令我沒那麼徬徨。特別感謝彥瑋平常都會跟我不限題材的亂聊紓解壓力，我們從大學開始到現在也算是一起經歷過非常多好的壞的的事，雖然以後我就回新竹了但還是要常聯絡。

感謝lab學弟書桓與承中，每次去學校都會找你們看有沒有新八卦，不過也很抱歉好像沒有教你們太多東西，希望你們的論文也能順利完成。

感謝一起來成大的政達、杰穎、大哥，偶爾一起聚餐，還有工工傳說團的大家，讓我在做研究之餘能有適當的放鬆。

最後感謝我的家人，讓我可以沒有負擔的完成學業，以及放心的探索自己想做的事。謝謝！

目錄

摘要	I
表目錄	VIII
圖目錄	IX
1 第一章 緒論	1
1.1 研究動機	3
1.2 研究目的	4
1.3 論文架構	5
2 第二章 文獻探討	6
2.1 貨架選擇問題	6
2.2 多機器人路徑規劃及其相關的變形問題	7
2.2.1 多機器人路徑規劃問題(MAPF)	8
2.2.2 包裹可交換式機器人路徑規劃(PERR)	12
2.3 移動式倉儲其它相關文獻	13
2.4 小結	14
3 第三章 多台移動式倉儲機器人最佳揀貨作業路徑規劃模式	15
3.1 問題描述	15
3.2 問題假設	16
3.3 網路架構	17
3.4 數學模式	19
3.4.1 符號定義	19
3.4.2 目標式	20
3.4.3 限制式	21
3.5 小結	22
4 第四章 多台移動式倉儲機器人最佳揀貨作業路徑規劃演算法	24
4.1 滾動式求解	24
4.2 貪婪演算法	26
4.3 重新排列訂單	30
4.4 小結	35
5 第五章 數值測試與討論	36
5.1 測試資料參數設定	36
5.2 目標式不同之比較	37
5.3 數學模式與演算法之比較	38
5.3.1 一次性求解與批次求解	38
5.3.2 數學模式與貪婪演算法	39
5.4 小結	43
6 第六章 結論與未來研究方向建議	44
6.1 結論與貢獻	44

6.2 建議之未來研究方向.....	46
參考文獻	48



表目錄

表 2.1 貨架交換相關文獻比較	7
表 2.2 MAPF 相關文獻比較	11
表 4.1 滾動式求解流程	24
表 4.2 貪婪演算法流程	27
表 4.3 遞迴機器人路徑搜尋	28
表 4.4 遞迴貨架路徑搜尋	29
表 4.5 刪除節線流程	30
表 4.6 訂單重新排序流程	32
表 5.1 參數設定	36
表 5.2 目標式不同之比較	37
表 5.3 目標式不同最遲完工時刻之比較	38
表 5.4 一次性與批次數學模式比較	39
表 5.5 數學模式與貪婪演算法比較	39
表 5.6 滾動式求解與貪婪演算法比較	40
表 5.7 範例訂單分布	41
表 5.8 範例可用貨架分布	41

圖目錄

圖 1.1 移動式貨架倉儲佈置(Wurman et al., 2008)	1
圖 1.2 靠機器人移動之貨架(Wurman et al., 2008)	2
圖 1.3 貨架選擇示意圖	4
圖 2.1 時空網路圖	8
圖 2.2 CT 限制生成	10
圖 2.3 CBS-TA 架構(Hönig et al., 2018)	11
圖 3.1 PERR 交換機制	15
圖 3.2 本研究交換機制	16
圖 3.3 儲存區域示意圖	17
圖 3.4 對角移動	17
圖 3.5 時空網路圖跨時空節線—以節點 1 為例	18
圖 3.6 層空網路圖—以九宮格為例	18
圖 4.1 16 節點範例	25
圖 4.2 滾動式求解 16 節點範例第一步	25
圖 4.3 滾動式求解 16 點範例第二步	26
圖 4.4 排序範例	30
圖 4.5 法 2 步驟一	33
圖 4.6 法 2 步驟二	33
圖 4.7 法 3 步驟一	34
圖 4.8 法 3 步驟二	34
圖 4.9 法 3 步驟三	34
圖 5.1 倉儲佈置	36
圖 5.2 16 節點範例	40
圖 5.3 模式求解(1)	41
圖 5.4 模式求解(2)	41
圖 5.5 模式求解(3)	41
圖 5.6 模式求解(4)	41
圖 5.7 模式求解(5)	41
圖 5.8 模式求解(6)	41
圖 5.9 模式求解(7)	42
圖 5.10 模式求解(8)	42
圖 5.11 模式求解(9)	42
圖 5.12 模式求解(10)	42
圖 5.13 模式求解(11)	42
圖 5.14 模式求解(12)	42
圖 5.15 模式求解(13)	42

圖 5.16 模式求解(14).....	42
圖 5.17 模式求解(15).....	42
圖 5.18 模式求解(16).....	42
圖 5.19 模式求解(17).....	42
圖 5.20 模式求解(18).....	42
圖 5.21 模式求解(19).....	43



第一章 緒論

隨著線上購物風氣的盛行，如何快速地完成訂單的揀貨作業以應付大量的需求成了重要的課題，de Koster, Le-Duc, & Roodbergen (2007)的研究中，大致將一個訂單完成的流程分成幾個部分：倉儲的布局(Layout design)、儲位指派(Storage assignment)、倉儲內分區(Zoning)、訂單分組(Batching)、路徑規劃(Routing methods)等問題。在過去，倉儲多採用固定式的貨架，因此倉儲的貨架擺放通常在最初建廠時的倉儲布局階段皆已固定，因為固定式的貨架通常又大又重，若是要改變貨架的配置並不容易，同時產品的儲位規劃一旦確立，若有需要變動也非常困難。隨著科技的進步，自動化的設備如機器人逐漸被運用於倉儲內的物體運送。在傳統固定式貨架的倉儲內，儘管使用機器人能加快取貨效率，但因貨架不易移動，大幅限縮機器人移動的彈性。現有新興型態的倉儲系統(Wurman, D'Andrea, & Mountz, 2008)佈置如圖 1.1，在該系統下會有許多靠機器人移動的貨架（如圖 1.2）存放於圖 1.1 右方深色方格區域，每個貨架上可同時存放多種不同的產品，同一種產品也可存放於多個不同的貨架上，此種貨架能透過機器人從貨架底部頂起而隨著機器人移動至適當的位置，例如根據訂單的需求由單一機器人或多個機器人接力移動至揀貨工作站並由操作員取出所需要的產品，機器人則依據地板上的條碼讀取來辨別移動的方向。

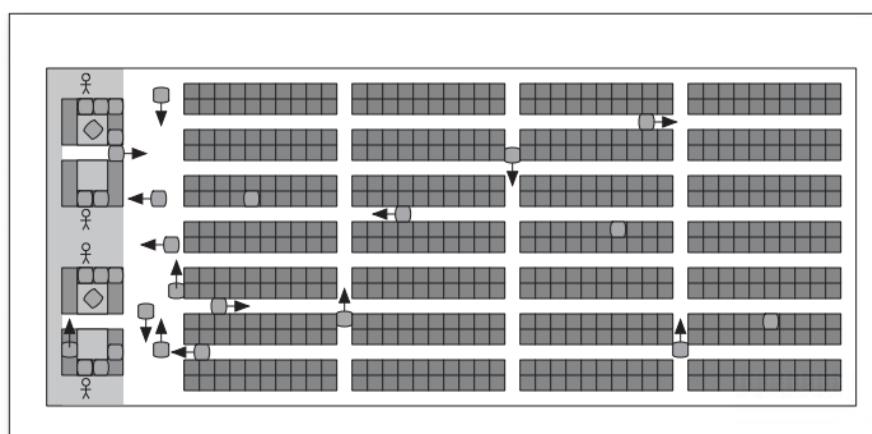


圖 1.1 移動式貨架倉儲佈置(Wurman et al., 2008)



圖 1.2 靠機器人移動之貨架(Wurman et al., 2008)

透過此種系統，能大大降低人為出錯的比率提升揀貨效率(Li & Liu, 2016)，許多大型電商平台如美國亞馬遜、中國的阿里巴巴及京東公司等皆開始使用這樣類型的倉儲配置。在 2018 的新聞報導中，亞馬遜在導入心型機器人前，單個物流中心最高峰一天能出貨 70 萬個品項，而導入該系統後則提升出貨量到 150 萬個品項。也由於員工不需要再進入貨架區揀貨，僅須待在定點的揀貨工作站負責後續包裝的動作，使得效率提升三倍，出貨準確率甚至達到 99.99%。在這一新興型態的倉儲內，同樣也有許多不同的子問題，例如：儲貨時如何將產品分配於合適的貨架上？揀貨時如何將訂單分組？如何選擇合適的貨架與機器人來完成訂單？移動貨架時如何處理多台機器人與貨架的路徑規劃以避免碰撞或塞車？過往的研究中通常將上述子問題分別獨立處理，但其兩兩子問題之間確實有著相當程度的關聯，例如商品的擺放會影響訂單的分組，因此 Xiang, Liu, & Miao (2018)探討如何整合「產品分配至貨架」以及「訂單分組」兩子問題的規劃，但針對其它子問題的可能組合，至今仍無整合性的規劃研究。

1.1研究動機

在本研究牽涉之數個子問題中，屬較後階段的「多機器人路徑規劃問題」(Multi-Agent Path Finding, MAPF)有較多文獻探討，但大多數文獻皆設各貨架已被配置一台專屬的機器人，且所有的機器人(即貨架)皆已被給定其起訖點，所求的目標為找到一組避免碰撞之路徑，且希望該路徑之總移動距離或最遲完工時刻(makespan)能被最小化。套用至先前提到的可動式貨架倉儲的話，其問題情境如下：假設目前所要用於完成各訂單的貨架都已被選定，且機器人皆已就定位於其所要搬運的貨架下方。在此一設定下，未必其所選定的貨架可被證明是最好的組合，這是由於目前對於貨架選擇的衡量指標多使用最小化第一階段的「貨架交換次數」(Boysen, Briskorn, & Emde, 2017)，亦即如何使用最少的貨架完成所有訂單。但現實中若某一訂單僅需要某一貨架即可完成，而該貨架位於相對於該揀貨工作站最遠的角落的話，那原始的貨架選法可能有不好的表現。以圖 1.3 為例，假設在星號處需要 a、b 兩種產品來完成一筆訂單，此時若依據最小化貨架交換次數的原則，在進行 MAPF 的規劃時，我們將會選擇 r_1 來完成訂單，雖然貨架交換次數為 1 次，但最終得到的結果會需要移動 23 步；若是能夠將貨架的選擇與路徑規劃同時作考量而選擇 r_2 與 r_3 ，即使貨架交換次數增加為 1 次，但可將總移動距離縮小至 5 步。由上述範例可得知，貨架的選擇與 MAPF 問題兩者之間具有相當程度的關聯，在本研究中我們將兩個問題同時處理，預期會較傳統文獻單一分別處理方式有更好的效果。

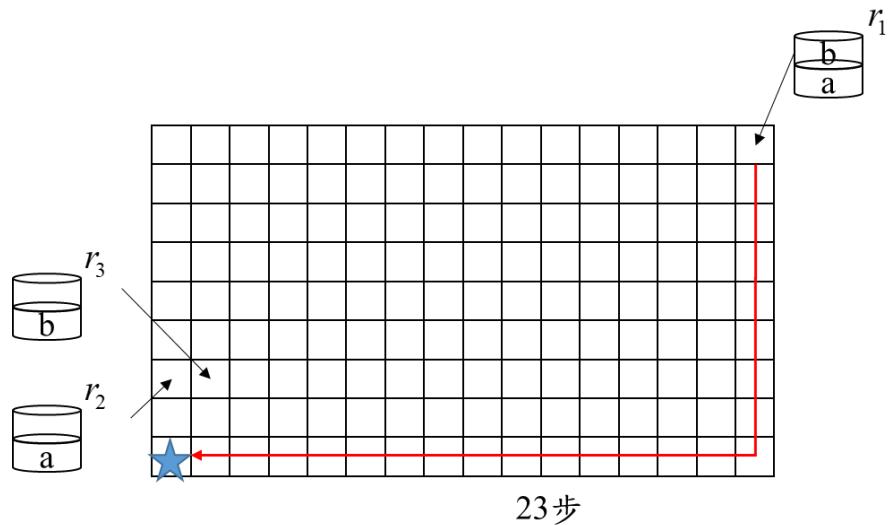


圖 1.3 貨架選擇示意圖

1.2研究目的

透過圖 1.3 的例子，我們可以瞭解到貨架選擇問題事實上與後續的路徑規劃問題是息息相關的，就算在貨架選擇時加入預估的來訪路徑時間(Li, Zhang, Zhang, & Hua, 2017)，並不代表其能在該預期時間內移動至目的地，這是因為上述方式並未考慮到後續路徑規劃可能會造成的碰撞及塞車狀況，因此實際抵達的時刻極可能將更為延長，所以要如何能夠同時考慮貨架選擇與貨架路線規劃等兩個子問題極為重要。在過去的研究中不僅僅是未將貨架的選擇與路徑規劃同時考慮，也在使用 MAPF 來進行此類倉儲問題的求解時進行相當程度的簡化(例如缺少交換貨架的機制)，因此有些學者放寬 MAPF 的限制，使兩機器人在相鄰兩點間能進行包裹(貨架)的交換，此問題被稱為「包裹可交換式機器人路徑規劃問題」(Package-Exchange Robot Routing Problem, PERR)，在 PERR 的假設之下，機器人不再是被綁定於特定包裹(貨架)上，而可被允許做包裹(貨架)的交換，但是此機制也僅限於兩相鄰的機器人而已。而在現實應用中，貨架的個數應該會遠大於機器人個數的，而且貨架其實是被允許暫時放下、於通道中等待其它機器人來接力的，但過去文獻皆簡化或忽略這部分的設定，因此我們希望能更放寬問題假設以

處理一個機器人數目可小於貨架數目，且允許機器人交換貨架的更為一般性的問題。另外，在 MAPF 問題的假設下，大多相關文獻針對完成任務後的機器人該如何移動皆無著墨，而是將終點視為類似黑洞的節點（亦即機器人抵達終點後，將宛如消失一般），但實際上，機器人與貨架在抵達終點後該如何歸位應該也要一併考慮，因為其位置與移動路徑也會影響到後續來訪的貨架之路徑。

在所有貨架與機器人位置、產品之於貨架的儲存位置、各個揀貨工作站訂單皆為已知的情況下，本研究希望能提出嚴謹的數學模式以解決所需最小的機器人與貨架移動距離、或是最小化最遲完工時刻，並同時規劃貨架與機器人的歸位路徑。因數學模式能求解的規模有限，且完成訂單的路徑規劃問題具有即時性，又因 MAPF 與其變形 PERR 本身即是 NP-hard 問題(Ma, Tovey, Sharon, Kumar, & Koenig, 2016; Yu & LaValle, 2013b)，因此考量更多決策的本研究亦為 NP-hard 問題，本研究亦希望提出有效率且兼顧求解品質的演算法，以達到實際應用的需求。

1.3論文架構

本論文之架構如下：第二章回顧貨架選擇、MAPF 及其變形問題以及倉儲設定類似之文獻；第三章討論結合貨架選擇之多機器人路徑規劃問題，並在此章提出新的數學模式；第四章提出兩種演算法以加快求解速度並設計訂單重排的機制作為輔助，保持一定的求解品質；第五章比較數學模式與演算法之包括求解規模、速度以及品質等；第六章則是結論與未來研究方向的建議。

第二章 文獻探討

本研究為過去貨架選擇與 MAPF 等兩種問題之混合問題，因此本章先探討貨架選擇問題相關文獻，接著探討 MAPF 問題相關文獻與其變形 PERR 相關文獻，再深入介紹常用於求解該兩類問題的數學模式與演算法，最後回顧一些相似問題情境設定之文獻，並與本研究之問題進行比較。

2.1 貨架選擇問題

假設有一可動式貨架倉儲，所有的貨架上裝有的產品皆為已知，單一貨架上可能儲存多種類的產品，同一種類的產品也可能儲存於多個不同的貨架上，所有訂單的需求皆為已知的情況下，如何選擇最適當的貨架順序來完成訂單是此問題的核心，而用來衡量貨架順序好壞的指標也根據不同的研究而有所不同。

Boysen et al. (2017) 針對單一揀貨工作站的訂單進行規劃，給定該工作站的訂單內容，以及可同時處理的訂單上限，訂單必須依照順序完成但可自行規劃訂單順序，例如：該工作站有 4 筆訂單為 o_1, o_2, o_3, o_4 且可同時處理 2 筆訂單，若決定處理順序為 o_1, o_3, o_4, o_2 則該工作站必須先完成 o_1 或 o_3 後，才處理 o_4 。該研究將貨架交換次數訂為其最佳化的目標，並加入時間的概念，若前一時刻 $t-1$ 與現在時刻 t 來訪的貨架為不同貨架，則定義為一次貨架交換。他們提出一個數學模式外，也將問題更拆解為兩個子問題，其一為：若已給定來訪貨架的順序，該如何決定訂單的順序？其二為相反的問題：若給定訂單的順序，該如何決定貨架的來訪順序？該研究針對此兩子問題，分別提出不同的啟發式演算法求解。

Li et al. (2017) 的研究中同樣針對單一揀貨工作站的訂單進行規劃，並希望利用總移動距離來當作衡量貨架選擇的好壞，因此他們定義一個繞行時間矩陣，繞行時間之定義如下：若選擇該貨架，該貨架從其初始位置移動至揀貨工作站再返回其初始位置之時間。此繞行時間在求解時皆為定值，因此並無考量現實中真實

位置所可能造成的倆倆交互影響，例如選擇兩貨架可能造成實際塞車等問題，在該研究就會被忽略。此外，其研究也放寬對於訂單的假設，並無要求先完成先來的訂單。他們除提出一個數學模式外，亦使用三階段的混合啟發式演算法求解：第一階段先使用啟發式演算法找到一組可行解，使得所有的訂單需求品項可被此組解所選擇的貨架滿足；第二階段刪除在此組解內多餘的貨架；第三階段再採用交換的策略，不斷將未選擇的貨架加入可行解、刪除多餘的貨架，並保留較好的結果。他們的研究中也提到，事實上貨架的選擇問題本質上是集合覆蓋問題(Set-Covering Problem)的延伸，可以將訂單視為需要被覆蓋的大集合，而各個裝有不同商品的貨架則可以被視為用來覆蓋的小集合，因此他們也將所提出的三階段混合啟發式演算法與先前常用於求解集合覆蓋問題的貪婪啟發式演算法(Chvatal, 1979; Sundar & Singh, 2012)進行比較，並指出他們的混合啟發式演算法相較於貪婪演算法的近似解，能得到全域的最佳解。並在最後提到針對訂單的完成，包括訂單分組(order-batching)、機器人指派(robot task assignment)、以及路徑規劃(path routing)的整合規劃等，皆為未來應再探討的議題。

本研究與文獻中貨架選擇問題研究最大相異處，在於我們考量了貨架完成任務後，其後續的歸位路徑規劃問題，因此文獻中所使用的某些衡量指標對本研究的設定將變得不甚合適；此外，有別於過往的研究皆僅考慮單一揀貨工作站的規劃，本研究欲同時規劃多個揀貨工作站。

表 2.1 貨架交換相關文獻比較

作者	貨架交換 次數	貨架移動 距離	實際規劃 可行性	多個揀貨 工作站
Boysen et al. (2017)	V			
Li et al. (2017)		V		
本研究		V	V	V

2.2 多機器人路徑規劃及其相關的變形問題

MAPF 在相當多的領域如人工智慧(artificial intelligence)、機器人學

(robotics)、理論計算機科學 (theoretical computer science) 以及作業研究(operation research) 中皆曾被廣泛地研究(Ma et al., 2017)，其主要問題為：給定一個連結的無向簡單網路圖(connected undirected simple graph) $G = (V, E)$ ，其中 $V = \{v_i\}$ 為此網路圖之節點集合， $E = \{(v_i, v_j)\}$ 為網路圖之節線集合；給定一群有任務的機器人集合，每個機器人皆有自己的起訖點；如何在網路圖上找到一組可最佳化衡量指標的路徑集合，而每個機器人在完成其任務的移動過程中，不得與其它機器人發生節點及節線上的碰撞。MAPF 問題通常使用以下三種目標衡量指標(Yu & LaValle, 2013b)：個別任務完成時間之加總、最遲完工時刻、或總移動距離。

2.2.1 多機器人路徑規劃問題(MAPF)

Yu & LaValle (2013a)的研究中，提出能夠描述 MAPF 問題的數學模式，透過將原始的網路圖擴展為時空網路圖(time-expanded network)，如圖 2.1 所示，如此便有利於描述有無碰撞的情形發生。在他們的研究中也提到，這個問題本質上是一個多元商品流量問題(Multi-commodity flow)，運用類似概念來建立數學模式，將時空網路圖上的節線與節點之流量皆限制為 1 單位，即能避免碰撞發生於節線或節點。由於此研究使用數學模式來進行求解，後來相關的 MAPF 演算法研究也多與此篇結果進行比較。

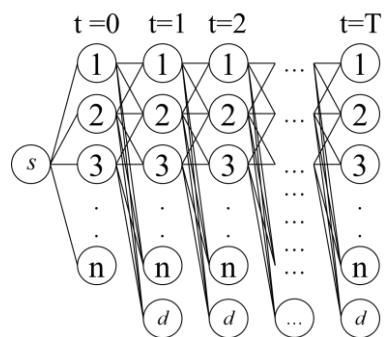


圖 2.1 時空網路圖

Erdem, Kisa, Oztok, & Schüller (2013)的研究中，提出了一個求解 MAPF 的框

架，利用問答集程式設計(Answer Set Programming, ASP)的方式，將一個問題化為一個程式，並認為使用 ASP 能很容易地表達許多路徑規劃的問題。

Surynek (2015)的研究中希望能夠用放寬對於最遲完工時刻的最佳化需求來化簡時空網路圖，其化簡之直覺在於他們觀察到若是在某一點不太可能重複經過，則那個點就不需要在時空網路圖上於不同時刻再擴展，並主要使用基於命題可滿足性(propositional satisfiability, SAT)的求解方式，改變最遲完工時刻的上界，來檢查是否有存在滿足 MAPF 的解。

Sharon, Stern, Felner, & Sturtevant (2015)的研究中，提出了使用基於衝突搜尋(Conflict-Based Search, CBS)演算法應用於 MAPF 的求解，其架構分為高層級搜尋(high-level search)以及低層級搜尋(low-level search)。其中，高層級搜尋是在一個衝突樹(Conflict Tree, CT)下進行產生限制的動作，不同的限制組合皆會在 CT 上新增一個節點，在 CT 下的每個節點包含了對於某特定機器人在時間及地點的限制，例如機器人 k_1 不得在時刻 t 出現於節點 n ，且機器人 k_2 不得在時刻 $t-1$ 出現於節點 n 等。因此，低層級搜尋即是針對某個在 CT 節點上的限制來針對每個機器人產生其最短路徑，若是低層級搜尋進行完畢後的結果還是有碰撞情形發生，則高層級搜尋便會在 CT 上產生新的節點。以圖 2.2 為例，若有兩機器人其任務起訖組合分別為(1,9)與(5,3)，高層級搜尋首先建立一個無限制的根節點，低層級搜尋依據此限制分別對機器人 1 與機器人 2 進行最短路徑搜尋，得到結果如圖 2.2 中的根節點所示，所花費的總移動步數為 6。之後在進行碰撞檢查時，發現在兩機器人的第一步即在節點 2 發生碰撞，因此高階層搜尋針對 $t=1$ 以及節點 2 產生限制，生成限制機器人 1 的左節點以及限制機器人 2 的右節點；因左節點之低層級搜尋結束後即符合無碰撞原則，右節點雖然尚有碰撞發生，但 CBS 是依照最佳優先搜尋(best first search)，而右節點在有碰撞發生的情況下亦與左點之成本相同，再分支下去成本必定只可能增加，因此可以結束搜尋。在他們的研究中，亦證明 CBS 的結果必為最佳解，但也解釋由於一旦有碰撞的發生即要分支

新增節點，會造成運算時間的大量增加。

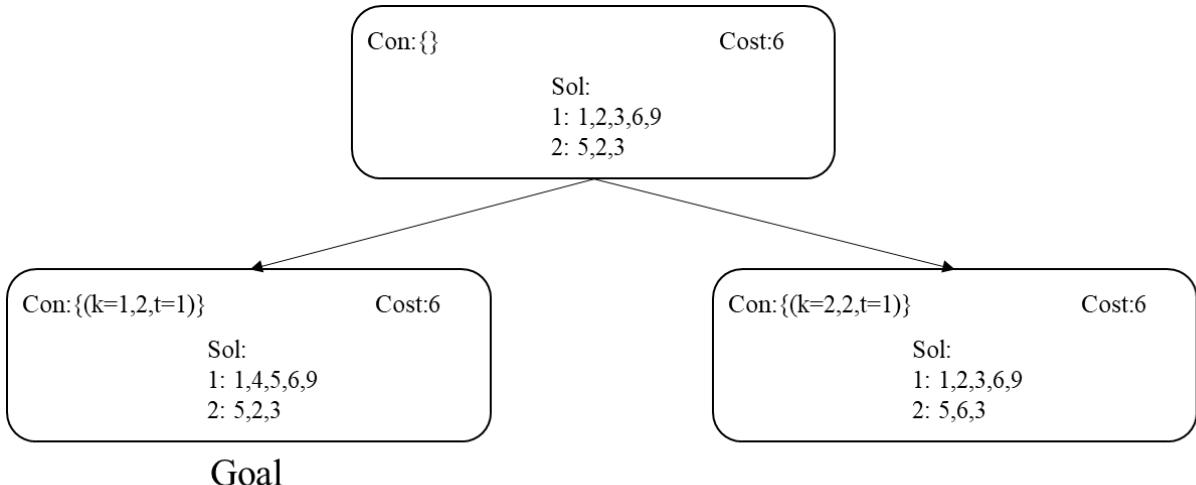


圖 2.2 CT 限制生成

Hönig, Kiesel, Tinka, Durham, & Ayanian (2018)將 MAPF 問題加入任務指派 (Task Assignment, TA) 的元素，在他們的問題假設下，機器人的起始位置雖然一樣皆為已知，但每個機器人有一或多個可能的潛在的訖點，只要能夠抵達任意一個潛在訖點即算完成任務。他們改良 CBS 演算法為 CBS-TA 演算法，修改高層級的搜尋機制，將原本產生的 CT 改為衝突森林(Conflict Forest)，但只在有需求時才新增根節點。新增根節點的依據如下：先建立一個指派搜尋樹(Assignment search tree)，如圖 2.3(c)所示，圖中每節點左邊數字代表機器人編號，右邊則代表訖點。若有節線連接，代表該訖點為該機器人的潛在訖點，節線上數字則代表其不考慮碰撞的最短路徑距離；指派搜尋樹的子節點則是刪除一個親代節點 (parent node) 的潛在路徑組合而成，接著依照指派搜尋樹建立限制森林，如圖 2.3(d)所示，森林內各節點的低層級搜尋則依照 CBS 演算法進行低層級搜尋。

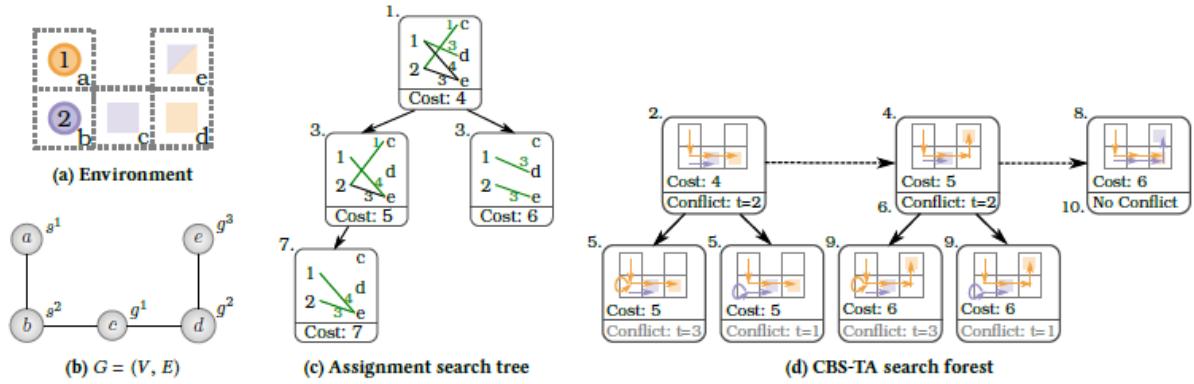


圖 2.3 CBS-TA 架構(Hönig et al., 2018)

Ma & Koenig (2016)的研究中，提出基於衝突的最小成本流(Conflict-Based Min-Cost-Flow, CBM)演算法用於求解 MAPF 加上 TA 的問題。該研究與 CBS 不同處如下：在低層級的搜尋中，他們對時空網路圖運用最小成本最大流(min-cost max-flow)演算法來尋找最佳的路徑，並透過在有碰撞時調整節線的經過成本，來使得能夠避免碰撞也提出能夠得到局部最佳解(suboptimal)的增強式基於衝突搜尋(Enhanced CBS, ECBS)演算法。

表 2.2 MAPF 相關文獻比較

作者	結合 TA	貨架與 機器人	數學 模式	演算 法
Yu & LaValle (2013a)			V	
Erdem et al. (2013)				V
Surynek (2015)				V
Sharon et al. (2015)				V
Hönig et al. (2018)	V			V
Ma & Koenig (2016)	V			V
本研究		V	V	V

回顧上述文獻後可以發現在 MAPF 的研究上，有許多不同的學者提出不同的方法來解決，包括能得到最佳解的方法以及近似解的方法，但目前對於 MAPF 大規模的求解，尚未有非常有效率的方法(Ma & Koenig, 2017)。本研究與 MAPF

的不同處，在於我們將貨架與機器人視為獨立的兩種不同個體，因此包含如何分配機器人來移動至哪個貨架？或是在中途是否需要由不同的機器人接力完成？等問題，都是本研究與相關文獻不同之處。

2.2.2包裹可交換式機器人路徑規劃(PERR)

在 MAPF 中，所有的機器人起始位置皆已被給定，且貨架概念上是與機器人綁定的，亦即可將貨架假想成其能夠自己移動，因為這樣的假設下，機器人之間無法進行任務的交換。然而這代表某貨架從頭到尾皆必須由相同的機器人來搬運，此簡化假設太不符合貨架個數遠多於機器人個數的現實設定，因此本小節將回顧另一個 MAPF 的延伸：可交換機器人搬運的 PERR 之相關文獻。在 PERR 問題中，兩兩相鄰的機器人可以交換他們的包裹，交換時僅包裹的移動，機器人並不移動，此設定將包裹與機器人視為不同的獨立個體，因此比 MAPF 更貼近現實；但該設定下的包裹之所以能在兩相鄰機器人間移動，其實靠得是機器人配有特殊的機構。若以本研究考慮的機器人而言，並沒有類似的特殊機構可在機器人不動的情況下直接交換包裹；在本研究的設定下，反而是包裹不動、但靠機器人相互之間的位移接力移動包裹，因此本研究無法使用 PERR 的設定。

Ma et al. (2016)的研究中探討 K 種包裹的 PERR 問題(K-PERR)，將其轉化為一個整數的多元商品網路流量 (integer multi-commodity network-flow) 問題，並透過基於 Yu & LaValle (2013a) 提出的模型來求解。Drain (2018)的研究中則提出兩種新的演算法用於解決 PERR 問題。

PERR 問題目前算是相當新穎的一個問題，因此可查到的文獻並不多。另外儘管 PERR 問題中允許交換，但其機制較不適用於可動式貨架倉儲，因實際上有別於包裹(貨架)的交換，應該是機器人的交換較符合可動式貨架倉儲的架構。

2.3 移動式倉儲其它相關文獻

由於本研究結合了包含貨架選擇以及路徑規劃兩階段的整合，在規劃的過程中亦產生其它相關衍伸問題，例如貨架的暫存位置等，因此本節將回顧同樣為移動式貨架倉儲設定下的相關研究所採用的一些策略。

Weidinger, Boysen, & Briskorn (2018) 討論有關於貨架的儲存位置問題，其中也比較了傳統固定式貨架倉儲的儲位分配策略在移動式貨架倉儲的表現，該研究進行了以下幾種策略的比較：

1. 隨機儲位(Random storage)

對於每個完成任務的貨架，隨機指派一個空位。

2. 最近儲位(Closest open location storage)

將完成任務的貨架指派至距離該揀貨工作站最近的空位。

3. 專用儲位(Dedicated storage)

每個貨架皆有其固定的位置，當完成任務時必須從揀貨工作站返回原儲位。

4. 全存取(Full-turnover storage)

原本指將使用率愈高的產品，放置離工作站愈近的儲位，但該研究指出，因為移動式貨架上的商品可能隨時間而變動，此種儲位分配策略並不適合轉換為移動式貨架的儲位分配。

5. 分級儲位(Class-based storage)

原本指將儲位分為 ABC 三種等級，並將商品依照使用到的頻率分配至不同等級的儲位，但因為移動式貨架上的商品可能隨時間而變動，與全存取一樣，該研究也指出此種儲位分配策略並不適合表達移動式貨架的儲位分配。該研究最後認為將完成任務的貨架指派至最近的空閒儲位的「最近儲位」策略，意外地是表現相當好的策略，也建議實際應用採取「最近儲位」策略。

Xiang et al. (2018) 探討如何將產品分配到不同的貨架上，以及如何將訂單分

組，目的是使得完成訂單所需要的貨架數最少。在他們的研究中，盡量將常一起出現於同訂單的產品組合放置於同一貨架上，並將需求類似的訂單分在同一組，是以另一種角度來規劃貨架的選擇問題。

2.4小結

本章回顧了包含貨架選擇問題、MAPF 與其相關變形問題，以及其它提供策略面參考的類似情境設定文獻，但以上文獻通常過於簡化實際問題，因此我們認為更為一般性的研究有其必要。且從現實應用的層面，我們認為整合貨架選擇與路徑之整合規劃極為必要，然而光是移動式貨架倉儲同一種類問題的研究已相當稀少，更遑論整合式問題的規劃，這突顯了本研究的創新性，但也代表本研究將面臨的挑戰。下一章本研究將嘗試針對貨架選擇與多機器人路徑規劃的整合問題，提出其對應的數學模式。

第三章 多台移動式倉儲機器人最佳揀貨作業路徑規劃模式

本章將針對本研究問題進行定義及假設，為了方便數學模式的建立，我們將原始的網路圖轉變以層空網路圖的結構來提出整數規劃模式，並同時考量機器人以及貨架個別的路徑規劃，最後以簡易數值測試闡述說明數學模式的運作結果。

3.1 問題描述

假設有一個無向網路圖 $G = (N, A)$ ， N 為網路圖中所有節點集合， A 為網路圖中所有節線的集合，在網路圖上存在 k 台機器人與 r 個貨架裝載不同種產品，並有 o 筆訂單需要被完成，每筆訂單皆已被指派必須由哪個揀貨工作站完成。有別於以往 MAPF 問題，在本研究中貨架本身並不會自己移動，需要由機器人來進行搬運，且在搬運的過程中允許機器人的交換，也就是一個貨架的搬運可能是藉由接力的形式來完成，與 PERR 問題不同的是我們的交換機制較符合現實，在 PERR 中是上面包裹的交換(圖 3.1)，本研究中則是機器人的交換(圖 3.2)。

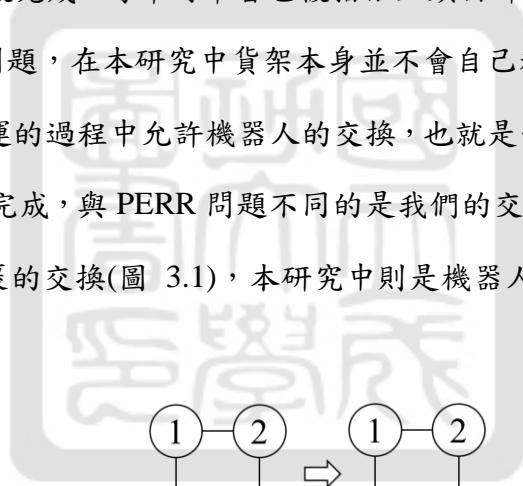


圖 3.1 PERR 交換機制

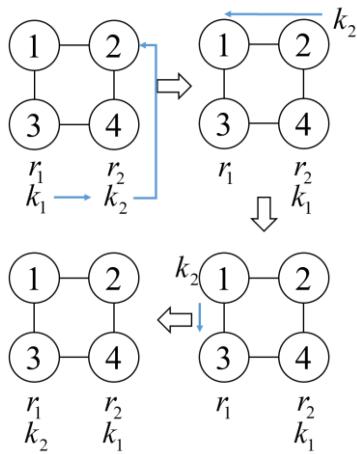


圖 3.2 本研究交換機制

本研究另一個與 MAPF 問題不同之處，在於本研究的貨架允許重複使用。

當完成某一揀貨工作站的訂單時，若有需要，可再移往其它揀貨工作站被使用。因為產品可能儲存於多個貨架上，要如何選擇適當的貨架並安排機器人以前來運送是本研究主要的問題

3.2 問題假設

為了適當簡化問題，本研究有以下基本假設：

1. 假設機器人自底部將貨架頂起之動作不耗時，機器人轉向之動作亦不耗時。
2. 當所需產品之貨架被機器人運抵工作站即完成訂單，忽略訂單處理時間。
3. 相同揀貨工作站之訂單可同時處理，無先後之順序關係。
4. 每筆訂單上皆只需要一種產品，只考慮種類需求，但忽略其數量。
5. 貨架上有裝載的產品皆假設為量永遠足夠，亦即只考慮種類，但忽略數量。
6. 貨架的任務完成後，須待在指定的儲存區域，如圖 3.3 陰影區所示

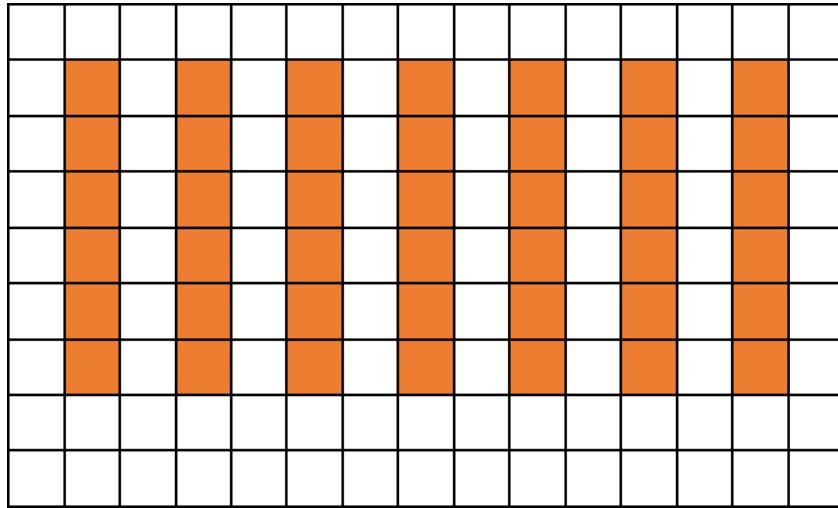


圖 3.3 儲存區域示意圖

3.3 網路架構

在可動式貨架倉儲中，機器人會依據地上的條碼進行移動，因此每個條碼可被視為一個節點，每一個節點可沿四個方向進行移動，由於對角移動(圖 3.4)會導致每個移動動作耗時不一致，且對於碰撞發生的認定亦將變得更為複雜，因此在本研究中並不建立對角節線（亦即，只考慮縱橫雙向的移動）。

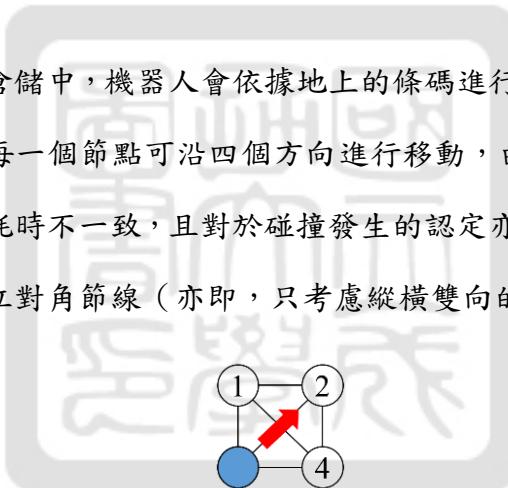


圖 3.4 對角移動

本研究之倉儲佈置為網格網路圖(grid network)，為了使數學模式易於表達碰撞的情形，參考 Yu & LaValle (2013a) 之作法，將時間的概念加入網路圖裡。但若使用時空網路圖，則可能允許跨時空的節線存在(如圖 3.5 中橫跨兩單位時間長度以上的節線)，因此停留在該節點的機器人或貨架，可能可以經由連到不同時間點的同一空間節點來表示，如此假設將會增加網路圖的複雜度且難以檢驗是否發生碰撞，因此本研究使用一次規劃一步的層空網路圖(level space network)，如圖 3.6 所示，每一層可視為一個移動或停留的動作，由於我們也假設每一個移

動或停留的動作耗時相同，因此本「層空網路」實質上亦與「時空網路」相同，只不過前者的所有節線一律耗時一單位時間。如此便有利於描述停留原位可能遭遇碰撞的情況。此外，本層空網路圖只有設立虛擬起點 s ，並無設立虛擬訖點，這是因為我們的貨架皆是可被重複利用的，若將其連入虛擬訖點，則會造成其實際上存在，在網路圖上卻消失的情形發生，如此一來可能使產生的解實際上會與已連入虛擬訖點的機器人或貨架發生碰撞。

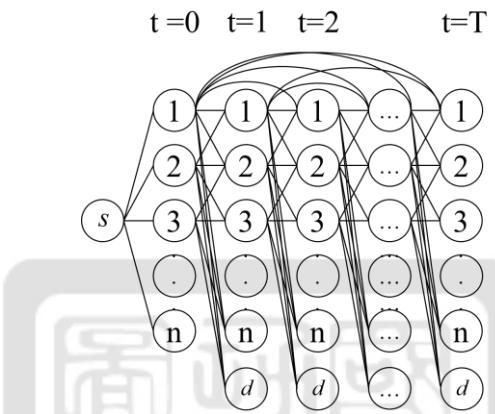


圖 3.5 時空網路圖跨時空節線—以節點 1 為例

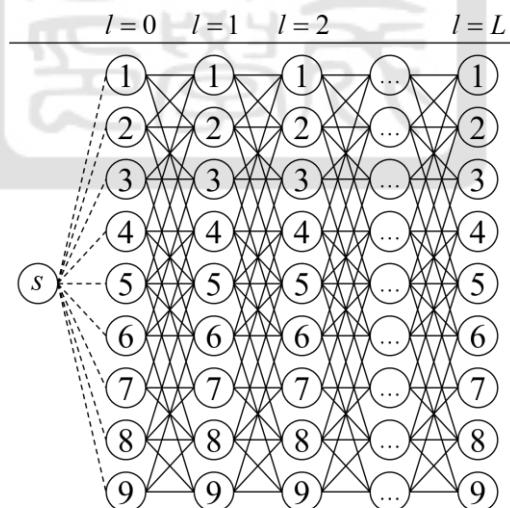


圖 3.6 層空網路圖—以九宮格為例

3.4 數學模式

由於本研究將機器人與貨架視為不同之獨立物件，因此在建構數學模式時，不僅僅考量機器人之路徑規劃，同時貨架的路徑亦是重點，當貨架移動時必須依附於某一個機器人的路徑。3.4.1 小節將介紹此次數學模式所用到之符號定義，3.4.2 小節說明目標式設定，最後 3.4.3 小節說明限制式。

3.4.1 符號定義

集合

N	所有節點之集合
N_s	儲存區域節點之集合， $N_s \subseteq N$
N_p	揀貨工作站節點集合， $N_p \subseteq N$
A	所有節線之集合
K	機器人之集合
O	訂單之集合
R	貨架之集合
R_o	可完成訂單 o 之貨架集合， $o \in O$

參數

L	最大層數
M	極大的數
s_k^{Robot}	機器人 $k \in K$ 之起始位置， $s_k^{Robot} \in N$
s_r^{Rack}	貨架 $r \in R$ 之起始位置， $s_r^{Rack} \in R$

變數

x_{ij}^{kl} 機器人 k 在第 l 層經過節線 $(i, j) \in A$ 為 1；反之為 0 y_{ij}^{rl} 貨架 r 在第 l 層經過節線 $(i, j) \in A$ 為 1；反之為 0 z_o^l 訂單 $o \in O$ 在第 l 層結尾被完成為 1；反之為 0 w

最遲完工時刻

節線集合 A 亦包含了某節點連向自己的節線 $(i, i) \in A, \forall i \in N$ 。由於簡化訂單為每筆訂單僅需一種產品，因此在前處理階段可預先將貨架分類成各個訂單 o 可用之貨架集合 R_o

3.4.2 目標式

此類問題最常最小化的兩種指標，分別為「最遲完工時刻」，如式(3.4.1)，以及「總等待時間」，如式(3.4.2)。

$$\text{Minimize } w \quad (3.4.1)$$

$$\text{Minimize } \sum_{l=0}^L \sum_{o \in O} l \times z_o^l \quad (3.4.2)$$

由於移動距離的因素可能相對於時間較不重要，但在不影響時間的前提下，考量實際運作時，各機器人皆有其電量限制，理論上並不希望做多餘的移動，因此我們也考慮給予移動一個很小的懲罰值 ε 來避免無謂的移動，因此可將式(3.4.1)與式(3.4.2)修改成式(3.4.3)與式(3.4.4)。

$$\text{Minimize } w + \varepsilon (\sum_{(i,j) \in A, i \neq j} \sum_{k \in K} \sum_{l=0}^L x_{ij}^{kl} + \sum_{(i,j) \in A, i \neq j} \sum_{r \in R} \sum_{l=0}^L y_{ij}^{rl}) \quad (3.4.3)$$

$$\text{Minimize } \sum_{l=0}^L \sum_{o \in O} (l \times z_o^l)$$

$$+ \varepsilon (\sum_{(i,j) \in A, i \neq j} \sum_{k \in K} \sum_{l=0}^L x_{ij}^{kl} + \sum_{(i,j) \in A, i \neq j} \sum_{r \in R} \sum_{l=0}^L y_{ij}^{rl}) \quad (3.4.4)$$

3.4.3限制式

本研究同時考量貨架與機器人之路徑規劃，因此在許多限制式上皆具有對稱性。亦即同樣一組情況的限制，將分別有一組限制式用來限制機器人，而另一組類似的限制式則用來限制貨架。

限制式(3.4.5)與限制式(3.4.6)分別為機器人與貨架之流量守恆限制式。

$$\sum_{i:(i,j) \in A} x_{ij}^{kl} = \sum_{i:(j,i) \in A} x_{ji}^{kl+1} \quad \forall j \in N, k \in K, 0 \leq l < L \quad (3.4.5)$$

$$\sum_{i:(i,j) \in A} y_{ij}^{rl} = \sum_{i:(j,i) \in A} y_{ji}^{rl+1} \quad \forall j \in N, r \in R, 0 \leq l < L \quad (3.4.6)$$

限制式(3.4.7)與限制式(3.4.8)為機器人與貨架之初始位置。

$$x_{0s_k^{Robot}}^{k0} = 1 \quad \forall k \in K \quad (3.4.7)$$

$$y_{0s_r^{Rack}}^{r0} = 1 \quad \forall r \in R \quad (3.4.8)$$

限制式(3.4.9)與限制式(3.4.10)為節點之容量限制，其目的在於避免兩機器人或兩貨架在同一階層移動至相同節點。

$$\sum_{i:(i,j) \in A} \sum_{k \in K} x_{ij}^{kl} \leq 1 \quad \forall j \in N, 0 \leq l \leq L \quad (3.4.9)$$

$$\sum_{i:(i,j) \in A} \sum_{r \in R} y_{ij}^{rl} \leq 1 \quad \forall j \in N, 0 \leq l \leq L \quad (3.4.10)$$

限制式(3.4.11)與限制式(3.4.12)則為節線之容量限制，其目的在於避免兩機器人或貨架在同一階層於同一節線上同向或反向移動。

$$\sum_{k \in K} x_{ij}^{kl} + \sum_{k \in K} x_{ji}^{kl} \leq 1 \quad \forall (i,j), (j,i) \in A, i \neq j, 0 \leq l \leq L \quad (3.4.11)$$

$$\sum_{r \in R} y_{ij}^{rl} + \sum_{r \in R} y_{ji}^{rl} \leq 1 \quad \forall (i,j), (j,i) \in A, i \neq j, 0 \leq l \leq L \quad (3.4.12)$$

限制式(3.4.13)與限制式(3.4.14)限制在任一個階層上，任一機器人或貨架只能且必須存在於唯一的某一個節線上。

$$\sum_{(i,j) \in A} x_{ij}^{kl} = 1 \quad \forall k \in K, 0 \leq l \leq L \quad (3.4.13)$$

$$\sum_{(i,j) \in A} y_{ij}^{rl} = 1 \quad \forall r \in R, 0 \leq l \leq L \quad (3.4.14)$$

限制式(3.4.15)定義了貨架的移動必須有某一個機器人沿著同方向同時移動
(即以該機器人頂住該貨架而一起移動)，否則貨架必定只能留在原位。

$$y_{ij}^{rl} \leq \sum_{k \in K} x_{ij}^{kl} \quad \forall (i,j) \in A, i \neq j, r \in R, 0 \leq l \leq L \quad (3.4.15)$$

限制式(3.4.16)則保證每筆訂單，都必須由某一個有裝載該訂單所需產品的
貨架走訪至負責該訂單的揀貨工作站。

$$\sum_{i:(i,d_o) \in A} \sum_{r \in R_o} \sum_{l=0}^L y_{id_o}^{rl} \geq 1 \quad \forall o \in O \quad (3.4.16)$$

限制式(3.4.17)至限制式(3.4.19)與目標式夾擠，得到完成訂單之時刻與計算出
最遲完工時刻。

$$\sum_{i:(i,d_o) \in A} \sum_{r \in R_o} y_{id_o}^{rl} \geq z_o^l \quad \forall o \in O, 0 \leq l \leq L \quad (3.4.17)$$

$$\sum_{i:(i,d_o) \in A} \sum_{r \in R_o} y_{id_o}^{rl} \leq \sum_{l'=0}^L z_o^{l'} \quad \forall o \in O, 0 \leq l \leq L \quad (3.4.18)$$

$$w \geq \sum_{l=0}^L (l \times z_o^l) \quad \forall o \in O \quad (3.4.19)$$

限制式(3.4.20)用來使貨架最終都要回到指定儲存區域

$$\sum_{i:(i,j) \in A} \sum_{j \in N_s} y_{ij}^{rL} = 1 \quad \forall r \in R \quad (3.4.20)$$

限制式(3.4.21)至限制式(3.4.23)則限制了變數的範圍。

$$x_{ij}^{kl} \in \{0,1\} \quad \forall (i,j) \in A, k \in K, 0 \leq l \leq L \quad (3.4.21)$$

$$y_{ij}^{rl} \in \{0,1\} \quad \forall (i,j) \in A, r \in R, 0 \leq l \leq L \quad (3.4.22)$$

$$z_o^l \in \{0,1\} \quad \forall o \in O, 0 \leq l \leq L \quad (3.4.23)$$

3.5小結

本章嘗試用層空網路圖的概念，針對整合貨架選擇以及機器人與貨架路徑規
劃，提出一個新的數學模式，但若是要應用於實際例子，勢必需要設計出有效率

的演算法，以更快得到近似最佳或品質尚可接受的解。



第四章 多台移動式倉儲機器人最佳揀貨作業路徑規劃演算法

第三章所提出之數學模式，經過小規模測試後發現求解速度緩慢，若是要在實務上使用較不可行，因此我們希望設計演算法讓我們能在愈短的時間內得到愈好的可行解，本章將會介紹兩種演算法與搭配的訂單重排方法以加速求解。

4.1 滾動式求解

由於一次求解數學模型耗費的時間較長，本研究思考是否能將訂單進行分組，一次求解部分訂單，使單次求解的規模變小，進而達到加速的效果，因此我們設計了滾動式求解的流程如表 4.1，其中 O_{now}, O_{done} 分別為此次要求解之訂單集合以及已固定之訂單集合。滾動式解法首先要決定一次求解的量 $Batch_size$ 以及每次要固定的訂單數 Fix_size ，接著將第一批的訂單加入 O_{now} 用第三章提出的數學模式求解，每次求解完一批，再將要固定的訂單加入至 O_{done} ，並自 O_{now} 移除，同時將這些訂單所使用到的貨架路徑丟入限制式池 CP 成為之後批次的限制式，重複動作直到 $O_{done} = O$ ，也就是所有的訂單都被完成為止。

表 4.1 滾動式求解流程

Rolling Model

Data: $O, Batch_size, Fix_size$

- 1: $O_{now}, O_{done}, CP = \text{null}$
 2. **While** ($O_{done} \neq O$) **do**
 3. fill O_{now} up to $Batch_size$
 4. do Model(O_{now}, CP)
 5. **For** o in top Fix_size finished order in O_{now} **do**
 6. $O_{done}.add(o)$
 7. $O_{now}.delete(o)$
 8. add used rack r 's path to CP
 9. **End for**
 10. **End while**
-

以圖 4.1 為例，有五筆訂單 O_1, \dots, O_5 需要被處理，假設一次處理的訂單數為 3，每次固定兩筆訂單用到的貨架首次求解可以得到貨架與機器人的路徑如圖 4.2 所示，從結果可以發現 O_1 到 O_3 中是 O_1 與 O_2 先分別被貨架 r_1 與 r_4 完成，因此這兩個貨架從儲存區域移動至揀貨工作站的路徑將被加入下回合求解時貨架的限制式(4.2.1)，接著將 O_1 與 O_2 移除，加入 O_4 與 O_5 再次求解，得到新的路徑如圖 4.3，因已求解完所有訂單而可以跳出。

$$y_{ij}^{rl} = 1 \quad \forall (i, j, r, l) \in CP \quad (4.2.1)$$

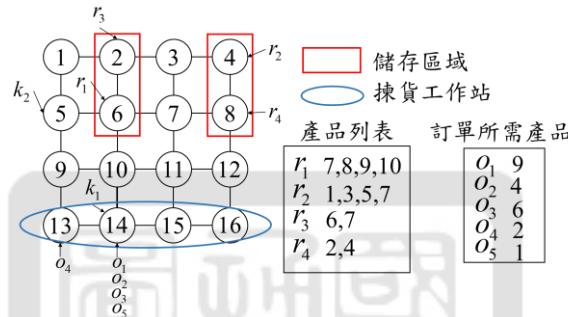


圖 4.1 16 節點範例

```

Robot 1 routing:
14 -> 10 -> 6 -> 10 -> 14 -> 15 -> 11 -> 7 -> 6 -> 10 -> 14 -> 10 -> 10 -> 10 -> 6 -> 6 -> 7 -> 8 -> 8 -> 8 -> 8 ->
8 -> 8 -> 8 -> x
Robot 2 routing:
5 -> 6 -> 2 -> 6 -> 7 -> 8 -> 12 -> 11 -> 15 -> 14 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 9 -> 9 ->
5 -> 1 -> 2 -> x
Rack 1 routing:
6 -> 6 -> 6 -> 10 -> 14 -> 15 -> 11 -> 7 -> 7 -> 7 -> 7 -> 7 -> 7 -> 7 -> 7 -> 8 -> 8 -> 8 -> 8 -> 8 ->
8 -> 8 -> 8 -> x
Rack 2 routing:
4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 ->
4 -> 4 -> 4 -> x
Rack 3 routing:
2 -> 2 -> 2 -> 6 -> 6 -> 6 -> 6 -> 6 -> 10 -> 14 -> 10 -> 10 -> 10 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 ->
6 -> 6 -> 6 -> x
Rack 4 routing:
8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 12 -> 11 -> 15 -> 14 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 13 -> 9 -> 9 ->
5 -> 1 -> 2 -> x

```

圖 4.2 滾動式求解 16 節點範例第一步

```

Robot 1 routing:
14 -> 10 -> 6 -> 10 -> 14 -> 15 -> 16 -> 12 -> 8 -> 4 -> 3 -> 7 -> 11 -> 15 -> 14 -> 15 -> 11 -> 7 -> 3 -> 4 -> 4 ->
4 -> 4 -> 4 -> x
Robot 2 routing:
5 -> 6 -> 2 -> 6 -> 7 -> 8 -> 12 -> 11 -> 15 -> 14 -> 13 -> 9 -> 5 -> 6 -> 10 -> 14 -> 14 -> 10 -> 6 -> 5 -> 1 ->
2 -> 2 -> 2 -> x
Rack 1 routing:
6 -> 6 -> 6 -> 10 -> 14 -> 15 -> 16 -> 12 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 -> 8 ->
8 -> 8 -> 8 -> x
Rack 2 routing:
4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 4 -> 3 -> 7 -> 11 -> 15 -> 14 -> 15 -> 11 -> 7 -> 3 -> 4 -> 4 ->
4 -> 4 -> 4 -> x
Rack 3 routing:
2 -> 2 -> 2 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 -> 6 -> 10 -> 14 -> 14 -> 10 -> 6 -> 6 -> 6 ->
6 -> 6 -> 6 -> x
Rack 4 routing:
8 -> 8 -> 8 -> 8 -> 8 -> 12 -> 11 -> 15 -> 14 -> 13 -> 9 -> 5 -> 5 -> 5 -> 5 -> 5 -> 5 -> 5 -> 5 -> 1 ->
2 -> 2 -> 2 -> x

```

圖 4.3 滾動式求解 16 點範例第二步

用滾動式求解方法可以大幅縮減運算時間，圖 4.1 的例子原本一次求解 5 筆訂單要花費 1 萬多秒的時間，改使用滾動方式求解僅需 200 秒左右，但因為核心部分依然是使用數學模式求解，因此當遇到更大規模時同樣會遭遇到瓶頸，因此我們將嘗試設計不使用數學模式的演算法來求解。

4.2 貪婪演算法

在本研究的問題設定下，如何避免貨架或是機器人間發生碰撞是非常重要的，此外在避免碰撞的同時，還要兼顧如何選擇合適的貨架問題，因此光是要產生一組好的可行解便有相當程度的難度。由於貨架與機器人的位置會隨著時間根據使用與否而變動，因此在原始網路圖上用最短路徑的找法很容易會有碰撞的發生，而當碰撞發生時再來進行修正的話容易「牽一髮而動全身」，或許可能於他處再發生碰撞，導致尋找可行解的困難。針對此路徑規劃之困難，我們可在層空（在此亦為時空）網路圖上一次處理一筆訂單，自己選定的訂單中，選擇當下最快可抵達又避撞的機器人與貨架組合路徑，接著再選擇其最快能回到的儲存區與其路徑，將整條路徑固定，刪除任何可能與之碰撞的節線，避免之後搜尋避撞路徑的困擾，流程如表 4.2。

首先設定初始的訂單完成時刻及貨架歸位時刻為 $L+1$ ，接著針對每個可用貨架以及機器人判斷最快能抵達的時刻。這邊要先檢查這組機器人及貨架的組合先前是否有被使用過，若有，則可以在上次任務結束的工作站進行截斷，並取消後續歸位

的路徑，透過 *special_find_Path* 函式進行路徑規劃；反之，若是先前沒有被使用過的組合，則透過 *find_Robot* 函式檢查由機器人的位置移動至貨架的位置的可行路徑，再透過 *find_Path* 函式檢查由貨架的位置移動至訂單所在的揀貨工作站的避撞路徑。這些用來確認路徑的函式皆是透過給定起始時刻跟結束時刻，在層空網路圖上用深度優先搜尋(Depth First Search, DFS)的概念檢查其連通性。

表 4.2 貪婪演算法流程

Greedy Algorithm

1. **For** o in O **do**
 2. **For** (r, k) in (R, K) **do**
 3. **If** (r, k) in *ReUse* **then**
 4. **For** t in (last finish time, L) **do**
 5. **If** *special_find_Path* $(r, k, pos, dest[o], last_finish, t)$ **then**
 6. update path, network
 7. **Else then**
 8. **For** t in (r 's available time, L) **do**
 9. **If** *find_Robot* $(r, r_pos, k_pos, avail_t, t)$ **then**
 10. update robot path
 11. **For** t' in (t, L) **do**
 12. **If** *find_Path* $(k, k_pos, dest[o], t, t')$ **then**
 13. update path, *ReUse*, network
 14. **For** t in (finish time, L) **do**
 15. **If** *find_Path* $(k, dest[o], storage, finish_t, t)$ **then**
 16. update path, network
-

針對每個貨架與機器人組合使用遞迴的方式來檢查連通性以尋找可行路徑如表 4.3 與表 4.4 所示。*find_robot* 函式主要用於判斷機器人 *robot* 是否能在時刻 t_start 至 t_end 間能夠從節點 *from* 移動至節點 *to*，因此在函式一開始先進行基本的可行性判斷：例如機器人或貨架不可用，已被先前的訂單保留下的情形，又或是給定起始時間已超過截止時間等等皆為「不可行」。接著若節線(*from, to, t_start + 1*)存在於屬於機器人的節線集合 A_K 中，則回傳「可行」，若否，則針對節點 *from* 的

相鄰節點 n 去判斷是否能在時刻 $t_start + 1$ 至 t_end 移動到節點 to 。每當有可行節線，則把節線暫存於集合 $temp_KP$ ；而當發現不可行時，便將集合 $temp_KP$ 還原至空集合。

表 4.3 遞迴機器人路徑搜尋

Find Robot

Data: $temp_KP, A_K$

```

1. Function find_robot( $k, from, to, t\_start, t\_end$ )
2.   If  $t\_start >= t\_end$  then
3.      $temp\_KP = \emptyset$ 
4.     return 0
5.   Else if robot  $k$  not available before  $t\_end$  then
6.      $temp\_KP = \emptyset$ 
7.     return 0
8.   Else if ( $from, to, t\_start + 1$ ) in  $A\_K$  then
9.      $temp\_KP.add((from, to, t\_start + 1))$ 
10.    return 1
11.  Else then
12.    For  $n$  in  $from$ 's adjacent list do
13.      If ( $from, n, t\_start + 1$ ) in  $A\_K$  then
14.        If find_robot( $k, n, to, t\_start + 1, t\_end$ ) then
15.           $temp\_KP.add((from, n, t\_start + 1))$ 
16.        return 1
17.       $temp\_KP = \emptyset$ 
18.    return 0

```

Result: $temp_KP$

搜尋貨架路徑的函式 *find_path* 相當類似於尋找機器人的函式，不同的是貨架的移動需要機器人的搬運，因此在判斷節線($from, to, t_start + 1$)是否存在於貨架的節線集合 A_R 之外，亦需要判斷是否存在於機器人的節線集合 A_K 。第二個不同之處在於，會使用到貨架的搜尋函式有兩種情形：首先是貨架從儲存區域移動至

揀貨工作站以完成任務的過程，再來則是從工作站返回儲存區域，因此會產生兩種不同的貨架路徑原始狀態 RP 。若是在第一階段則 $RP = \emptyset$ ，在第二階段則 RP 將保留先前已找好的儲存區至工作站的路徑。

表 4.4 遞迴貨架路徑搜尋

Find Path

Data: $temp_RP, RP, A_K, A_R$

```

1. Function find_path(rack, from, to, t_start, t_end)
2.   If  $t\_start >= t\_end$  then
3.      $temp\_RP = RP$ 
4.     return 0
5.   Else if rack or node to not available before  $t\_end$  then
6.      $temp\_RP = RP$ 
7.     return 0
8.   Else if ( $from, to, t\_start + 1$ ) in  $A\_K$  and ( $from, to, t\_start + 1$ ) in  $A\_R$  then
9.      $temp\_RP.add((from, to, t\_start + 1))$ 
10.    return 1
11.  Else then
12.    For  $n$  in  $from$ 's adjacent list do
13.      If ( $from, to, t\_start + 1$ ) in  $A\_K$  and ( $from, to, t\_start + 1$ ) in  $A\_R$  then
14.        If find_path(rack, n, from, to, t_start + 1, t_end) then
15.           $temp\_RP.add((from, n, t\_start + 1))$ 
16.          return 1
17.       $temp\_RP = RP$ 
18.    return 0
Result:  $temp\_RP$ 

```

當找到一條可行的路徑時，我們要固定這條路徑，並刪除其它可能會與之碰撞的走法，流程如表 4.5。根據機器人與貨架的不同，刪除不同集合中的節線，包括被選中要走的節線及其反向的節線，以及在該時刻連入節點 to 的所有節線。

表 4.5 刪除節線流程

Delete Arc

Data: A_K, A_R

```

1: Function delete_arc(agent, from,to,t)
2. If agent = robot then
3.   A_K.delete((from,to,t))
4.   A_K.delete((to,from,t))
5. For n in to's adjacent list do
6.   A_K.delete((n,to,t))
7. Else if agent = rack then
8.   A_R.delete((from,to,t))
9.   A_R.delete((to,from,t))
10. For n in to's adjacent list do
11.   A_R.delete((n,to,t))

```

Result: A_K, A_R

4.3 重新排列訂單

在 4.1 節與 4.2 節介紹的方法中，我們都是一次考慮部分訂單，並固定其對應到的貨架路徑，可知訂單的順序會影響最後目標值，因此我們以圖 4.4 為例，設計了三種方法來將訂單重新排列順序。

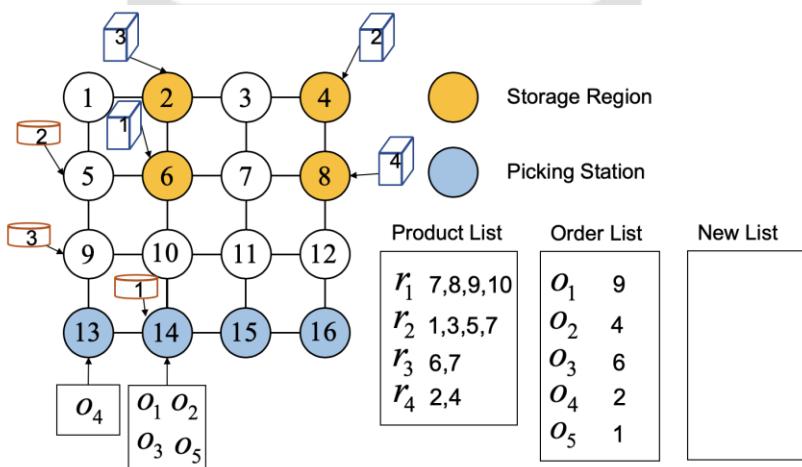


圖 4.4 排序範例

第一種方法(法 1)使用求解數學模式來將訂單重新排序，根據可用來完成某一訂單的貨架數量，以及貨架距離該訂單被分配到的工作站的距離，計算出一個分數，當距離愈近或是貨架數量愈多則該分數之值將愈高。其目標為最小化該分數與新排序乘積的總和，而限制式則規範每筆訂單都必須也只能對應到唯一的一個新編號。

參數

α, β 距離與可用貨架數的權重

$dist_i$ 可滿足訂單 $i \in O$ 最近的貨架距離

$avail_i$ 可滿足訂單 $i \in O$ 的貨架數

變數

a_{ij} 原本排序在第 $i \in O$ 之訂單新排序在第 $j \in O$ 為 1；反之為 0

目標式

$$\min \sum_{i \in O} \sum_{j \in O} j \times a_{ij} \times \left(\frac{\alpha}{dist_i} + \beta \times avail_i \right)$$

限制式

$$\sum_{j \in O} a_{ij} = 1 \quad \forall i \in O \quad (4.1.1)$$

$$\sum_{i \in O} a_{ij} = 1 \quad \forall j \in O \quad (4.1.2)$$

$$a_{ij} \in \{0,1\} \quad \forall (i, j) \in A \quad (4.1.3)$$

第二種方法(法 2)則從貨架選擇的角度發想，已知一個包含所有訂單所需的產品之集合 P ，以及各工作站 m 所需產品集合 $P_{Station}[m]$ 與所需負責的訂單 $O_{Station}[m]$ ，希望除了使用較少的貨架減少貨架交換次數，同時考量貨架離工作站的距離，表 4.6 為訂單重新排序流程。

表 4.6 訂單重新排序流程

Sort Order

Data: $P, P_{Station}, O_{Station}$

- 1: $Priority = \text{null}$
2. **While** ($P \neq \emptyset$)
3. choose top k rack r_1, \dots, r_k include the most product in P
4. choose the closest rack r in r_1, \dots, r_k to picking station m in N_P
5. **For** o in $O_{Station}[m]$ **do**
6. **If** r can finish order o **then**
7. $Priority.add(o)$
8. $O_{Station}[m].delete(o)$
9. delete product needed from rack r in $P_{Station}[m]$
10. update rack r 's position
11. **For** p in P **do**
12. **If** p not in any $P_{Station}$ **then**
13. $P.delete(p)$

Result: $Priority$

假設現在有 k 台機器人，那同時最多可移動的貨架數亦為 k 個，所以我們先從貨架集合中挑選 k 個貨架 r_1, \dots, r_k ，此 k 個貨架會包含最多集合 P 內的產品，針對這些貨架分別尋找每個貨架可服務的最近工作站後，再從中挑選出距離最短的工作站 m 與貨架 r 組合，接著將工作站 m 需要貨架 r 完成的所有訂單加入新的排序當中，移除工作站 m 對這些產品的需求，並更新貨架 r 的位置至工作站 m ，此步驟能夠避免下次選到該貨架時又從原始位置開始計算距離，使得貨架重複利用的機會提升。最後根據改變後的產品集合 $P_{Station}$ 來更新所有訂單需要的產品集合 P ，重複步驟直到 P 為空。

以圖 4.4 為例，因機器人有三台，因此挑選出三個能包含最多所需要貨物的貨架 r_1, r_2, r_4 ，分別計算其與需要該貨架的工作站距離如圖 4.5 所示，其中貨架 r_1 距離工作站 14 最近，因此選擇該貨架，並將該貨架可在工作站 14 完成的訂單 o_1 加入新的訂單順序，並更新貨架 r_1 的位置至節點 14 如圖 4.6，接著便可重複步驟，直到所有訂單都已加入新的訂單序。

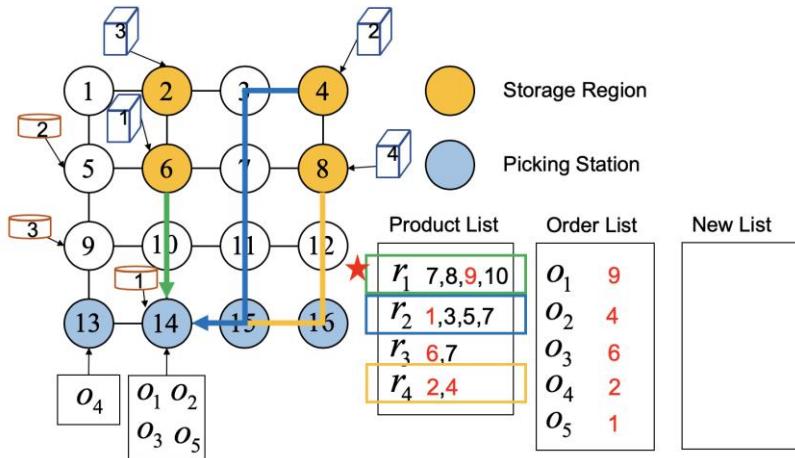


圖 4.5 法 2 步驟一

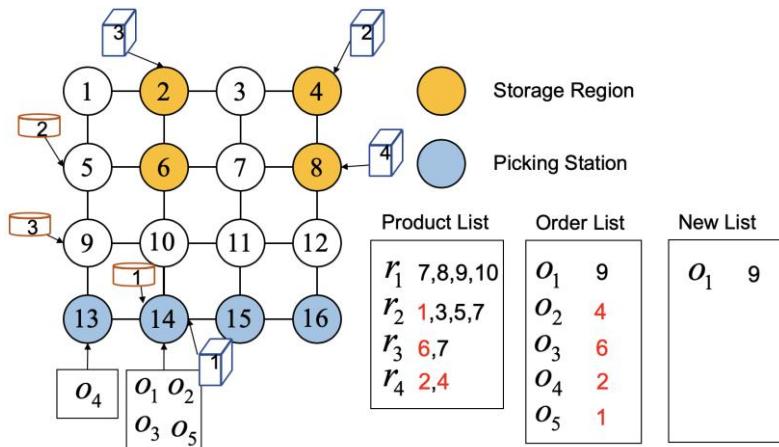


圖 4.6 法 2 步驟二

第三種方法(法 3)一樣站在貨架的角度，將貨架與最近有需求的揀貨工作站的距離進行排序，並考量機器人移動至貨架所需要的時間，接著由近至遠依序選擇貨架，並將其可完成的訂單加入新的訂單序當中。以圖 4.4 為例，先分別計算四個貨架與工作站的距離，並挑選其中最短的 r_1 如圖 4.7，將 r_1 可完成的訂單，不分工作站全數加入新訂單序並選取下一個距離最短的貨架 r_3 如圖 4.8，接著將 r_3 可完成的訂單不分工作站全數加入新訂單序，重複步驟如圖 4.9，當選擇完 r_4 時，會將訂單 O_2, O_4 加入新訂單序。

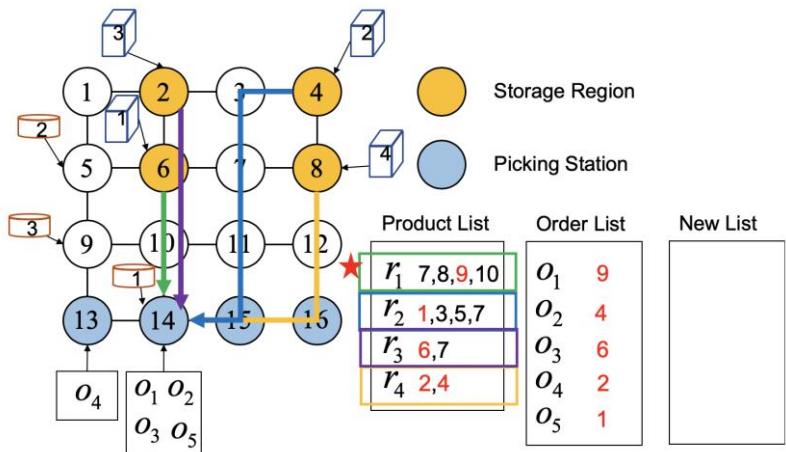


圖 4.7 法 3 步驟一

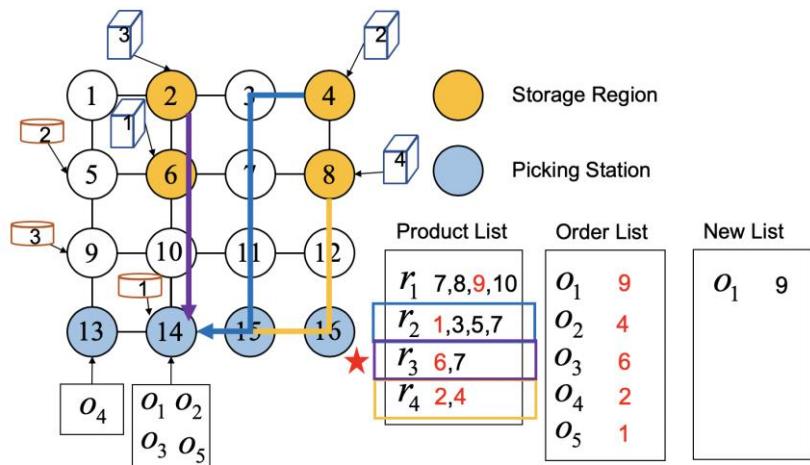


圖 4.8 法 3 步驟二

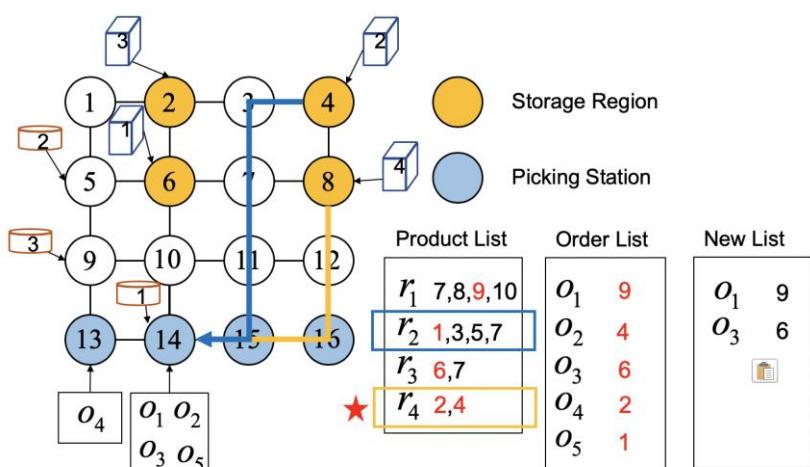


圖 4.9 法 3 步驟三

4.4小結

本章介紹了兩種不同的演算法以加速求解，若是使用滾動式解法能夠大部分保留解的品質，但在更大規模的例子中依然會遇到求解速度的瓶頸；貪婪演算法的優勢在於不需要花多餘的步驟進行排解碰撞的調整，因此能快速地得到可行解，缺點是數學模式能考慮到的不同任務間的協同機制，貪婪演算法並無法考量到，但由於速度快上很多，因此在實際應用上較為可行。同時，因為我們設計的兩種演算法皆會固定先操作的訂單，因此何者有較高的優先序會影響到最終的目標值，尤其是貪婪演算法是每做一筆即固定，受到的影響會比滾動式求解更大，因此我們也設計了三種不同的訂單排列方法以改善求解品質。



第五章 數值測試與討論

本章將對第三章提出之數學模式與第四章提出之演算法進行測試，包括求解大小、速度及品質進行比較，並分析可能原因。本研究測試環境為 Windows 10 作業系統，搭配 Intel Core i7-8700，3.20GHZ*6 處理器，與 8G 記憶體。所有程式包括數學模式以及演算法皆以 Python 為程式語言撰寫，數學模式利用 Gurobi 9.0.2 版求解。

5.1 測試資料參數設定

本研究根據給定不同參數(表 5.1) 產生測資，其中我們用於測試的網路圖的產生皆為正方形之網格圖，因此 n 皆為平方數，倉儲佈置如圖 5.1，儲存區域皆相隔一排走道，工作站為最下方一排的 \sqrt{n} 個點，為貼近實際情形，也設定貨架個數 $R \geq$ 機器人數 K ，求解層數給定基本值 25，若因為層數不足導致無解再將基本值每次 +5 進行求解。訂單排法有五種依序分別為，原始亂數產生、法 1、法 2、法 3 之訂單重排以及由模型結果得到之訂單完成序。

表 5.1 參數設定

參數設定						
節點 數	機器 人數	貨架 個數	求解 層數	訂單 排法	訂單 數量	產品種類
n	K	R	L	$type$	O	P

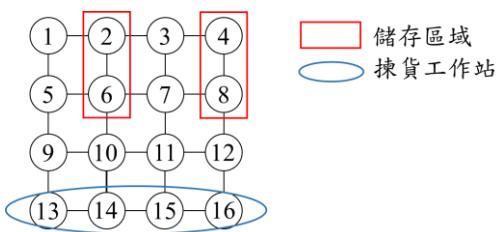


圖 5.1 倉儲佈置

5.2目標式不同之比較

在 3.4.2 節我們提供兩種不同的目標式，一是最遲完工時刻，二是總等待時間，在此我們將比較兩者求解時間的差異，並觀察使用何種目標式較為合適。

由表 5.2 可以發現，當使用最遲完工時刻作為目標式時，求解時間比起使用總等待時間要多上許多，推測是因為不使用最遲完工時刻作為目標式，可以省去整數變數 w ，使得整個數學模式的變數皆為 0,1 變數，同時也可減少許多限制式如式(3.4.19)。在同樣的小例子中，以最遲完工時刻為目標式甚至已經無法在一小時內得到可行解了，因此若是以求解規模來看，是使用總等待時間當作目標式較為合適。但我們亦希望可以得到好的最遲完工時刻，因此我們觀察使用總等待時間當作目標式的時候，計算出的最遲完工時刻表現如表 5.3 所示。

表 5.2 目標式不同之比較

n	O	R	K	P	Obj makespan		Obj total waiting	
					CPU Time(s)	Status	CPU Time(s)	Status
16	5	4	2	8	12	Optimal	4	Optimal
16	5	4	4	8	200	Optimal	11	Optimal
16	8	4	2	8	55	Optimal	14	Optimal
16	8	4	4	8	627	Optimal	48	Optimal
25	5	6	4	10	3324	Optimal	13	Optimal
25	8	4	4	8	1298	Optimal	47	Optimal
25	8	6	2	8	3600	Feasible	1102	Optimal
25	8	6	2	10	3600	Infeasible	1157	Optimal

從表 5.3 的結果也可以看到，兩者所得到的最遲完工時刻差異是不大的，但所花費的求解時間卻相差很多，因此在實用上會建議求解以總等待時間作為目標式的模式即可，接下來後續實驗若是有比較數學模式的部分，也會使用總等待時

間當目標式的模式來比較。

表 5.3 目標式不同最遲完工時刻之比較

n	O	R	K	P	<i>Obj makespan</i>		<i>Obj total waiting</i>		
					CPU Time(s)	makespan	CPU Time(s)	makespan	Gap(%)
16	5	4	2	8	12	6	4	7	16.67
16	5	4	4	8	200	7	11	8	14.29
16	8	4	2	8	55	8	14	8	0.00
16	8	4	4	8	627	6	48	6	0.00
25	5	6	4	10	3324	6	13	7	16.67
25	8	4	4	8	1298	8	47	9	12.50
25	8	6	2	8	3600	10	1102	13	30.00
25	8	6	2	10	3600	--	1157	14	--
36	10	10	6	25	3600	--	3600	--	--

5.3 數學模式與演算法之比較

本節將進行數學模式與演算法在求解規模、時間、品質之比較，也測試在不同的訂單排序下如何影響演算法的求解。

5.3.1 一次性求解與批次求解

將一次性求解數學模式 M_0 與批次求解演算法 M_{iter} 進行比較如表 5.4，其中 M_0 的 Gap 是利用最佳化軟體 Gurobi 之測試結果(求解時限為 3600 秒)，計算方式如下：

$$Gap = \frac{UB - LB}{UB} \times 100\% \quad UB \text{ 及 } LB \text{ 為 Gurobi 所求得之上下界。}$$

M_{iter} 的 Gap 則是以 M_0 求解的結果當作最佳解，計算方式如下：

$$Gap = \frac{M_{iter} - M_0}{M_0} \times 100\% \quad \text{分別帶入兩者求解之值。}$$

M_{iter} 使用三種排序中 Gap 最小的一種來進行比較。

表 5.4 一次性與批次數學模式比較

n	O	R	K	P	M_0		M_{Iter}	
					CPU Time(s)	Gap(%)	CPU Time(s)	Gap(%)
16	8	4	4	10	80	0	8	6.25
16	10	4	2	8	520	0	20	5.56
16	10	4	2	10	127	0	7	19.60
25	8	4	2	8	369	0	29	0
25	8	4	2	10	1128	0	12	0
25	8	6	2	8	1102	0	24	0
36	10	10	6	25	3600	--	600	--

表 5.4 可以看出，透過使用批次求解，能夠有效地減少求解時間，並達到與一次性求解幾乎相同的表現。

5.3.2 數學模式與貪婪演算法

接著我們比較數學模式與貪婪演算法(表 5.5)，其中 *Greedy* 的 Gap 也是以 M_0 的求解結果當作基準進行比較。

表 5.5 數學模式與貪婪演算法比較

n	O	R	K	P	M_0		<i>Greedy</i>	
					Obj	CPU Time(s)	Obj	CPU Time(s)
16	5	4	2	10	24	45	32	0.003
16	8	4	2	10	46	68	56	0.003
16	8	4	4	10	32	80	38	0.004
16	10	4	2	8	72	520	98	0.005
16	10	4	4	8	36	163	43	0.005
25	5	6	2	10	42	492	43	0.027
36	10	10	6	25	--	3600	86	2.376

從表 5.5 可以發現，一次性的數學模式已經開始無法得到可行解了，因此我們比較滾動式求解與貪婪演算法的求解性能，兩者都限時 600 秒，結果如表 5.6。

表 5.6 滾動式求解與貪婪演算法比較

n	O	R	K	P	M_{Iter}		Greedy		
					Obj	CPU $Time(s)$	Obj	CPU $Time(s)$	$Gap(%)$
49	10	10	6	25	86	600	92	80	6.98
64	10	10	6	25	--	600	108	77	--
64	10	10	6	35	82	204	83	57	1.22

從以上結果來看，貪婪演算法的運行速度快上許多，但也發現由於我們使用遞迴的方式確認連通性，因此在節點變多時，層空網路圖的大小變大也使得求解時間增加很快。除了速度外，貪婪演算法的求解的品質也尚可接受，但並不是到非常小的差距，我們希望透過觀察數學模式與貪婪演算法求解出路徑來釐清這些差異是由哪些類型的原因所造成。

以圖 5.2 為例，假設有一 4×4 網格圖，其中節點 2、4、6、8 為儲存區域，節點 13 到 16 為揀貨工作站點，在儲存區域的方塊狀為貨架，底部圓柱狀為機器人，訂單分布如表 5.7，可用貨架如表 5.8。

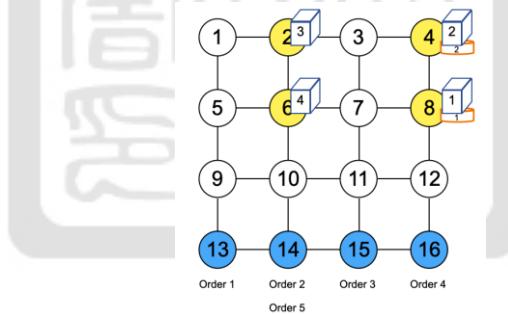


圖 5.2 16 節點範例

圖 5.3 至圖 5.21 為數學模式的求解過程，從中我們可以發現一些用貪婪演算法無法考慮到的情形，例如步驟(1)與步驟(2)，貨架 1 被移動，但我們觀察整個求解過程後發現其實貨架 1 是沒有被工作站使用到的，因此它的移動只是為了使其它的貨架能夠有更短的路徑，但大多貪婪演算法幾乎都讓機器人一直載著貨架移動，不會考慮到讓機器人可以半途離開原貨架、去移動別貨架、再回來繼續搬運原貨架這種情況；另外我們可以發現，最後歸位的時候貨架的位置也與一開

始出發位置不同，反觀在貪婪演算法時，因為規劃一筆訂單的當下要假設其它的機器人不會移動，因此有很大的機率會回到原位，進而影響到後續的路徑規劃。

表 5.7 範例訂單分布

工作站節點	訂單編號
13	1
14	2、5
15	3
16	4

表 5.8 範例可用貨架分布

訂單編號	可用貨架
1	2
2	2
3	2、4
4	3
5	1、4

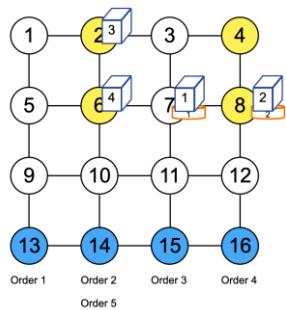


圖 5.3 模式求解(1)

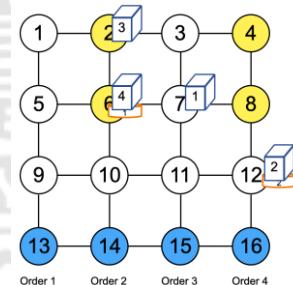


圖 5.4 模式求解(2)

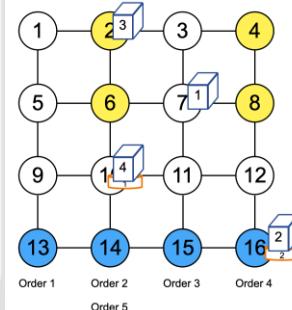


圖 5.5 模式求解(3)

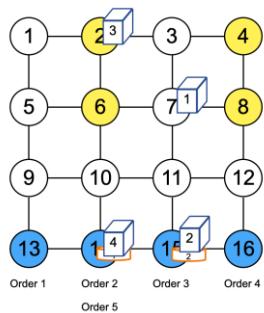


圖 5.6 模式求解(4)

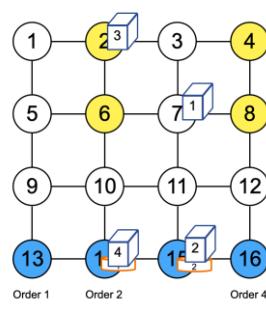


圖 5.7 模式求解(5)

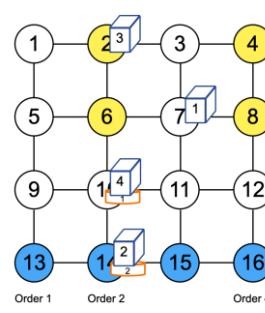


圖 5.8 模式求解(6)

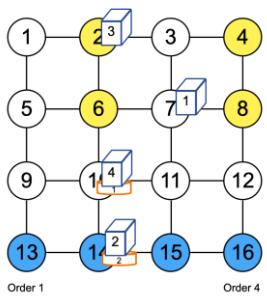


圖 5.9 模式求解(7)

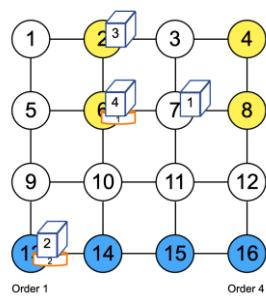


圖 5.10 模式求解(8)

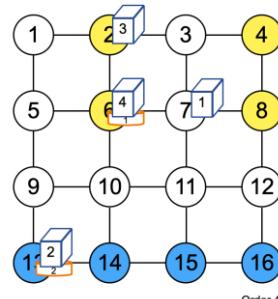


圖 5.11 模式求解(9)

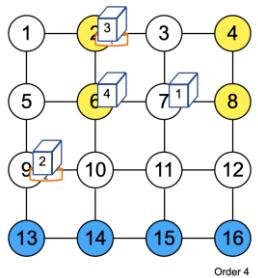


圖 5.12 模式求解(10)

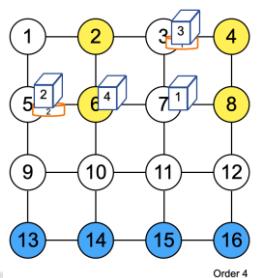


圖 5.13 模式求解(11)

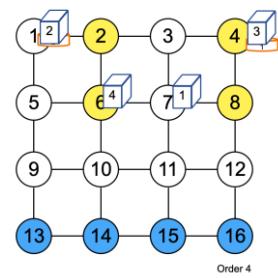


圖 5.14 模式求解(12)

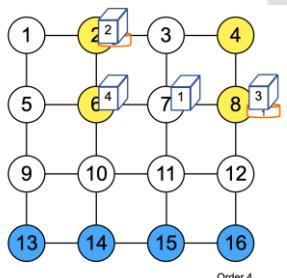


圖 5.15 模式求解(13)

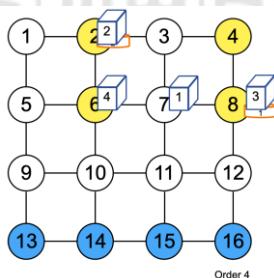


圖 5.16 模式求解(14)

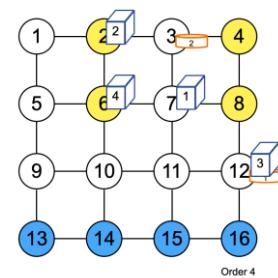


圖 5.17 模式求解(15)

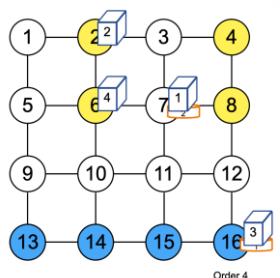


圖 5.18 模式求解(16)

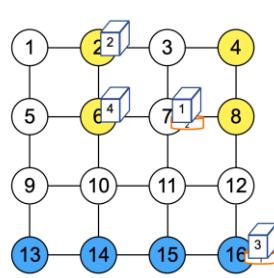


圖 5.19 模式求解(17)

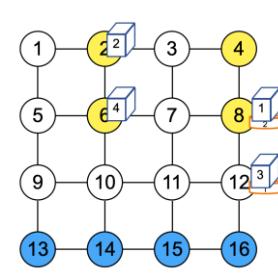


圖 5.20 模式求解(18)

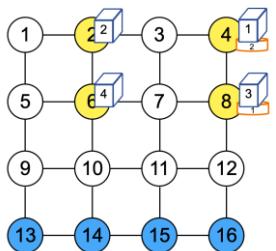


圖 5.21 模式求解(19)

5.4小結

本章比較完數學模式與演算法之優缺點，數學模式 M_0 花費許多運算時間，但能得到最好的結果，而 M_{Iter} 減少了每次的求解規模，因此可以透過犧牲部份品質換取節省許多時間，但當求解規模擴大，同樣會花費很大量的時間求解。貪婪演算法可以很快的得到可行解，求解規模也能比前兩種方法大上很多，品質也可以接受，若是作為起始解再搭配其它方法，可能會有更好的結果。

第六章 結論與未來研究方向建議

6.1 結論與貢獻

在可動式貨架倉儲的架構下，可以將整個系統分為六個階段，包括(1)產品如何分配擺放至哪個貨架？(2)訂單如何分配至工作站？(3)工作站如何選擇貨架來完成訂單？(4)貨架如何選擇機器人（或機器人如何選貨架）來運送？(5)機器人與貨架的路徑規劃？(6)貨架完成任務後該回歸至何處暫存？過去的文獻大多只針對其中單一部分研究，但現實中，這些決策往往互相牽連影響，尤其是貨架與機器人的選擇與後續的路徑規劃。本研究認為若是能整合多個階段進行規劃，應能得到更佳（早）的訂單完工時刻，

我們在第二章回顧了包含貨架選擇及多機器人路徑規劃相關問題以及其它相關可動式貨架倉儲相關文獻，了解過去貨架選擇的策略及避免碰撞的方法。在貨架選擇之相關文獻中，大多考量以最少的貨架數來完成訂單，此種考慮方法會忽略距離造成因素，導致後續規劃路徑時要花費較久的時間；多機器人路徑規劃相關文獻，則是假設機器人數量等於要搬運的貨架數量，忽略機器人選擇的問題並給定貨架的起訖點，且抵達訖點後也不需考慮後續回暫存區的問題，但實際上貨架在揀貨完畢後若仍持續停留在工作站，當然會造成後續塞車；此外，該種問題假設下，貨架其實無法被不同的工作站重複使用，因此照理應該要將該貨架如何歸位（離開工作站而返回合宜的位置）之路線一位規劃，但目前並無文獻探討之。

本研究提出的方法能夠整合從機器人與貨架選擇到貨架回歸儲位的整個流程，過往文獻皆不似本研究處理如此龐大之整合性規劃問題。在這樣龐大的問題設定下，如何避免機器人與貨架互相碰撞，同時考慮貨架重複使用的可能性與機器人與貨架搭配的多種組合，變得相當困難。

本研究在第三章提出一個混整數規劃數學模式用以解決此整合性之規劃問

題，透過給定機器人與貨架的避撞限制與任一可選貨架必須走訪完成任務的限制，並以機器人與貨架在層空網路圖上的節線經過與否作為變數。至於最終機器人與貨架該如何配對、訂單又該用哪個貨架來完成等問題，皆可透過最終的路徑求解結果回推而得，因此不需要新增多餘的變數至數學模式當中，以縮小問題規模。

為了能更有效地處理更大規模問題的求解過程，本研究在第四章提出了滾動式求解數學模式的啟發式方法，每次求解較小規模的簡化版問題，並固定較早被完成的訂單所使用的路徑。這種解法所得到的目標值，經測試後我們發現與原先求解耗時的大規模範例之最佳目標值十分接近，然而這種作法的求解時間卻可大幅減少；同時我們也提出能非常迅速得到可行解的貪婪演算法，但礙於問題架構的龐大，貪婪演算法的求解品質較差。也因為此兩種方法皆會與訂單處理的先後順序非常相關，因此我們也考量可使用的貨架個數與貨架到工作站的距離，據此設計了三種不同的訂單重新排列方法進行測試。

綜合以上所述，本研究目前貢獻如下：

1. 整合多貨架選擇至機器人與貨架路徑規劃問題：

本研究提出之數學模式與演算法，整合了從機器人與貨架的選擇至貨架的歸位，過去相關研究並無發現有相關文獻提出辦法整合性規劃處理這樣的問題。在貨架選擇部分，不同於過往研究僅能考量單一工作站的訂單，本研究可同時處理多個揀貨工作站的貨架選擇問題，並依照後續實際路徑來當作選擇的依據；在多機器人協同路徑規劃部分，我們將機器人與貨架視為不同的物件，因此可處理機器人少於貨架數目的情形；過往研究中每個機器人都已被指派其起訖點，因此並無可重複利用同一機器人完成任務的可能；此外，過往研究皆無考量到處理完訂單後的貨架歸位時會影響正在來訪的貨架路徑。反之，本研究提出之數學模式及演算法，皆可重複使用貨架，亦考量貨架後續的歸位路徑。

2. 符合現實的交換機制：

過去文獻中，「包裹交換」之方式為兩相鄰之機器人可直接交換包裹，但移動式倉儲的設定中，兩個載著貨架的相鄰機器人，是無法直接進行直接交換的，因此過往研究的假設較不適合此類移動式倉儲。然而，我們的數學模式卻可以用更接近現實的交換機制來交換貨架。

3. 以層空網路建構之數學模式：

由於機器人間會發生碰撞，貨架間也會發生碰撞，因此需要兩個層空網路圖來分別表達機器人的網路以及貨架的網路，以最佳解所用到的階層節線表示機器人與貨架之移動之路徑，每一條節線可視為一步(或一個時間區間)，此作法為本研究首創。

4. 滾動式求解法

透過將原來的數學模式分次求解，我們可成功地縮短求解時間，擴展數學模式能求解的問題規模。在目前的測試結果顯示，若是給定適當的分次求解問題規模與訂單順序，此方法能夠得到與原模式幾乎相同的目標值。

5. 可用於得到可行解之貪婪演算法：

本研究提出一個就算在如此龐大問題設定下，亦能快速得到可行解的貪婪演算法，可適用快速且即時的規劃需求，也能提供其它演算法當作初始解。

6.2建議之未來研究方向

本研究上有不少未臻完善之處，以下列出幾點仍可改善或延伸的研究議題：

1. 貪婪演算法之改善：

本研究提出之貪婪演算法，在求解品質的部分還有很大的改善空間，可透過觀察數學模式求解的結果，來看演算法未考慮到的狀況再加以改善；另外若是能作為初始解搭配其它的啟發式演算法，或許亦能夠收斂至更好的解。

2. 訂單重排之改善：

本研究提出之訂單重排方法，也有許多不周之處，該如何更有系統地重新編排訂單也是未來可再研究的問題。

3. 訂單的產品種類與數量：

目前本研究皆假設每筆訂單僅需一種產品，並忽略其數量。在現實中，訂單可能包含多種產品，且每種產品所需要的數量也可能不同；同時，每個貨架上所存放的產品數量也可能有所不同，因此如何將本研究之問題擴展至能處理包含產品數量考量的問題，也是另一個難題。

4. 機器人的電量限制：

目前本研究皆不考慮機器人能夠在倉儲內移動的電力耗費，現實中當然會有電池電量與續航力的限制，導致機器人必須在其電量耗盡前進行充換電，因此，未來更實務的研究可將機器人的電量耗損與充換，列入整體路線與排程的規劃。

5. 機器人的修復問題：

在現實生活中，機器人可能會遭遇故障的情形，當倉儲內有發生機器人故障需要排除狀況的時候，過往會封鎖部分區域，限制其它的機器人移入封鎖區，以讓維修人員可進入修理，因此要如何訂定封鎖區或許也值得探討。目前有一作法是讓維修人員穿上有條碼的背心，使機器人辨識該點必須繞過。在此種作法下，如何同時安排機器人與貨架再加上維修人員的整體路徑規劃，或也值得研究。

參考文獻

- Boysen, N., Briskorn, D., & Emde, S. (2017). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, 262(2), 550-562.
- Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3), 233-235.
- de Koster, R., Le-Duc, T., & Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2), 481-501.
- Drain, J. (2018). *Two algorithms for the package-exchange robot-routing problem*. Retrieved from <https://arxiv.org/abs/1812.10215>
- Erdem, E., Kisa, D. G., Oztok, U., & Schüller, P. (2013). *A general formal framework for pathfinding problems with multiple agents*. Paper presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence. Retrieved from https://www.researchgate.net/publication/244864562_A_General_Formal_Framework_for_Pathfinding_Problems_with_Multiple_Agents
- Hönig, W., Kiesel, S., Tinka, A., Durham, J. W., & Ayanian, N. (2018). *Conflict-based search with optimal task assignment*. Paper presented at the Seventeenth International Conference on Autonomous Agents and Multi Agent Systems. Retrieved from <https://dl.acm.org/doi/10.5555/3237383.3237495>
- Li, J.-t., & Liu, H.-j. (2016). Design optimization of amazon robotics. *Automation, Control and Intelligent Systems*, 4(2), 48-52.
- Li, Z. P., Zhang, J. L., Zhang, H. J., & Hua, G. W. (2017). Optimal Selection of Movable Shelves under Cargo-to-Person Picking Mode. *International Journal*

- of Simulation Modelling*, 16(1), 145-156.
- Ma, H., & Koenig, S. (2016). *Optimal target assignment and path finding for teams of agents*. Paper presented at the Fifteenth International Conference on Autonomous Agents & Multiagent Systems. Retrieved from <https://arxiv.org/abs/1612.05693>
- Ma, H., & Koenig, S. (2017). AI buzzwords explained: Multi-agent path finding (MAPF). *AI Matters*, 3(3), 15-19.
- Ma, H., Koenig, S., Ayanian, N., Cohen, L., Höning, W., Kumar, T. K., Uras, T., Xu, H., Tovey, C., & Sharon, G. (2017). *Overview: Generalizations of multi-agent path finding to real-world scenarios*. Retrieved from <https://arxiv.org/abs/1702.05515>
- Ma, H., Tovey, C., Sharon, G., Kumar, T. K. S., & Koenig, S. (2016). *Multi-agent path finding with payload transfers and the package-exchange robot-routing problem*. Paper presented at the Thirtieth AAAI Conference on Artificial Intelligence. Retrieved from <https://dl.acm.org/doi/10.5555/3016100.3016346>
- Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219, 40-66.
- Sundar, S., & Singh, A. (2012). A hybrid heuristic for the set covering problem. *Operational Research*, 12(3), 345-365.
- Surynek, P. (2015). *Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally*. Paper presented at the Twenty-Fourth International Joint Conference on Artificial Intelligence. Retrieved from <https://www.ijcai.org/Proceedings/15/Papers/272.pdf>
- Weidinger, F., Boysen, N., & Briskorn, D. (2018). Storage assignment with rack-moving mobile robots in KIVA warehouses. *Transportation Science*,

52(6), 1479-1495.

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1), 9-19.

Xiang, X., Liu, C., & Miao, L. (2018). Storage assignment and order batching problem in Kiva mobile fulfilment system. *Engineering Optimization*, 50(11), 1941-1962.

Yu, J., & LaValle, S. M. (2013a). *Planning optimal paths for multiple robots on graphs*. Paper presented at the 2013 IEEE International Conference on Robotics and Automation. Retrieved from <https://ieeexplore.ieee.org/document/6631084>

Yu, J., & LaValle, S. M. (2013b). *Structure and intractability of optimal multi-robot path planning on graphs*. Paper presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence. Retrieved from https://www.researchgate.net/publication/287708637_Structure_and_intractability_of_optimal_multi-robot_path_planning_on_graphs