國立成功大學

工業與資訊管理學系碩士班

碩士論文

以多台智慧倉儲機器人協作搬運移動式貨架之揀貨路徑規劃研究

A Study on the Multi-robot Collaborative Path Planning for Carrying Movable Racks in an Autonomous Picking System

研 究 生　　：胡 承 中

指 導 教 授：王 逸 琳　教授

中華民國一百一十年六月

國立成功大學

碩士論文

以多台智慧倉儲機器人協作搬運移動式貨架之揀貨路徑規劃研究

A Study on the Multi-robot Collaborative Path Planning for Carrying Movable Racks in an Autonomous Picking System

研究生：胡承中

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

周詩梵

吳治寧

指導教授：王逸琳

單位主管：翁慈宗

（單位主管是否簽章授權由各院、系（所、學位學程）自訂）

中 華 民 國 110 年 6 月 11 日

摘要

隨著電子商務的迅速發展，每日有大筆訂單需處理，這也帶動了物流倉儲的演變。新型態的物流倉儲系統 Robotic Mobile Fulfillment System (RMFS)，考慮移動式貨架，貨架內儲存空間分層分隔，且儲格大小可彈性調整，存放不同體積的商品，特別適合網路購物少量多樣的消費特性。此系統的揀貨作業是透過機器人搬運可移動式貨架至揀貨工作站，讓人員進行揀貨。倉儲內之訂單揀貨流程可分為(1)貨架選擇問題，與(2)多機器人路徑規劃問題(Multi-Agent Path Finding, MAPF)，本研究處理第二階段的多機器人路徑規劃問題，假設各訂單該由哪組貨架及機器人完成、各個貨架及機器人的初始位置、貨架上的商品品項與數量、各個揀貨工作站的訂單需求皆是已知，探討如何安排機器人去移動貨架，期望最小化訂單總等待時間。因為同時有多個貨架與機器人進行移動，如何規劃一組避免碰撞的路徑使其能以最快時間完成所有訂單，本質是一個 NP-hard 的多元商品網路流量問題。本研究使用時空網路圖建構一個整數規劃模型，規劃多台機器人的避撞路徑，使機器人以最短時間內完成所有訂單。為加速求解並能處理較大規模的路線與排程規劃，本研究也提出兩種貪婪演算法的求解機制，並使用模擬退火法(Simulated Annealing)迭代求解出最佳的訂單完成順序。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品質不錯的可行解，可供相關系統在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；多元商品網路流量；時空網路；模擬退火法

A Study on the Multi-robot Collaborative Path Planning for Carrying Movable Racks in an Autonomous Picking System

Cheng-Chung Hu

I-Lin Wang

Department of Industrial and Information Management, National Cheng-Kung University

SUMMARY

This research investigates a multi-agent path finding (MAPF) problem from a Robotic Movable Fulfillment System (RMFS). In this system, racks containing items are removable. Robots carry racks to workstations for finishing orders. It is an NP-hard problem to plan a collision-free path for several racks and robots. In this research, the number of robots is less than the number of racks. We route a robot from its position to its destined rack. Avoiding congestion near the workstation, we also route an off-duty rack from a workstation to its destined storage region. Most related literature simplifies this complex problem by assuming that each rack is a robot. They only plan collision-free paths for some racks (i.e., robots) to a workstation. Compared with our research, they only deal with around one-third of the routings. We propose an integer programming (IP) model based on the Multi-Commodity network flow problem on a time-space network to route robots and racks with a minimum total waiting time to collect all items following given orders. The IP model can only deal with small-scale cases. To deal with cases of a larger scale, we propose two greedy routing algorithms after a Simulated Annealing algorithm that calculates a good order sequence. Our computational experiments indicate that our solution methods are efficient and effective. This makes our solution methods arguably the first to deal with the MSPF problem of the most flexible settings with the best performance.

Key words: Multi-Agent Path Finding, Robotic Movable Fulfillment System, Multi-Commodity Network Flow, Time Space Network, Simulated Annealing

Introduction

In a traditional warehouse, racks storing items are unmovable with fixed locations. A staff approaches a rack to pick up some items, carries them to a workstation to finish a piking order. Such a picking method lacks efficiency. With the rapid development of technologies in automation, the Robotic Movable Fulfillment System (RMFS) has become a hot trend in modern warehouses. In an RMFS, every rack can be moved by a robot. Therefore, a staff no more needs to walk to pick up items but stay at a workstation waiting for some rack carried by a robot to arrive. Such an automatic RMFS can finish significantly more orders than a traditional one.

The challenges of RMFS involve several problems. In this research, we focus on a multi-robot collaborative path planning problem. Three routing problems in RMFS should be planned: (1) a collision-free path for a robot destinated to its assigned rack, (2) a collision-free path for a robot to carry an on-duty rack destinated to a workstation, and (3) a collision-free path for a robot to carry a just-off-duty rack from a workstation back to storage regions. To the best of our knowledge, most related RMFS literature deal with simplified settings: first, they always assume that each rack is a robot; second, they only deal with the second routing problem for an on-duty rack, as listed above and ignore the needs to relocate those just-off-duty agents (racks/robots). Since the number of robots is much less than the number of racks in practice, we believe all three routing problems should be planned.

MATERIALS AND METHODS

In this research, we assume all the orders need to be finished, the location for each robot, and the items and location for each rack are already given in the beginning of the planning horizon. We convert the space in a warehouse as a grid network and assume each move along an edge takes a unit of time to construct an acyclic time-space network. Such a setting is advantageous to model connections between arriving and departing agents and recording the movement of an agent over time.

We formulate this routing problem as an integer programming (IP) model M on a time-space network to calculate the best collision-free agent path with the minimum total waiting time. In this model M, we simplify the actions of a rack by only considering two actions for a rack by a robot: the lifting and the dropping processes. We do not need to set a binary variable to record the movement of a rack. Because a robot moves a rack, a path for a rack can be known by tracing the path of its assigned robot.

Since M is time-consuming for larger cases, we propose two greedy algorithms that calculate a feasible solution quickly. One is $Greedy_{order}$, the other is $Greedy_{task}$. The mechanism of $Greedy_{order}$ is to process one order at a time. Each order has three paths to be decided. After all three paths associated with one order are calculated, we process the next order. The mechanism of $Greedy_{task}$ is to process one task at a time. Each task represents planning a path. We observe that the order sequence may significantly affect the solution quality of these two greedy algorithms. Therefore we proposed a Simulated Annealing algorithm to determine a good order sequence for improving the solution quality.

RESULTS AND DISCUSSION

Our testings are performed on a personal computer with Windows 10, Intel Core i9-10900, 2.80GHZ Processors, and 8GB RAM. Gurobi 9.1 version is used for solving IP. Our mathematical programming model is implemented in Python, and the proposed algoritms are implemented in C++. We randomly generated orders, items, racks, and robots. Based on our computational results, the IP can not deal with some 36-node cases within 3600 seconds. Nevertheless, our proposed two greedy algorithms can calculate feasible solutions within one second. The range of the solution gap by $Greedy_{order}$ varies between 22% to 42%. And the range of the solution gap by $Greedy_{task}$ varies between 8% to 23%. Thus $Greedy_{task}$ has a better performance than $Greedy_{order}$.

CONCLUSIONS

To deal with the multi-robot collaborative path planning problem, we have proposed several solution methods. We construct a time-space network to formulate an IP model. To reduce the model scale, we simplify the action of a rack. Because our IP model can not efficiently calculate a feasible solution for large cases, we proposed two greedy algorithms to solve large cases. Our computational experiments show that the two proposed greedy algorithms can efficiently calculate good feasible solutions. Using the Simulated Annealing algorithm, we can converge the orders to a sequence as good input to the greedy algorithms to improve their effectiveness. To the best of our knowledge, the problem settings of this research are closer to the real-world problems than most related research, and our solution methods also have the best efficiency and effectiveness.

This research, however, also has room to be further improved. For example, we assume each order contains items of a single type, which should be extended to multiple types in practice. One of the challenges in designing a heuristic algorithm would be to consider moving some rack in some intermediate steps, which might seem unrelated but help clear out a shortcut for future rack movement.

誌謝

本論文得以順利完成，首先感謝指導教授王逸琳老師的指導，在這研究所兩年的期間，學術研究或是課業知識上給予許多幫助，很感謝老師願意花費大量時間指導我不足的地方，謝謝老師無私的付出。Proposal 與口試期間，承蒙書審委員李賢得老師與利德江老師以及口試委員吳浩庠老師與周詩梵老師的建議與指正，使本論文更完備，感謝各位教授。

感謝 Lab 61205 的學長姊，在我剛進實驗室什麼都還不熟悉的時候給予我不少幫助。感謝同屆的書桓，從碩一開始與你一起討論課業、準備助教課，到後面一起處理專案、論文，程式上或其他事務幫了我不少的忙，聽你分享其他 lab 的各種消息，感謝碩班這兩年有你的陪伴。Rani 和 Ocha，抱歉身為本地生的我卻沒有幫到妳們什麼事，希望妳們未來一切順利。

Lab 61205 的學弟妹威銓、郁恩與蕎仔，還有芯瑜，碩二的時候有你們約打桌遊及各種哈拉，感謝你們替我煩悶的研究生活增添一些樂趣，不過很抱歉，我好像沒有教你們太多東西，幫到什麼忙，希望你們的論文也能順利完成，找到理想的工作。另外，在台南時，多虧有嘉豪、皓程與冠霖，不時約出來聊天聚一聚，吃吃喝喝，還有旻諺跟勳德不定時關心我的近況，未來有空大家再一起約出來聊聊吧!

最後感謝我的家人，在求學期間有妳們的支持，讓我可以沒有後顧之憂的完成學業跟研究。謝謝!

目錄

表目錄

圖目錄

第一章 緒論

本研究探討已知每筆訂單由哪一機器人與貨架完成，如何進行路徑規劃以最小化路徑之移動距離或最遲完工時刻。本章第一節介紹本研究之背景與動機，並於第二節說明研究目的，最後一節介紹研究架構。

1.1 研究背景與動機

隨著網際網路盛行與行動上網的普及，網路購物越來越便利與順暢，為了在時間內，順利將商品送達消費者的手中，物流公司扮演重要的角色。在傳統物流作業中，揀貨流程是需要大量時間的環節。傳統揀貨得由揀貨員依照訂單的需求，直接走到固定式貨架前取貨，人員需於貨架中來回尋找商品。De Koster et al.（2007）指出揀貨作業屬於勞力密集高且高成本的活動，佔物流中心的作業成本高達 55%。Bartholdi & Hackman（2011）指出揀貨作業中行走時間是最浪費的，揀貨人員的行走時間並不會增加產品的價值，是最需要被改進，並且隨著品項增加、揀取範圍擴大，單靠人力進入貨架揀貨，已經無法負荷現今電商的出貨需求。

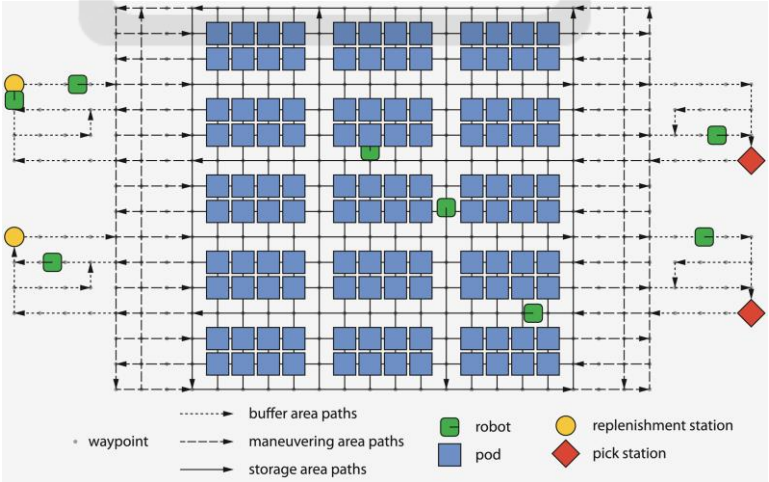

圖 1.1 RMFS 系統示意圖(Marius, et al., 2019)

圖 1.2 可移動式貨架與機器人(Wurman et al., 2008)

現有新型態的物流倉儲系統，倉儲佈置如圖 1.1，捨棄傳統的固架式倉儲，改用眾多可移動式的貨架（如圖 1.2）來儲存產品，充分運用倉庫空間，不似傳統固架式倉儲系統常因不當的儲位規劃以及狹窄之揀貨通道，導致繁忙又不順暢之揀貨作業貨架內儲存空間分層分隔，且儲格大小可彈性調整，存放不同體積的商品。揀貨流程是由機器人將貨架離地抬起，透過讀取地面上的條碼辨認方向進行移動，前往揀貨工作站（如圖 1.3），由揀貨人員進行揀貨，完成訂單後，再由機器人將貨架移至指定的儲貨區域。此倉儲系統，可減少人力，揀貨人員不必走進倉儲區，集中在揀貨工作站進行訂單分揀，在揀貨作業上可以大大降低揀貨錯誤率和提高訂單揀貨效率(Li & Liu, 2016)。許多著名電商如中國阿里巴巴、中國京東、美國亞馬遜及印度 GreyOrange，為了因應電子商務的迅速增長，皆有使用類似的倉儲系統來進行揀貨作業。

此倉儲系統的成功不僅是依靠其硬體設備的創新，同樣重要的是其決策方面的問題解決，像是如何指派儲位、倉儲內分區、訂單如何分組、機器人與貨架的路徑規劃等問題，各個問題都會影響到整體揀貨效率。其中路徑規劃問題相當困難，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。本研究使用時空網路圖探討後續的路徑規劃問題，規劃一組避免碰撞的路徑使其能以最快時間完成所有訂單。

圖 1.3 揀貨工作站(Wurman et al., 2008)

1.2 研究問題之設定與目的

本研究探討的多機器人路徑規劃問題(Multi-Agent Path Finding, MAPF)，以往大多文獻皆假設貨架與機器人數量相同，每個貨架皆配置一台專屬的機器人，給定機器人與貨架的起訖點，目標找出一組無碰撞之路徑，以最小化該路徑之總移動距離，且大多文獻都只規劃貨架到揀貨工作站的路徑，沒有考慮到完成訂單後，貨架從揀貨工作站歸位至儲或區域的路徑。然而實際情況貨架個數遠多於機器人個數，而且若不考慮後續貨架歸位，將導致揀貨工作站周圍停滿貨架，造成交通阻塞。

在本研究中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、各訂單該由哪組貨架及機器人完成皆為已知的情況下，並放寬問題設定，貨架個數多於機器人個數，使其更符合現實情況，還考慮到完成訂單後，貨架與機器人從揀貨工作站，如何歸位的路徑規劃，以免揀貨工作站周圍充滿貨架。本研究處理的路徑規劃可細分三種路徑，第一種路徑是機器人到貨架的路徑，如圖 1.4 所示。第二種路徑是機器人抵達貨架後，將貨架抬起，機器人與貨架一起到揀貨工作站的路徑，如圖 1.5 所示。最後一種路徑是機器人與貨架抵達揀貨工作站並完成訂單後，貨架歸位的路徑，如圖 1.6 所示。完成任務後的機器人，便可移動其它貨架，質性下一個任務，如圖 1.7 所示。本研究提出嚴謹的數學模式以最小化訂單總等待時間為目標，找出一組無碰撞的路線。

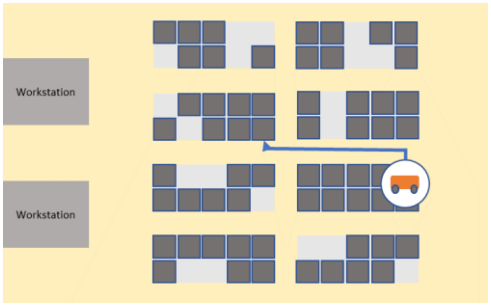

圖 1.4 機器人到貨架路線路線

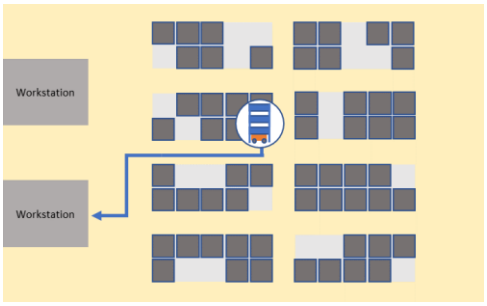

圖 1.5 貨架至工作站路線

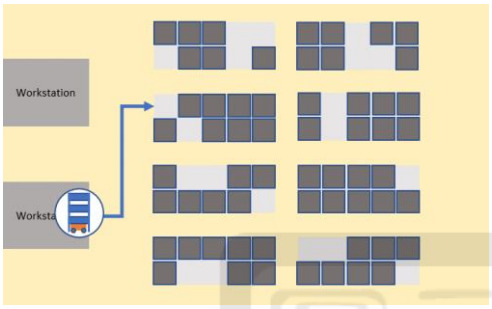

圖 1.6 貨架歸位路線

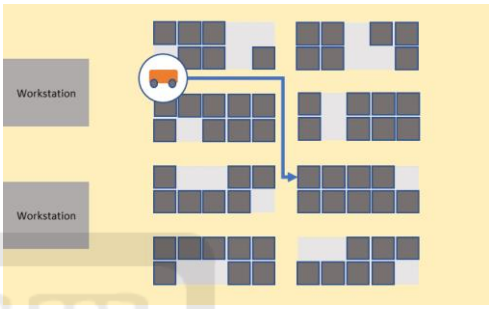

圖 1.7 機器人執行下個任務

1.3 研究範圍

本研究的研究範圍如下:

1. 每筆訂單皆已被指派必須由哪個揀貨工作站完成，且已知由哪組機器人與貨架服務訂單，目標找出一組完成所有訂單並無碰撞發生的路徑，最小化訂單總等待時間。
2. 開發貪婪演算法在短時間內得到可行解。
3. 使用模擬退火法迭代貪婪演算法，求得最佳訂單完成順序。

1.4 論文架構

本論文之架構如下：第二章文獻回顧，針對 MAPF 及其變形問題以及倉儲設定進行探討；第三章討論多機器人路徑規劃問題，此章會提出新的數學模式，最小化

總等待時間，找出一組無碰撞的路徑。第四章提出兩種貪婪演算法搭配模擬退火法，找出最佳訂單順序。第五章對數學模式及演算法進行數值測試及分析。第六章總結本論文，並列出未來研究方向建議。

第二章 文獻探討

本章分為三節，第一節為探討貨架選擇問題的相關文獻，第二節為 MAPF 問題與其它變形問題相關文獻，最後一節回顧其它文獻的問題設定，與本研究進行比較。

2.1 貨架選擇問題

Boysen et al. (2017) 針對單一揀貨工作站的訂單進行規劃，並指出訂單的揀貨順序與貨架來到順序需共同考量。給定每個貨架存放商品品項、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。該研究目標為最小化貨架交換次數，並加入時間的概念，提出一個數學模式，並將此問題更拆解為兩個相反的子問題，若已給定貨架的來到順序，該如何決定訂單的順序？若給定訂單的順序，該如何決定貨架的來到順序？針對此兩子問題，提出兩種的啟發式演算法進行求解。

Li et al. (2017)對單一揀貨工作站的訂單進行貨架選擇的最佳化，假設已知貨架位置、貨架上的商品品項以及訂單的商品需求數量，研究目標為最小化訂單之移動距離。該研究考慮訂單需求量與貨架商品存量，也將貨架與揀貨工作站的距離納入考量，提出一個數學模型，將貨架距離最為權重，也提出啟發式演算法進行求解。

Valle & Beasley. (2019)對單一揀貨工作站進行訂單指派與貨架選擇問題進行研究，假設不考慮貨架商品存量與訂單需求個數，將問題分成兩個部分，第一部分探討貨架與訂單的指派，各個工作站會被分配需處理的訂單，選擇貨架以滿足揀貨工作站的訂單需求；第二部分探討單一揀貨工作站的訂單揀貨順序與貨架來到順序。

2.2 多機器人路徑規劃(MAPF)與其它變形問題

MAPF 在人工智慧(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)等領域中皆被廣泛地研究(Ma et al., 2017)，MAPF 問題為：給定一個無向網路圖，每個機器人的起迄點皆為已知，規劃一組無碰撞的路線，使每個機器人抵達各自的迄點，完成任務。MAPF 問題常使用以下三種指標做為衡量(Yu & LaValle, 2013b)：各別任務完成時間之加總、最遲完工時刻、或機器人總移動距離。

Yu & LaValle (2013a)的研究中，建構出 MAPF 問題的數學模式，將原始的網路圖擴展為時空網路圖(time-expanded network)，如圖 2.1 所示，時空網路圖有利於描述有無碰撞的情形發生。該研究也提到，這個問題本質上是一個多元商品流量問題(Multi-commodity flow)，使用相似的概念來建立數學模式，將時空網路圖上的每條節線與每個節點之流量皆限制為 1 單位，即能避免節線或節點發生機器人碰撞。此研究使用數學模式來進行求解，後來相關的 MAPF 文獻中演算法也多與此研究進行比較。

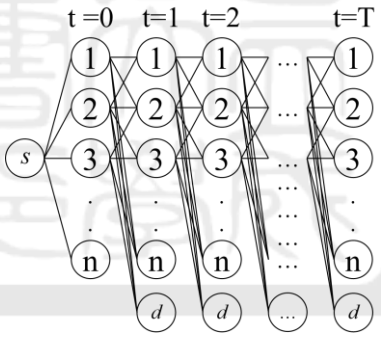

圖 2.1 時空網路圖

Surynek (2015)的研究中為了化簡時空網路圖，希望放寬對於最遲完工時刻的最佳化需求，其化簡之原因為該研究觀察到若是在某節點不太可能重複經過，則該節點就不需在時空網路圖上再擴展。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，來檢查是否有滿足 MAPF 的解存在。

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，其架構分為低層級搜尋(low-level search)以及高層級搜尋(high-level search)。其中，高層級搜尋是在一個衝突樹(Conflict Tree, CT)下執行限制動作的產生，不同的限制組合會在 CT 上產生一個新的節點，在 CT 下的每個節點包含了對於某機器人時間及地點的限制，舉例來說，在時刻 t 機器人 k_1 不能出現於節點 n，且在時刻 $t-1$ 機器人 k_2 不得出現於節點 n 等限制。低層級搜尋則針對某個在 CT 節點上的限制來針對各機器人產生其最短路徑，若是低層級搜尋結束後還是有發生碰撞，則高層級搜尋便會在 CT 上產生一個新的節點。由於一旦有碰撞的發生即要分支產生新節點，導致運算時間的大量增加。

Hönig, Kiesel, Tinka, Durham, & Ayanian (2018)將 MAPF 問題加入任務指派(Task Assignment, TA)的設定，假設機器人的起點一樣皆為已知，但各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就算任務完成。該研究改良 CBS 演算法為 CBS-TA 演算法，透過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)，只在有需求時才新增根節點。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。

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。

王宗翰(2020)將貨架與機器人視為獨立的兩種不同個體，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，探討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，該研究為貨架選擇與後續路徑規劃的整合性問題，使用層空網路圖的架構，

建立一個整數規劃模式，也提出兩個演算法進行求解。

表 2.1 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
王宗翰(2020)		V	V	V
本研究		V	V	V

本研究主要參考王宗翰(2020)之設定，已知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，該研究與本研究相異之處在於本研究著重處理多機器人路徑規劃，在已知每筆訂單由哪組機器人與貨架服務的情況下，規劃出一組無碰撞的路徑。

2.3 移動式倉儲其它相關文獻

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，從現實應用的層面我們認為更為一般性的研究有其必要。下一章本研究將針對多機器人路徑規劃的問題，提出其對應的數學模式。

第三章 多台智慧倉儲機器人之最佳揀貨路徑規劃模式

本章探討多台智慧倉儲機器人之最佳揀貨路徑規劃問題，3.1 節將描述研究問題。3.2 節及 3.3 節將詳細說明本研究之假設與網路架構，為了方便數學模式的建立，我們將原始問題之網路圖架構轉成以時空網路為架構的整數規劃模式，考量機器人的路徑規劃，提出一個整數規劃模式。3.4 節將說明模式之參數、變數、目標式及限制式。

3.1 問題描述

假設有一無向網路圖 $G=(N,A)$，N 為所有節點集合，A 為所有節線的集合，在無向網路圖上存在 k 台機器人與 r 個裝載多種產品的貨架，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，且已知由哪組機器人與貨架服務訂單，目標找出一組完成所有訂單並無碰撞發生的路徑，最小化訂單最總等待時間。與以往 MAPF 問題不同，本研究中貨架數量多於機器人數量，貨架需要透過機器人來移動，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。

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. 每筆訂單上只需一種產品，並只考慮種類需求不考慮數量。
2. 每筆訂單皆已被指派必須由哪個揀貨工作站完成。
3. 每筆訂單已知由哪組機器人與貨架服務。
4. 相同揀貨工作站之訂單可同時處理，並無優先順序。
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，不考慮數量。
6. 貨架的任務完成後，須待在指定的儲存區域。

7. 貨架被機器人自底部頂起之動作不耗時，機器人轉向之動作亦不耗時。

8. 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。

3.3 網路架構

在可移動式貨架倉儲系統中，讀取地面上的條碼辨認方向進行移動，所以每個條碼可被視為一個節點，任一個節點可沿四個方向進行移動，由於對角移動(圖3.1)會導致每個移動動作耗時不一致，且認定碰撞是否發生更為複雜，因此在本研究中並不建立對角節線，只考慮縱橫雙向的移動。

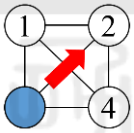

圖 3.1 對角移動

本研究所使用的倉儲佈置為網格網路圖(grid network)，參考 Yu & LaValle (2013a) 之作法，將時間的概念加入網路圖裡，使用時空網路架構能使數學模式易於表達碰撞的情形。如此便有利於描述停留原位可能遭遇碰撞的情況。此外，本時空網路圖只有設立虛擬起點 s，並無設立虛擬訖點，如圖 2.1 所示，這是因為我們的貨架皆可被重複使用的，若將其連入虛擬訖點，則會造成其實際上存在，但在網路圖消失的情況，如此一來可能使數學模式產生的解，實際上會與已連入虛擬訖點的機器人或貨架發生碰撞。

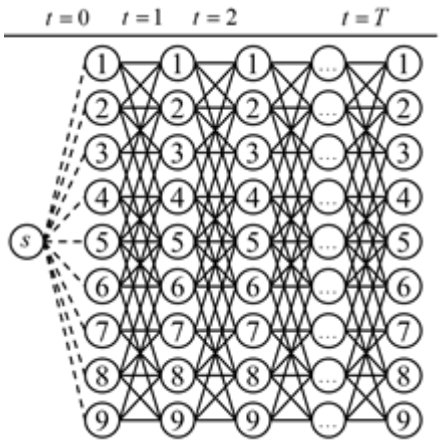

圖 3.2 時空網路圖

3.4 數學模式

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　貨架儲存區域之集合，$N_s \subseteq N$

N_p　揀貨工作站之集合，$N_p \subseteq N$

A　所有節線之集合

K　機器人之集合

R　貨架之集合

R_k　由機器人 k 進行移動的貨架之集合

O　訂單之集合

參數

T　最大單位時間數

M　極大的數

ε　極小的數

k_o^{Order}　完成訂單o的機器人，$k_o^{Order} \in K$

k_r^{Rack}　移動貨架r的機器人，$k_r^{Rack} \in K$

r_o^{Order}　完成訂單o的貨架，$r_o^{Order} \in K$

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，$s_r^{Rack} \in N$

d_o　訂單$o \in O$之揀貨工作站，$d_o \in N_p$

變數

x_{ijk}^t　機器人$k \in K$在時刻t經過節線$(i,j) \in A$為1；反之為0

u_j^{rt}　貨架$r \in R$在時刻t在停放於節點$j \in N$為1；反之為0

$y_{jk_r^{Rack}}^{rt}$　貨架$r \in R$在時刻t結尾於節點$j \in N$被機器人k_r^{Rack}抬起為1；反之為0

$z_{jk_r^{Rack}}^{rt}$　貨架$r \in R$在時刻t結尾於節點$j \in N$被機器人k_r^{Rack}放下為1；反之為0

$v_{ok_o^{Order}}^t$　訂單$o \in O$在時刻t結尾由機器人k_o^{Order}完成訂單為1；反之為0

3.4.2目標式

此類問題最常使用三種指標，分別為最小化「最遲完工時刻」，如式(3.4.1)，「總等待時間」，如式(3.4.2)以及「機器人總移動距離」，如式(3.4.3)。

$$Minimize \quad w \tag{3.4.1}$$

$$Minimize \quad \sum_{t=0}^{T}\sum_{o\in O} tv_{ok_o^{Order}}^{t} \tag{3.4.2}$$

$$Minimize \quad \sum_{(i,j)\in A, i\neq j}\sum_{k\in K}\sum_{t=0}^{T} x_{ijk}^{t} \tag{3.4.3}$$

本研究的目標式則採用最小化總等待時間，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，理論上並不希望機器人做多餘的移動，因此我們給予機器人移動一個很小的懲罰值ε，避免無謂的移動，因此可將式(3.4.2)結合式(3.4.3)修改成式(3.4.4)。

$$Minimize \quad \sum_{t=0}^{T}\sum_{o\in O}(tv_{ok_o^{Order}}^{t}) + \varepsilon\left(\sum_{(i,j)\in A, i\neq j}\sum_{k\in K}\sum_{t=0}^{T} x_{ijk}^{t}\right) \tag{3.4.4}$$

3.4.3限制式

本研究考量機器人之路徑規劃，機器人會移動至貨架所在節點，抬起貨架前往揀貨工作站，完成訂單後載將貨架移動至指定儲存區域，直到所有訂單皆被完成。

限制式(3.4.5)為每台機器人之流量守恆限制式。

$$\sum_{i:(i,j)\in A} x_{ijk}^{t} = \sum_{i:(j,i)\in A} x_{jik}^{t+1} \quad \forall j\in N, k\in K, 0\le t<T \tag{3.4.5}$$

限制式(3.4.6)為每個貨架之流量限制式。

$$u_j^{rt+1} = u_j^{rt} - y_{jk_r^{Rack}}^{rt+1} + z_{jk_r^{Rack}}^{rt+1} \quad \forall j\in N, r\in R, 0\le t<T \tag{3.4.6}$$

限制式(3.4.7)為機器人由虛擬起點至初始位置。

$$x_{0S_k^{Robot}k}^{0} = 1 \quad \forall k\in K \tag{3.4.7}$$

限制式(3.4.8)為貨架之初始位置。

$$u_{S_r^{Rack}}^{r0} = 1 \quad \forall r \in R \tag{3.4.8}$$

限制式(3.4.9)為節點之容量限制，其目的在於避免兩機器人在同一時刻移動至相同節點。

$$\sum_{i:(i,j)\in A} \sum_{k\in K} x_{ijk}^{t} \le 1 \quad \forall j \in N, 0 \le t < T \tag{3.4.9}$$

限制式(3.4.10)為節線之容量限制，其目的在於避免兩機器人在同一階層於同一節線上同向或反向移動。

$$\sum_{k\in K} x_{ijk}^{t} + \sum_{k\in K} x_{jik}^{t} \le 1 \quad \forall (i,j),(j,i) \in A, i \ne j, 0 \le t \le T \tag{3.4.10}$$

限制式(3.4.11)確保在任一個時刻上，任一機器人只能且必須存在於唯一的某一個節線上。

$$\sum_{(i,j)\in A} x_{ijk}^{t} = 1 \quad \forall k \in K, 0 \le t \le T \tag{3.4.11}$$

限制式(3.4.12)確保在任一個時刻上，任一節點最多停留一台貨架。

$$\sum_{r\in R} u_j^{rt} \le 1 \quad \forall j \in N, 0 \le t \le T \tag{3.4.12}$$

限制式(3.4.13)確保在任一個時刻，任一貨架最多停留在某一個節點上。

$$\sum_{j\in N} u_j^{rt} \le 1 \quad \forall r \in R, 0 \le t < T \tag{3.4.13}$$

限制式(3.4.14)限制貨架最終得回到指定儲存區域。

$$\sum_{j\in N_s} u_j^{rT} = 1 \quad \forall r \in R \tag{3.4.14}$$

限制式(3.4.15)限制了載有貨架的機器人不得移動至有貨架停留的節點。

$$\sum_{i:(i,j)\in A} x_{ijk}^{t} + \sum_{r\in R_k} \sum_{i\in N} \sum_{t'=0}^{t-1} \left(y_{ik}^{rt'} - z_{ik}^{rt'} \right) \le 2 - \sum_{r\in R} u_j^{rt-1} \quad \forall k \in K, j \in N, 0 < t \le T \tag{3.4.15}$$

限制式(3.4.16)確保在任一個時刻，任一節點，若沒有機器人抵達不得執行抬起貨架與放下貨架的動作。

$$\sum_{r\in R_k}\sum_{i\in N}\left(y^{rt}_{jk_r^{Rack}}+z^{rt}_{jk_r^{Rack}}\right)\le\sum_{i:(i,j)\in A}x^{t}_{ijk_r^{Rack}}\quad\forall j\in N,k\in K,0\le t<T \tag{3.4.16}$$

限制式(3.4.17)確保在任一個時刻，任一機器人最多搬運一台貨架。

$$\sum_{j\in N}\sum_{r\in R_k}\sum_{t'=0}^{t-1}\left(y^{rt'}_{jk}-z^{rt'}_{jk}\right)\le 1\quad\forall k\in K,0<t\le T \tag{3.4.17}$$

限制式(3.4.18)確限制 t=0 時，不能執行抬起貨架的動作。

$$y^{r0}_{jk_r^{Rack}}=0\quad\forall j\in N,r\in R \tag{3.4.18}$$

限制式(3.4.19)限制 t=0 時，不能執行放下貨架的動作。

$$z^{r0}_{jk_r^{Rack}}=0\quad\forall j\in N,r\in R \tag{3.4.19}$$

限制是(3.4.20)為確保前一時刻節點有貨架停放，機器人才能執行抬起貨架的動作。

$$y^{rt}_{jk_r^{Rack}}\le u^{rt-1}_{j}\quad\forall j\in N,r\in R,1\le t\le T \tag{3.4.20}$$

限制是(3.4.21)為確保前一時刻節點無貨架停放，機器人才能執行放下貨架的動作。

$$z^{rt}_{jk_r^{Rack}}\le 1-\sum_{r'\in R}u^{r't-1}_{j}\quad\forall j\in N,r\in R,1\le t\le T \tag{3.4.21}$$

限制是(3.4.22)為確保機器人承載貨架，才能執行放下貨架的動作。

$$z^{rt}_{jk_r^{Rack}}\le\sum_{j\in N}\sum_{t'=0}^{t-1}\left(y^{rt'}_{jk_r^{Rack}}-z^{rt'}_{jk_r^{Rack}}\right)\quad\forall j\in N,r\in R,1\le t\le T \tag{3.4.22}$$

限制式(3.4.23)及限制式(3.4.24)，確保訂單完成需有機器人抵達工作站的節點，並且載有能完成訂單的貨架這兩個條件皆滿足，才算訂單完成。

$$v^{t}_{ok_o^{Order}}\le\sum_{i:(i,d_o)}x^{t}_{id_ok_o^{Order}}\quad\forall o\in O,0\le t\le T \tag{3.4.23}$$

$$v^{t}_{ok_o^{Order}}\le\sum_{j\in N}\sum_{t'=0}^{t-1}\left(y^{r_r^{Order}t'}_{jk_o^{Order}}-z^{r_o^{Order}t'}_{jk_o^{Order}}\right)\quad\forall o\in O,0\le t\le T \tag{3.4.24}$$

限制式(3.4.25)確保所有訂單皆要被完成。

$$\sum_{t=0}^{T}v^{t}_{ok_o^{Order}}=1\quad\forall o\in O \tag{3.4.25}$$

限制式(3.4.26)至限制式(3.4.30)則限制了變數的範圍。

$$x^{t}_{ijk}\in\{0,1\}\quad\forall(i,j)\in A,k\in K,0\le t\le T \tag{3.4.26}$$

$$u_j^{rt} \in \{0,1\} \quad \forall j \in N, r \in R, 0 \le t \le T \tag{3.4.27}$$

$$y_{jk_r^{Rack}}^{rt}, z_{jk_r^{Rack}}^{rt} \in \{0,1\} \quad \forall j \in N, r \in R, 0 \le t \le T \tag{3.4.28}$$

$$v_{ok_o^{Order}}^{t} \in \{0,1\} \quad \forall o \in O, 0 \le t \le T \tag{3.4.29}$$

3.4.4 變數與限制式個數估算

本項進行本研究的數學模式變數與限制式個數估算，並與王宗瀚(2020)的數學模式做比較，估算的例子為 5X5 的網格圖，共有 80 條節線、8 個貨架及四台機器人，需處理 8 筆訂單，時空網路圖時長設定 20，王宗瀚(2020)使用層空網路圖，使用 L 為其網路圖層數，與本研究中的網路時長 T 相同概念，之為的相關計算接使用 T 來代表 L，參數相關設定如表 3.1 所示。

表 3.1 估算變數與限制式個數之參數設定值

節點個數	貨架	機器人	網路時長	訂單	節線數量
$\|N\|$	$\|R\|$	$\|K\|$	$\|T\|(L)$	$\|O\|$	$\|A\|$
25	10	4	25	8	80

表 3.2 變數總個數比較

本研究			王宗瀚(2020)		
Variable	**Calculation**	**Number**	**Variable**	**Calculation**	**Number**
x_{ijk}^{t}	$\|A\|\|K\|\|T\|\|$	8000	x_{ij}^{kl}	$\|A\|\|K\|\|T\|$	8000
u_j^{rt}	$\|N\|\|R\|T\|$	6250	y_{ij}^{rl}	$\|A\|\|R\|\|T\|$	20000
$y_{jk_r^{Rack}}^{rt}$	$\|N\|\|R\|\|T\|$	635	z_o^l	$\|O\|\|T\|$	200
$z_{jk_r^{Rack}}^{rt}$	$\|N\|\|R\|\|T\|$	6250	w		1
$v_{ok_o^{Order}}^{t}$	$\|O\|\|T\|$	200	--	--	--
變數總個數		21335	**變數總個數**		28201

本研究的數學模式將貨架的行動簡化為貨架被機器人抬起與放下，由求解結果

進行反推貨架的路徑，不需要新增多餘變數，可縮小求解問題的規模。由表 3.2 及表 3.3 的計算結果顯示本研究所提出的數學模式確實能減少變數與限制式的個數，進而改善求解效率。

表 3.3 限制式總個數比較

本研究			**王宗瀚(2020)**		
Constraint	**Calculation**	**Number**	**Constraint**	**Calculation**	**Number**
3.4.5	$\|N\|\|K\|\|T\|$	1000	3.4.5	$\|N\|\|K\|\|T\|$	1000
3.4.6	$\|N\|\|R\|\|T\|$	5000	3.4.6	$\|N\|\|R\|\|T\|$	5000
3.4.7	$\|K\|\|T\|$	2	3.4.7	$\|K\|$	2
3.4.8	$\|R\|$	10	3.4.8	$\|R\|$	10
3.4.9	$\|N\|\|T\|$	500	3.4.9	$\|N\|\|T\|$	500
3.4.10	$\|A\|\|T\|$	1600	3.4.10	$\|N\|\|T\|$	500
3.4.11	$\|K\|\|T\|$	40	3.4.11	$\|A\|$	1600
3.4.12	$\|N\|\|T\|$	500	3.4.12	$\|A\|\|T\|$	1600
3.4.13	$\|R\|\|T\|$	200	3.4.13	$\|K\|\|T\|$	40
3.4.14	$\|R\|$	10	3.4.14	$\|R\|\|T\|$	200
3.4.15	$\|N\|\|K\|\|T\|$	1000	3.4.15	$\|A\|\|R\|\|T\|$	16000
3.4.16	$\|N\|\|R\|\|T\|$	1000	3.4.16	$\|O\|$	8
3.4.17	$\|K\|\|T\|$	40	3.4.17	$\|O\|\|T\|$	160
3.4.18	$\|N\|\|R\|\|T\|$	250	3.4.18	$\|O\|\|T\|$	160
3.4.19	$\|N\|\|R\|$	250	3.4.19	$\|O\|$	8
3.4.20	$\|N\|\|R\|\|T\|$	250	3.4.20	$\|R\|$	10
3.4.21	$\|N\|\|R\|\|T\|$	5000	--	--	--
3.4.22	$\|N\|\|R\|\|T\|$	5000	--	--	--
3.4.23	$\|O\|\|T\|$	160	--	--	--
3.4.24	$\|O\|\|T\|$	160	--	--	--
3.4.25	$\|O\|$	8	--	--	--
限制式總個數		21980	**限制式總個數**		26798

3.5 小結

本章使用時空網路圖的概念，針對多機器人路徑規劃，提出一個新的數學模

式，由於使用時空網路圖建模，若是遇到中大型網路圖，求解時間過久，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 多台智慧倉儲機器人之最佳揀貨路徑規劃演算法

第三章使用之數學模式，在面對較大規模之問題，數學模式需耗時甚久進行求解，在實務上較不可行。因此我們希望設計演算法能在短時間內獲得可行解，本章將會提出兩種貪婪演算法，並使用模擬退火法(Simulated Annealing)迭代求解出最佳的訂單完成順序。

4.1 貪婪演算法

在本研究問題設定下，要如何規劃好數條路徑，以避免貨架及機器人之間發生碰撞非常重要。因為貨架與機器人會隨著時間推移，可能位置會有變動，若每條路徑在原始網路圖上同時使用最短路徑進行規劃，當碰撞發生時才對路徑來進行修正的話，容易於其它節點再發生碰撞，使得難以尋找到可行解。針對此路徑規劃之困難，我們使用時空網路圖進行路徑規劃。

4.1.1 批次處理訂單之貪婪演算法 $Greedy_{order}$

$Greedy_{order}$ 每次處理一筆訂單，在選定的該訂單，規劃三條路徑: (1)機器人到貨架的路徑。(2)機器人與貨架一起到揀貨工作站的路徑。(3)貨架歸位的路徑。每條路徑選擇當下最快可抵達又無發生碰撞，決定好路徑後更新時空網路圖節點與節線的資訊，避免之後路徑發生碰撞的問題，批次處理訂單之貪婪演算法流程如表 4.1 所示。

首先設定時空網路圖時長為 T，使用 *status* 記錄該訂單的路徑規劃狀態，每筆訂單的 *status* 初始值為 0。貪婪演算法一次處理一筆訂單，透過 *path_planning* 函式規劃三條路徑，若三條路徑皆找到可行路徑，*path_planning* 函式的回傳值為 3，接著處理下一筆訂單；反之，若三條路徑無法皆找到可行路徑，*path_planning* 函式回傳已完成的路徑個數，擴展時空網路圖時長，每次增加 5 單位時間，並再執行一次

path_planning 函式，規劃剩餘路徑，直到皆完成該訂單的三條路徑，才處理下一筆訂單，*path_planning* 函式如表 4.2 所示。

表 4.1 批次處理訂單之貪婪演算法流程

Greedy_order Algorithm

Data: *O,T*

Function *Greedy_order(O)*

1. ***For*** *o in O* ***do***
2. *status = 0*
3. *status = path_planning(o, status)*
4. ***While*** *status != 3* ***do***
5. *T = T+5*
6. *status = path_planning(o, status)*
7. *compute total_waiting*
8. ***return*** *total_waiting*

表 4.2 訂單路徑規劃函式

path_planning

Data: *RO,KO,dest*

1. ***Function*** *path_planning(o, status)*
2. *r,k,d = RO[o] ,KO[o], dest[o]*
3. *pr,pk = pos_r[r],pos_k[k]*
4. ***If*** *status = 0* ***then***
5. ***If*** *shortest_path(k,r,pk,pr,status)* ***then***
6. *status = 1*
7. ***If*** *status = 1* ***then***
8. ***If*** *shortest_path(k,r,pr,d,status)* ***then***
9. *status = 2*
10. ***If*** *status = 2* ***then***
11. *t = closest empty storage region*
12. ***If*** *shortest_path(k,r,t,status)* ***then***
13. *status = 3*
14. ***return*** *status*
15. ***end function***

每筆訂單皆需要規劃三條路徑，因此需執行三次 *shortest_path* 函式，當規劃完一條路徑，對訂單的路徑規劃狀態進行更新。第一條路徑起點為機器人位置，迄點為貨架位置；第二條路徑起點為貨架位置，迄點則為完成訂單的工作站；第三條路徑的起點為完成訂單的工作站，迄點為離工作站最近無貨架停放的儲存區域。*shortest_path* 函式如表 4.3 所示。

表 4.3 最短路徑函式

shortest_path

```
Data: N, pos_k, pos_r
1.  Function shortest_path(k,r,s,t,status)
2.      KP = {}
3.      For j in N do
4.          j.d = INFINITE
5.      s.d, s->pred = 0, NULL
6.      create heap H
7.      insert s into H
8.      While H is not empty do
9.          i = find_min(H)
10.         delete i from H
11.         current = i
12.         While current != NULL do
13.             current = current->next
14.             If status == 0 and current.state == 1 then
15.                 Continue
16.             If status != 0 and current.state != 0 then
17.                 Continue
18.             If current.Node == t.Node then
19.                 While current != NULL do
20.                     If status == 0 then
21.                         current.state = 1
22.                     Else then
23.                         current.state = 3
24.                     KP.add(current)
25.                     current = current->pred
26.                 pos_k[k], pos_r[r] = t, t
```

```
27.            return 1
28.         If j.d > temp then
29.            If j.d == INFINITE then
30.               temp = i.d + 1
31.               insert j into H
32.               j->pred = i
33.    return 0
Result: KP
```

4.1.2 批次處理任務之貪婪演算法 $Greedy_{task}$

$Greedy_{task}$ 將任務分成三種(1)規劃機器人到貨架的路徑。(2)規劃機器人與貨架一起到揀貨工作站的路徑。(3)規劃貨架歸位的路徑。整理每台機器人的任務序列，若機器人任務序列中有多筆訂單的所需貨架相同，優先處理這些訂單，透過更改排序靠後訂單的順序，移至排序靠前的訂單下一順位，並移除兩項任務(1)排序靠前訂單的任務三:規劃貨架歸位的路徑。(2)排序靠後的訂單的任務一: 規劃機器人到貨架的路徑。移除這兩項任務後，機器人與貨架就從目前工作站移至下一訂單的工作站。任務序列整理避免相同貨架頻繁進出貨架儲存區域，減少不必要的移動，加快後續任務進行，如圖 4.1 及圖 4.2 所示。

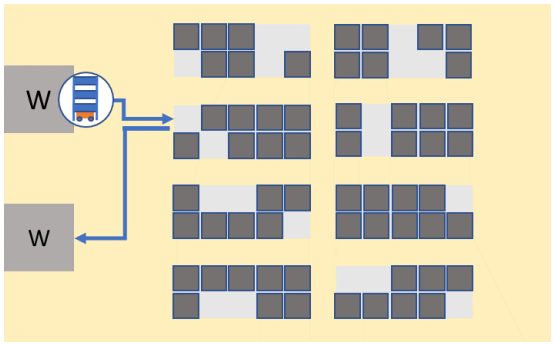

圖 4.1 序列整理前的路徑規劃

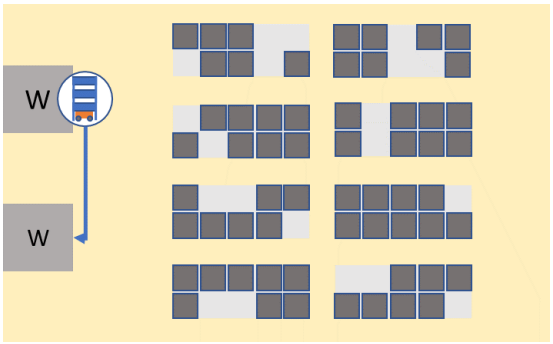

圖 4.2 序列整理後的路徑規劃

完成任務序列整理後，再來估算每項任務的完成時刻，依照任務完成時刻進行

排序，愈早完成的任務有較高的規劃路徑優先順序，所有任務估算方式為不考慮碰撞的最短路徑。根據路徑優先順序，每次處理一項任務，各項任務選擇當下最快可抵達又無發生碰撞的路徑，若任務無法規劃出路徑，擴展時空網路圖時長，每次增加 5 單位時間，並再執行一次 *shortest_path* 函式，規劃路徑，$Greedy_{task}$ 如表 4.4 所示。

表 4.4 批次處理任務之貪婪演算法流程

Greedy_task Algorithm
Data: *O,T*
Function *Greedy_task(O)*
1. arrange task sequence TS
*2. **For** task in TS **do***
3. s, t= task.start, task.to
4. k, r = task.robot, task.rack
5. status = task.status
6. FLAG = shortest_path(k,r,s,t,status)
*7. **while** FLAG == 0 **do***
8. T = T+5
9. FLAG = shortest_path(k,r,s,t,status)
10. compute total_waiting
*11. **return** total_waiting*

4.2 模擬退火法

模擬退火法是模擬冷卻晶體的過程，在熱力學上，退火現象為物體逐漸冷卻的物理現象，當溫度愈低，物體的能量狀態會降低，當溫度夠低時會開始結晶，在緩慢的降溫冷卻下，能形成結構整齊的晶體。模擬退火法（Simulated Annealing）是 Kirkpatrick, Gelatt and Vecchi (1983)所提出，將組合最佳化問題尋求最低成本的過程，類比為物體尋求最低能量，用來解決組合最佳化問題。

$Greedy_{order}$ 是每次處理一筆訂單，不得發生碰撞為前提，選擇當下能完成訂單

的機器人與貨架最短路徑，不同訂單順序會影響各訂單所選擇的機器人與貨架，規劃出不同的路徑。$Greedy_{task}$ 是依機器人任務序列的估算完成時刻先後，進行路徑規劃，不同訂單順序會影響到各機器人的任務序列，同樣會規劃出不同的路徑。由此可知訂單的順序會影響最終目標值，本節使用模擬退火法(Simulated Annealing)，透過改變訂單順序，來迭代求解，模擬退火法流程如圖 4.3 所示，虛擬碼如表 4.5 所示。

模擬退火法首先給定一初始解 S 與初始溫度 T，由初始解產生鄰近新解 S_{new}，計算當前解與新解的目標值差(ΔObj)，若新解的目標值小於目前解的目標值($\Delta Obj < 0$)，代表新解的訂單總等待時間較小，此時接受新解 S_{new} 作為當前解 S，並更新當前目標值；若新解的目標值不小於目前解的目標值($\Delta Obj \geq 0$)，代表新解的訂單總等待時間較大，此時有一定機率會接受新解，機率的計算方式如式 4.2.1:

$$E = \exp\left(\frac{-\Delta Obj}{T}\right) \tag{4.2.1}$$

並產生一亂數值 R，若 $R>E$，則接受新解作為當前解，更新當前目標值。迭代次數尚未完成且溫度高於終止溫度，需進行降溫。重複上述流程直到完成迭代次數或達到終止溫度。

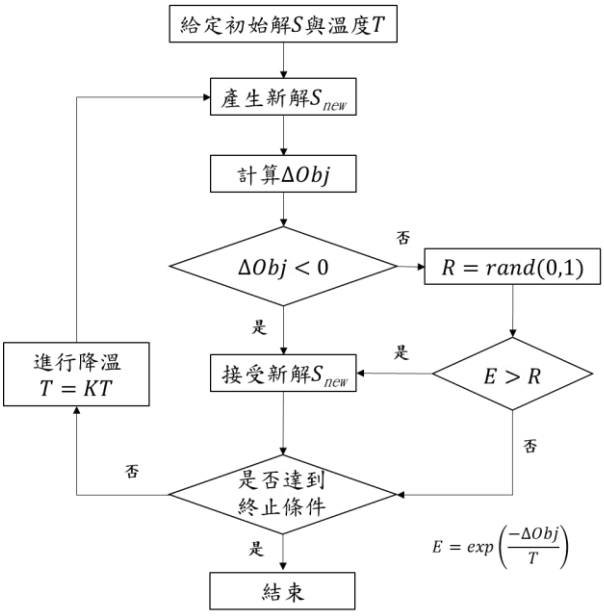

圖 4.3 模擬退火法流程圖

表 4.5 模擬退火法虛擬碼

Simulated Annealing

Data: *S,T*

Function *SA(S, T, K, max_iteration, Tmin)*

Obj, flag, iteration = Greedy(S), 1, 0
While *flag* ***do***
    Snew=neighbor(S)
    Obj_new=Greedy(Snew)
    delta = Obj_new - Obj
    If *delta < 0* ***then***
        S, Obj = Snew, Obj_new
    Else then
        R = rand(0,1)
        E = exp(-delta / T)
        If *R > E* ***then***
            S, Obj = Snew, Obj_new
    T = KT
    If *iteration >=max_iteration or T<Tmin* ***then***
        flag = 0
return *Obj*

如何尋找鄰近新解會影響是否能跳脫出區域最佳解，本研究使用模擬退火法期望尋找到最佳訂單優先順序，鄰近新解的產生方式為隨機挑選兩訂單(O_k 、O_m)，將排序靠前的訂單(O_k)，移至排序靠後的訂單(O_m)下一順位，如下圖 4.4 新訂單順序產生示意圖所示。

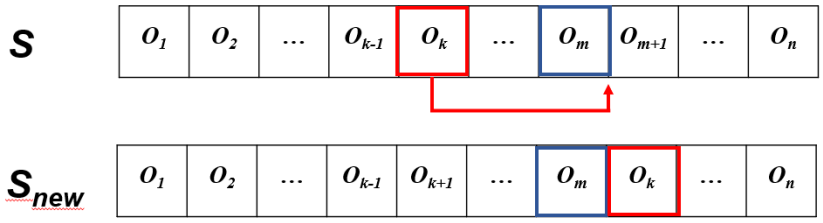

圖 4.4 新訂單順序產生示意圖

模擬退火法的參數設定會影響到演算法收斂速度，其中三個參數影響甚鉅，分別為(1)初始溫度(2)降溫速率(3)終止溫度，雖然初始溫度高，演算法要收斂需要耗費的時間就越長，但根據式 4.2.1，溫度越高，有更高機率接受較差的解，跳脫出區域最佳解，搜索到全域最佳解的機會就越高，隨著溫度逐漸下降，接受較差解的機率也逐漸縮小。

4.3 小結

本章介紹了兩種貪婪演算法以加速求解，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，但本研究的貪婪演算法並無法考量到，但由於求解速度快上很多，因此在實際應用上較為可行。因為我們設計的貪婪演算法深受訂單順序影響，我們使用模擬退火法去收斂訂單優先順序，期望求得最佳訂單優先順序以改善求解品質。

第五章 數值測試與分析

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與分析，包括求解大小、速度及品質進行比較。本研究測試環境為 Windows 10 作業系統，搭配 Intel Core i9-10900，2.80GHZ 處理器，與 8G 記憶體。演算法以 C++為程式語言撰寫，數學模式利用 Gurobi 9.1 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，因此節點個數n皆為平方數，儲存區域皆相隔一排走道，工作站位於最下方一排，考量實際情形，設定貨架個數$R \geq$機器人個數K，求解時長給定基本值 25，數學模式若因為時長不足導致無解，每次增加 5 單位時長，再進行求解。

表 5.1 參數設定

參數設定					
節點數	機器人數	貨架個數	求解時長	訂單數量	產品種類
n	K	R	T	O	P

5.2 數學模式求解結果

本小節針對第三章所提出數學模式進行測試，表 5.2 為數學模式在不同網路圖及參數設定下的求解表現，求解時限為 3600 秒。由表 5.2 可看出當訂單數量多以及機器人數量少時，CPU Time 較高，有些測資在 3600 秒內還找不到可行解，推斷原因為數學模式是基於時空網路圖為架構進行建模，時空網路圖的時長影響數學模式的求解速度甚鉅，一旦訂單數量多以及機器人數量少，每位機器人所要負責的訂單就多，時空網路的時長就得增加，造成求解時間大幅增加，甚至在求解時限內無

法獲得可行解。

表 5.2 數學模式測試

n	*O*	*R*	*K*	*P*	*CPU Time(s)*	*Obj*	*Status*
16	4	4	2	8	52.35	44	*Optimal*
16	4	4	2	10	6.42	20	*Optimal*
16	4	4	4	8	7.87	31	*Optimal*
16	4	4	4	10	7.91	24	*Optimal*
16	8	4	2	8	37.64	51	*Optimal*
16	8	4	2	10	306.56	82	*Optimal*
16	8	4	4	8	13.71	52	*Optimal*
16	8	4	4	10	32.45	62	*Optimal*
25	4	4	2	8	124.38	45	*Optimal*
25	4	4	2	10	906.39	46	*Optimal*
25	4	4	4	8	120.33	36	*Optimal*
25	4	4	4	10	22.62	23	*Optimal*
25	8	4	2	8	3600.37	--	*Time_limit*
25	8	4	2	10	3600.11	--	*Time_limit*
25	8	4	4	8	653.64	70	*Optimal*
25	8	4	4	10	138.94	67	*Optimal*
36	4	8	4	8	455.90	46	*Optimal*
36	8	8	2	8	3600.07	--	*Time_limit*
36	8	8	4	8	3600.11	--	*Time_limit*

5.3 數學模式與貪婪演算法之比較

本節為數學模式 M 與兩種貪婪演算法 $Greedy_{order}$ 、$Greedy_{task}$ 之比較，只選擇數學模式在時間限制內有解的測資，進行比較。從表 5.3 可看出，兩種貪婪演算法的運行速度比起數學模式快上許多。在求解品質上，因為貪婪演算法 $Greedy_{order}$ 設計每筆訂單完成後，貨架必須回到貨架儲存區域，有些貨架需負責多筆訂單，因此會往返貨架儲存區域，花費更多時間才能完成其負責訂單，$Greedy_{task}$ 則是有任務序列整理，避免掉有貨架重複往返儲存區域的情況，且因為 $Greedy_{order}$ 一次得規劃三條路徑，規劃後不得更改，比起 $Greedy_{task}$ 一次規劃一項任務，$Greedy_{order}$ 較無彈性，從測資的結果來對比，的確 $Greedy_{task}$ 求解品質較佳。

在模擬退火法的部分，初始溫度 T 為 1000，降溫速率 K 為 0.99，當訂單數量為 4 時，迭代次數為 200 次；當訂單數量為 8 時，迭代次數為 10000 次。由表 5.4 可看出使用模擬退火法每筆測資都能收斂到更佳的解，且不需耗費大量時間即可收

斂，但在訂單數量多以及機器人數量少的測資，相較其它筆測資，改善幅度較小，可能原因為機器人個數少，機器人在網路圖上移動的選擇較多，訂單優先順序影響目標值的程度較小。

表 5.3 數學模式與兩種貪婪演算法比較

n	***O***	***R***	***K***	***P***	***M****		$Greedy_{order}$		$Greedy_{task}$	
					CPU Time(s)	***Obj***	***CPU Time(s)***	***Gap (%)***	***CPU Time(s)***	***Gap (%)***
16	4	4	2	8	52.35	**44**	0.0001	36.36	0.00020	13.64
16	4	4	2	10	6.42	**20**	0.00012	35	0.00024	10.00
16	4	4	4	8	7.87	**31**	0.00013	29.03	0.00014	9.68
16	4	4	4	10	7.91	**24**	0.00014	20.83	0.00018	8.33
16	8	4	2	8	37.64	**51**	0.00035	39.22	0.00053	15.69
16	8	4	2	10	306.56	**82**	0.00034	42.68	0.00038	14.63
16	8	4	4	8	13.71	**52**	0.00031	38.46	0.00064	13.46
16	8	4	4	10	32.45	**60**	0.00032	26.67	0.00044	13.33
25	4	4	2	8	124.38	**45**	0.00022	42.22	0.00044	15.56
25	4	4	2	10	906.39	**46**	0.00023	32.61	0.00039	17.39
25	4	4	4	8	120.33	**36**	0.00025	22.22	0.00039	13.89
25	4	4	4	10	22.62	**23**	0.00027	30.43	0.00049	13.04
25	8	4	4	8	653.64	**70**	0.00066	25.71	0.00109	8.57
25	8	4	4	10	138.94	**67**	0.00061	35.82	0.00103	10.45
36	4	8	4	8	455.9	**46**	0.00063	36.96	0.00113	23.91

表 5.4 數學模式與模擬退火法比較

n	***O***	***R***	***K***	***P***	***M****		SA_{order}		SA_{task}	
					CPU Time(s)	***Obj***	***CPU Time(s)***	***Gap (%)***	***CPU Time(s)***	***Gap (%)***
16	4	4	2	8	52.35	**44**	0.18	18.18	0.22	13.64
16	4	4	2	10	6.42	**20**	0.13	15.00	0.15	10.00
16	4	4	4	8	7.87	**31**	0.16	12.90	0.25	6.45
16	4	4	4	10	7.91	**24**	0.20	12.50	0.29	8.33
16	8	4	2	8	37.64	**51**	7.77	15.69	8.41	11.76
16	8	4	2	10	306.56	**82**	7.00	26.83	7.01	8.54
16	8	4	4	8	13.71	**52**	6.94	17.31	9.51	5.77
16	8	4	4	10	32.45	**60**	6.01	18.33	10.79	6.67
25	4	4	2	8	124.38	**45**	0.37	8.89	0.48	8.89
25	4	4	2	10	906.39	**46**	0.40	19.57	0.58	8.70
25	4	4	4	8	120.33	**36**	0.39	11.11	0.74	8.33
25	4	4	4	10	22.62	**23**	0.38	8.70	0.66	8.70
25	8	4	4	8	653.64	**70**	24.20	10.00	28.23	7.14
25	8	4	4	10	138.94	**67**	21.00	22.39	40.83	8.96
36	4	8	4	8	455.9	**46**	0.95	19.57	1.60	8.70

圖 5.1 為測資(25,8,4,4,10)的收斂速度示意圖，可看出 SA_{order} 收斂效果較顯著，

因為 SA_{order} 是迭代 $Greedy_{order}$ ，$Greedy_{order}$ 因為處理每筆訂單需一次就規劃三條路徑，規劃較無彈性，求解品質深受訂單順序影響。SA_{task} 是迭代 $Greedy_{task}$ ，$Greedy_{task}$ 因有整理任務序列，若有多筆訂單需要相同貨架，會進行訂單順序的調整，並且是按照任務的估算完成時刻，依序進行路徑規劃，並不會需要一次性規劃三條路徑，求解品質受到訂單順序影響較小，收斂效果較不顯著。

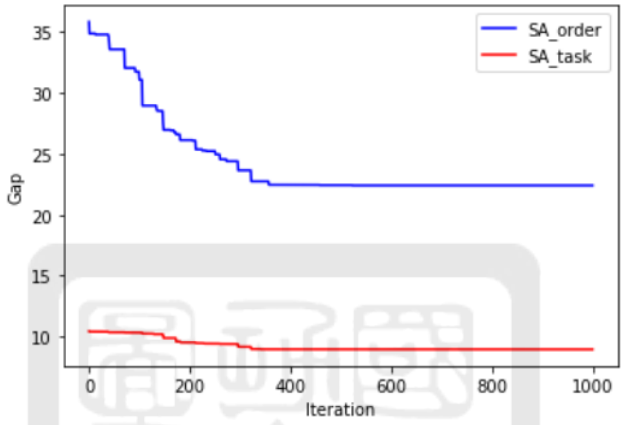

圖 5.1 測資(25,8,4,4,10)模擬退火法收斂速度

考量到實際的倉儲節點個數眾多，且需大量處理的訂單，我們亦測試了節點個數眾多的測資。因數學模式無法在時間內求解出可行解，因此只進行兩種貪婪演算法的比較，在節點個數多的測資中一樣是 $Greedy_{task}$ 有較佳的求解品質，在 n=1600 的測資，花費約三分鐘就能迅速得到一可行解，兩種貪婪演算法的測試結果如表 5.5 所示。

表 5.5 兩種貪婪演算法比較-大規模的網路圖

n	***O***	***R***	***K***	***P***	$Greedy_{order}$		$Greedy_{task}$ *	
					CPU Time(s)	***Obj***	***CPU Time(s)***	***Obj***
100	100	30	10	200	3.95	3710	5.94	**3497**
400	100	150	20	300	8.94	6289	11.24	**6016**
900	100	300	30	500	37.27	8306	43.02	**7731**
1600	100	600	40	1000	124.91	10624	187.56	**10137**

表 5.6 為參考文獻中所測試的最大網格圖大小，Sharon et al. (2015) 給定一群機器人集合，每個機器人皆有自己的起訖點，一旦到達訖點代表任務完成。Hönig et al. (2018) 同樣給定一群機器人集合，每個機器人皆有各自起點，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。這兩研究每台機器人只須規劃單一路徑。本研究機器人若有訂單需完成，至少規劃三條路徑，完成任務的機器人可被重新指派，大部分的機器人須完成多筆訂單，且還需考慮貨架與機器人彼此間的協作，問題設定較龐大。

表 5.6 測試最大的網格圖文獻比較

文獻	**測資網格圖**
Surynek (2015)	8x8
Sharon et al. (2015)	64x64
Ma & Koenig (2016)	30x30
Hönig et al. (2018)	32x32
王宗瀚(2020)	8x8
本研究	40x40

5.4數學模式與演算法 Gap 差異之處

雖然模擬退火法能收斂到一個更好的解但與數學模式還是存在一些差距，我們去觀察數學模式的結果，發現數學模式會指派閒置機器人去移走某些閒置卻擋路的貨架，將可能使其它乘載貨架的機器人能夠通過原本被卡住的節點以獲得一更短的路徑，進而縮短訂單完成時間。以下使用一小例子來說明，各機器人與貨架位置如圖 5.2。例子中的紅色機器人要搬運貨架從節點 3 移動至節點 9，數學模式會預先派一台閒置機器人去移走節點 6 的貨架，清空節點 6，使得紅色機器人能經過節點 6，直達節點 9，紅色機器人的總路徑長為 2，模式求解路徑如圖 5.3 至圖 5.7 所示。

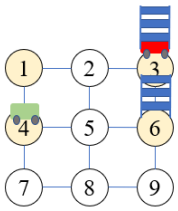

圖 5.2 小例子網路圖

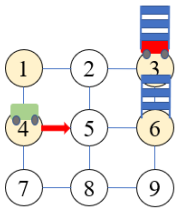

圖 5.3 模式求解(1)

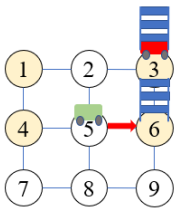

圖 5.4 模式求解(2)

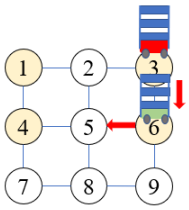

圖 5.5 模式求解(3)

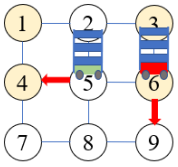

圖 5.6 模式求解(4)

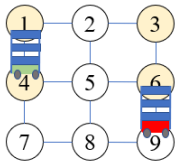

圖 5.7 模式求解(5)

演算法的設計只會尋找當下最短路徑，並不會指派閒置機器人移走障礙物，演算法會選擇繞開障礙物的最短路徑，此例子若使用本研究演算法，最短路徑為由節點 3 開始，經過節點 2、5、8，最終抵達節點 9，紅色機器人總路徑長為 4，相比數學模式的結果多了 2 單位，演算法求解路徑如圖 5.8 至圖 5.12 所示。演算法若要如數學模式考慮此情況，就要辨識路徑上哪些貨架可能是障礙物，移走哪個貨架會有更短的路徑，決定好要移開哪個貨架後，還得決定由哪個機器人移開，最後還得決定貨架移去哪個節點，也有可能選擇的節點又會擋到其他機器人，機器人間的協作相當複雜，上述為演算法設計的困難之處。

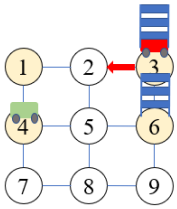

圖 5.8 演算法求解(1)

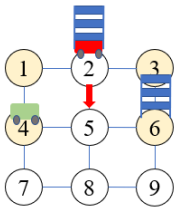

圖 5.9 演算法求解(2)

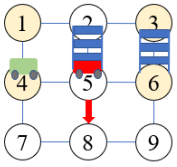

圖 5.10 演算法求解(3)

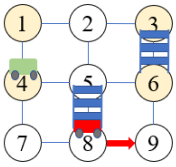

圖 5.11 演算法求解(4)

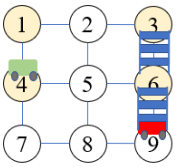

圖 5.12 演算法求解(5)

5.5 小結

本章比較完數學模式與演算法之優缺點，數學模式花費許多運算時間，但能得到最好的結果。貪婪演算法可以很快的得到可行解，也能求解比數學模式大上很多的規模，作為起始解再搭配模擬退火法，每筆測資皆收斂到更佳的解，且運算時間很短，可供實務上參考使用。

第六章 結論與未來研究方向建議

6.1 結論與貢獻

本研究處理多機器人路徑規劃問題。本研究假設各訂單該由哪組貨架及機器人完成、各個貨架及機器人的初始位置、貨架上的商品品項與數量、各個揀貨工作站的訂單需求皆是已知，期望最小化訂單總等待時間，規劃出無碰撞的機器人移動路徑，以下為本論文貢獻:

1. 貨架選擇與多機器人協同路徑規劃:

本研究提出之數學模式與演算法。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；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，因此可處理機器人小於貨架數目的情形；過往研究中每個機器人皆已被指派其起訖點，因此並無可重複利用同一機器人完成任務的可能；本研究提出之數學模式及演算法，皆可重複使利用貨架，亦考量貨架後續的歸位路徑

2. 以時空網路建構之數學模式：

由於機器人與貨架皆有可能會發生碰撞，因此我們使用時空網路圖來建構數學模式，以機器人在時空網路圖的節線經過與否作為變數，若將貨架在時空網路圖的節線經過與否也作為變數，整個數學模式的變數會過多。每個貨架必須藉由機器人搬運才能進行移動，本研究的數學模式將貨架的行動簡化為貨架被機器人抬起與放下，貨架的路徑可由求解結果進行反推，數學模式不需要新增多餘變數，可縮小求解問題的規模。

3. 快速得到可行解之兩種貪婪演算法：

批次處理訂單之貪婪演算法，每次處理一筆訂單，每筆訂單需規劃三條路徑，不得發生碰撞為前提，選擇當下能完成訂單的機器人與貨架最短路徑。批次處理任

務之貪婪演算法，預先整理機器人任務序列，避免貨架重複進出儲存區域，減少不必要的移動，另外，以不考慮碰撞的最短路徑進行任務的估算，按照任務預估完成時刻的先後，排序所有任務。根據任務優先順序，每次處理一項任務，各項任務選擇當下最快可抵達又無發生碰撞的路徑。上述兩個貪婪演算法能快速得到可行解，可適用快速且即時的規劃需求。

4. 使用模擬退火法進行收斂：

貪婪演算法的求解品質深受訂單的順序影響，本研究將貪婪演算法作為初始解搭配模擬退火法，透過改變訂單順序，來迭代求解，經過測試，每筆測資皆能收斂至更佳的解。

6.2 未來研究方向建議

本研究之多機器人路徑規劃問題，尚有許多問題仍可改善或延伸：

1. 數學模式及貪婪演算法之改善：

由於本研究之數學模式無法求解中、大規模的網路圖，最大能處理 36 個節點的網路圖，並且這小規模的網路圖亦需耗費不少時間方能得到最佳解，除了可考慮是否有其它建模方式，亦可嘗試開發啟發式演算法，先行縮小變數、限制式範圍或個數，以便縮小解集合，增加求解效率與品質。

本研究提出之貪婪演算法，在求解品質的部分尚有很大的改善空間。舉例來說，本研究的演算法並沒有針對閒置的機器人指派任務，而觀察數學模式的結果，我們發現閒置機器人若能被派去移走某些閒置卻擋路的貨架，將可能使其它乘載貨架的機器人能夠通過原本被卡住的節點以獲得一更短的路徑，進而縮短訂單完成時間，未來演算法的設計可針對閒置的機器人以及擋在最短路徑上的貨架，進行任務的安排，加以改善演算法的求解表現。

2. 模擬退火法之參數控制：

模擬退火法的參數設定會影響到演算法收斂速度，像是起始溫度的高低、降溫

速度的快慢以及終止溫度的大小，若想要有較高機會獲得高品質的解，可能得將迭代次數增加或是調整降溫速度等等，不過這就需要耗費的更長時間進行收斂，如何取得解的品質與求解時間平衡，亦是另一大議題。

3. 訂單的產品數量與種類：

本研究目前皆假設每筆訂單僅需一種產品，並忽略其數量。現實中，訂單可能包含多樣產品，且每樣產品所需要的數量可能不同；同時，存放在每個貨架上的產品數量可能也有所不同，如何將本研究之問題擴展到能處理產品數量的問題，也是另一個研究方向。

4. 機器人電量限制：

本研究目前皆不考慮機器人在倉儲內移動的所耗費的電力，現實中機器人會有電池電量與續航力的限制，使得機器人必須在其電量耗盡前充換電，因此，未來更貼近實務的研究可考慮將機器人的電量耗損與充換加入整體路線與排程的規劃。

5. 機器人的修復問題：

在實務上，機器人會有故障的情形發生，當有發生機器人故障需要進行排除狀況的時候，以往會封鎖部分區域，限制其它的機器人移入封鎖區，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，使機器人辨識去進行繞路。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

王宗翰. (2020) *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*. 國立成功大學工業與資訊管理學系碩士論文, 台南市. Retrieved from https://hdl.handle.net/11296/g7k43r

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.

Erdem, E., Kisa, D. G., Oztok, U., & Schüller, P. (2013). *A general formal framework for path finding 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.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence, 219,* 40-66.

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

Valle, C. A., & Beasley, J. E. (2019). Order allocation, rack allocation and rack sequencing for pickers in a mobile rack environment. *arXiv preprint arXiv:1903.06702.*

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 fulfillment 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