國立成功大學
工業與資訊管理學系碩士班
碩士論文

推盤式高密度倉儲系統揀貨路徑規劃問題

Picking Path Planning in High Density Puzzle-Based Storage Systems

研 究 生　　：沈 郁 恩
指 導 教 授：王 逸 琳　教授

中 華 民 國　一 百 一 十 一 年　六 月

國立成功大學

碩士論文

推盤式高密度倉儲系統揀貨路徑規劃問題

Picking path planning in high density puzzle-based storage systems

研究生：沈郁恩

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

周詩梵

指導教授：王逸琳

單位主管：

（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 111 年 6 月 17 日

摘要

隨倉儲空間成本增加，如何提升空間利用率並保持機動性成為重要課題。在推盤式高密度倉儲系統中，因大部分面積皆被承載貨物的推盤所佔據，推盤移動路徑有賴精準的空格位置調配，其移動過程常需先移動其它推盤才得以挪出空間以移動該推盤，宛如數字推盤遊戲一般，此與傳統自動倉儲系統常見的輸送帶或機器手臂的移動方式差異甚大，是一個多元商品網路流量問題。文獻中大多僅探討如何規劃單一推盤的移動路線，而忽略多推盤同時進行移動與是否碰撞等相關議題。

本研究考慮多個出貨推盤之揀貨路徑規劃問題，相較單一出貨推盤的移動，須避免為移動某出貨推盤卻因而導致其他出貨推盤更遠離其原先目的地，亦將「時間維度」納入考量，針對同時可移動的推盤數與碰撞限制等進行更細緻的設定。允許推盤含有多商品品項，需決定選擇哪些推盤送至揀貨點之路徑，以滿足訂單需求品項。本研究以推盤位置分佈為節點，可行移動為節線展開「時空網路」，並分別依「障礙物」與「空格」等不同觀點而設計兩個整數規劃模式。雖然此二整數規劃模式可得精確最佳解，但求解效率有限，為驗證與比較求解效率，本研究再設計「動態規劃演算法」與「滾動式求解演算法」，其中前者可設定求取精確解或估算解；而後者則試圖將原系統劃分成數個規模較小但有部分空間重覆的子系統，以滾動方式一次求解一個子系統，使得欲移動的推盤在每個子系統中得以持續往其目的地移動。

在本研究提出的三種可計算精確最佳解的方法中，依空格數由多至少，其運算效率表現具優勢者依序為「障礙物觀點數學模式」、「空格觀點數學模式」，與「動態規劃演算法」。針對求解空格數較少的大規模系統，上述三種精確解法將耗時過久，較適合使用效率較佳的「滾動式求解演算法」來處理，但其求解品質與運算時間仍會隨空格的位置分佈、子系統範圍劃分方式、以及子系統涵蓋之空格數而有所差異。由測試結果顯示，滾動式求解演算法在處理空格數較多的測資時效能較佳，且能藉由擴增子系統涵蓋範圍來改善求解品質，但需找到合適的子系統範圍劃分方式才能兼具求解效率與效能，建議未來可嘗試優化滾動式演算法子系統劃分邏輯，以及使用滾動式演算法計算之完工時刻為深度優先動態規劃法之初始最大時限來加速求解。

關鍵詞：高密度倉儲、多元商品網路流量、整數規劃、動態規劃、滾動式求解；

Picking Path Planning in High Density Puzzle-Based Storage Systems

Yu-En Shen

I-Lin Wang

Department of Industrial and Information Management, National Cheng Kung University

SUMMARY

This research discusses the problem of picking path planning in high density puzzle-based storage systems (PBS). The grid storage space is nearly fully occupied by loads (e.g., Autonomous Mobile Robot, AMR) allowed to move in the up, down, left, and right directions on a floor plane. Multiple loads can move simultaneously. This research plans the movement path of loads (including the target loads and the loads to be moved during the process) to minimize the time required for any specified load to move to the picking point. Unlike most literature that only minimizes the number of moving steps of a single target, this study considered movements of multiple loads and targets. When multiple loads can move simultaneously, issues such as waiting time between loads and avoiding collisions must also be considered.

The picking path planning problem can be regarded as a variant of the multi-commodity network flow problem. Two mixed integer programming (MIP) models are developed from the load or space point of view. They can calculate exact optimal solutions but are time-consuming. A dynamic programming algorithm (DP) and a rolling-horizon algorithm are developed to calculate a good solution with acceptable quality and time by parameter adjustments. Numerical analysis results show that when there are fewer spaces (i.e., almost fully occupied by loads), the MIP running time increases, but the DP running time decreases. The solution quality of the rolling-horizon algorithm depends on how the subsystems are divided. A subsystem of a larger space ratio leads to a smaller optimality gap. For future research, we suggest improving the partitioning logic of subsystems and solving the problem by the rolling-horizon algorithm cooperated with DP.

Key words: Puzzle-based Storage, Multi-commodity Network Flow, Integer Programming, Dynamic Programming, Rolling-horizon

.

INTRODUCTION

As the cost of storage space increases, how to improve space utilization and maintain mobility has become an important issue. In a high density storage system, since the loads almost occupy the entire space, the movement path of a load requires suitable space allocation. The same items may be stored in multiple loads. So, any such a load can be considered for shipping that item to the picking point (destination). Moving a load is similar to sliding a tile in a tile puzzle game, which is very different from the problems of conveyor belts or robotic arms commonly used in traditional automatic storage and retrieval systems. It is a variant of multi-commodity network flow problem. Most literature only discusses planning a single target's movement route, considering related issues such as the simultaneous movement of multiple loads and collisions. This research considers the picking path planning problem of multiple loads. The challenge lies in effective ways of not blocking another load's path while moving one of the target loads.

MATERIALS AND METHODS

In this research, we assume the warehouse space is a grid network, and each load takes a unit of time to conduct a step (up, down, right, or left). Assuming the initial map is given, we can construct a time-space network used to model the movement of loads over time and space. In addition to the common settings used for planning the single target path, we have also developed modeling techniques to deal with extended settings such as block movement, multiple targets, and multiple goods in a load.

Two integer programming (IP) models are designed: the first one is based on the "load" viewpoint, and the second one is based on the "space" viewpoint. They differ from each other, especially on the collision constraints. These IP models calculate exact optimal solutions, yet they consume too much time on problems of larger scale or low space ratio.

We propose a dynamic programming algorithm (DP) and a rolling-horizon algorithm that could solve a good solution quickly. In particular, DP can calculate an exact optimal solution for cases containing a single space. For instances containing more spaces, DP needs to limit the number of state enumerations to calculate a good solution. The rolling-horizon algorithm divides the original system into several smaller but partially overlapped subsystems. It solves one subsystem at a time in a rolling fashion so that the target can keep moving towards its destination in each subsystem.

RESULTS AND DISCUSSION

Numerical testings are performed on a computer with Windows 10, Intel Core i9-10900, 2.80GHZ Processors, and 8GB RAM. IP models are implemented in Python and solved by

Gurobi 9.3 version. We simulated cases that randomly distribute loads and spaces for a specific number of spaces in systems. Among the three exact optimal solution methods we have proposed, the load IP model is the most efficient, the space IP model is the second most efficient, and the DP takes longer than others. For a large-scale system of very high density (i.e., very few spaces), all the above three solution methods take too much time, and thus this is more suitable to be solved by the rolling-horizon algorithm. The important factors are the allocation of loads, the decomposition mechanism to obtain overlapped subsystems, and the number of spaces covered by the subsystems.

CONCLUSIONS

This thesis seeks optimal or nearly optimal movements for target loads to reach the picking station as soon as possible in high density puzzle-based storage systems. We formulate two new IP models: the load IP model (with a load as the object to move) and the space IP model (with a space as the object to move) on the time-space network to calculate optimal paths for all loads and spaces. We proposed a DP and a rolling-horizon algorithm to deal with cases of larger sizes or fewer spaces. The test results show that the rolling-horizon algorithm performs better when dealing with instances containing many spaces, whose solution quality can be further improved by expanding the coverage of the subsystems if the subsystems can be suitably decomposed and overlapped. For future research, we suggest investigating good subsystem partitioning logics for the rolling-horizon algorithm. We can use the rolling-horizon algorithm to calculate a good completion time as the initial planning horizon length for the depth-first searching DP algorithm to save the calculating time.

致謝

感謝指導教授王逸琳老師的指導，無論在專業知識、學術研究，還是人生際遇，都有老師的照顧與提點，很幸運能在碩士生涯遇到如此通情達理的一盞明燈，總是提供許多思考方向，老師的肯定與鼓勵也給予我許多勇氣去面對更艱鉅的挑戰，感謝老師的用心與包容。Proposal 與口試期間，承蒙書審委員李賢得老師與呂執中老師，以及口試委員丁慶榮老師、周詩梵老師與林仁彥老師的建議與指正，使本論文更加完備。祝福各位老師身體健康，事事順心。

感謝 Lab 61205 的學長承中和書桓的照顧，樂於分享許多所見所聞，增廣我的見識，毫無隔閡的親和力與面對困境的幽默感，使實驗室充滿歡聲笑語。同屆的威銓和蕎仔，有你們一起修課、一起玩樂，還有一起完成實驗室的大小事，讓我的學習之路不孤單。Rani 和 Ocha，謝謝妳們包容我半吊子的英文，帶我認識世界的不同角落。畢業後大家都有各自的忙碌，願大家都能找到生活的平衡，穩步發展。

Lab 61205 的學弟妹彥融、怡玲、凱翔和閔樺，還有實驗室的常客冠廷和芯瑜，雖然見面的機會不多，但每次見到你們總是熱熱鬧鬧，讓生活不那麼枯燥沉悶，謝謝你們邀請我加入你們的世界，希望你們可以在彼此的扶持中順利完成研究，邁入人生的下一個里程。另外，多虧 Lab 61120 的朋友們，讓我有事沒事跑去串門子，限時限量的零食和無限的幹話，午餐不知道要吃甚麼的去處，還有快樂後宮群的不定時關心和橘子的種種陪伴，讓我的碩士生活有所歸屬。有空就聚聚，沒空就偶爾聯繫，期待未來再創造更多回憶。

「因為要謝的人太多了，只好謝天」這種常見的結語似乎不足以表達我的謝意，但一路走來承載了太多的幸運與機遇，無法一一列舉與致謝，只能在此致上最真摯的感謝與祝福，希望未來可以將這份溫暖傳承下去，深深鞠躬。

目錄

表目錄

圖目錄

第一章 緒論

電子商務盛行，使物流公司的理貨能力的要求也日益提升。移動式貨架的設定可大幅降低揀貨所需時間，而推盤式倉儲系統可降低存貨空間的使用，皆可提升理貨能力。本章將依推盤式倉儲系統為研究背景，再依此設定研究目的與研究問題，最後簡介本研究之論文架構。

1.1研究背景與動機

電子商務盛行，使產品需求趨向少量多樣，加上對於商品即時性的追求，因此對物流公司的理貨能力的要求也日益提升，而揀貨便是其中需耗費大量時間的環節。傳統的揀貨作業中，貨架的位置固定在倉儲系統中，揀貨員需在貨架間移動，以取出訂單所需之商品，此作業模式需耗費許多時間。再加上訂單量上升，傳統模式所需的倉儲空間亦大幅增加，單依人力在貨架間揀貨，難以負荷現今遇發盛行之電商需求。

現有新型態之倉儲系統，使貨物可透過機器人、推盤或輸送帶（如圖 1.1 至圖 1.3）送至揀貨點(Yalcin, 2017)，使揀貨員只需在揀貨工作站，便能取得各式商品，大幅提升揀貨效率並降低揀貨錯誤率(Li and Liu, 2016)。

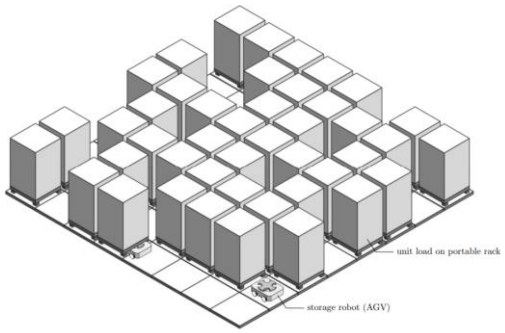


圖 1.1 使用機器人搬運之倉儲系統(Yalcin, 2017)

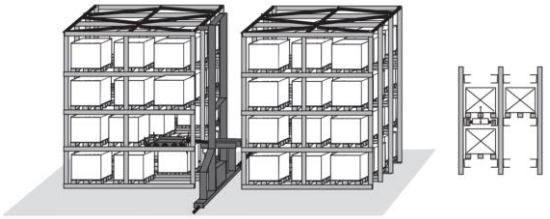


圖 1.2 使用推盤搬運之倉儲系統(Yalcin, 2017)

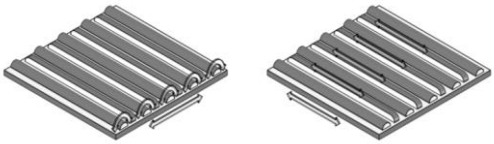


圖 1.3 可變換方向之輸送帶 (Yalcin, 2017)

固然移動式貨架能使揀貨流程更加彈性，但由於台灣的腹地空間有限，若仍保持相同的倉儲佈置，會造成過高的空間成本。因此需要更有效的倉儲空間運用方式。Gue (2006)定義在取貨過程中需挪開其他物品者為高密度儲存系統，為了提升倉儲密度，探討在各尺寸的倉儲系統中不同深度（即存貨與走道之間的最大距離）時，系統可達到之最高密度，並發展演算法生成該深度對應之佈置。

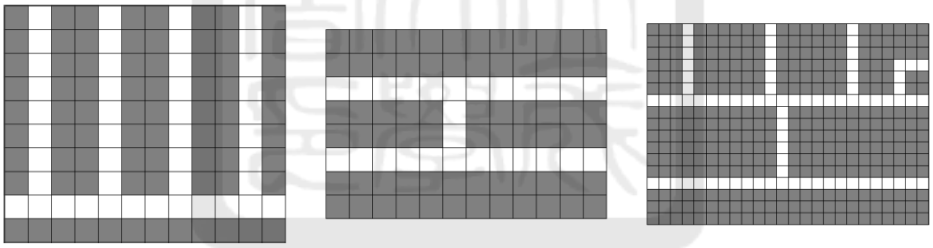


圖 1.4 走道深度為 1、2 和 3 時的佈置(Gue, 2006)

儲存密度提升使移動空間被壓縮，良好揀貨路徑的設計更發重要(R. De Koster et al, 2007)，雖然目前有許多文獻討論如何以良好的效率從倉儲系統取出貨物，但皆囿於「有走道」的佈局模式。但當系統密度提升時，限縮的走道空間將大幅提升移除障礙物所需時間，以圖 1.5 之左圖為例，欲取出圖中實心位置之推盤，需將黑框中所有推盤移置走道，將出貨推盤送置揀貨點並將其餘推盤送回儲存區，此行為需耗費許多時間。若能打破走道的侷限，使每個格點皆可用於運送與儲存推盤（如圖 1.5 之右圖所示），可大幅提升高密度倉儲系統之取貨效率。

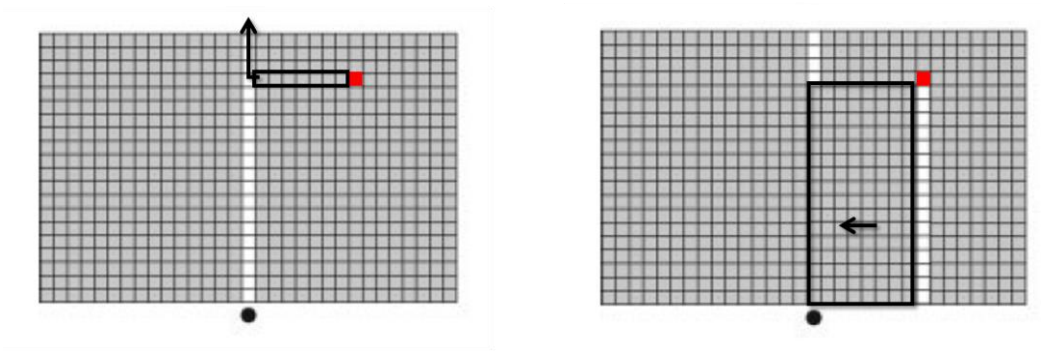


圖 1.5 於高密度環境取出實心推盤之示意

Gue and Kim (2007)提出推盤式倉儲系統，以數字推盤遊戲（如圖 1.6）為基礎，即四乘四之網格中，其中十五格佔有貨物，並且只能透過垂直或水平位移將貨物移至唯一空格的模式。打破傳統對倉儲系統必須預留「走道」空間的佈置格局，只要有空格便能透過空格位移，將貨物移至目的地，藉由釋出走道空間使空間利用更加彈性，最大化空間利用率。Yalcin (2019)等學者亦有討論推盤式倉儲系統之相關問題（如圖 1.7）。

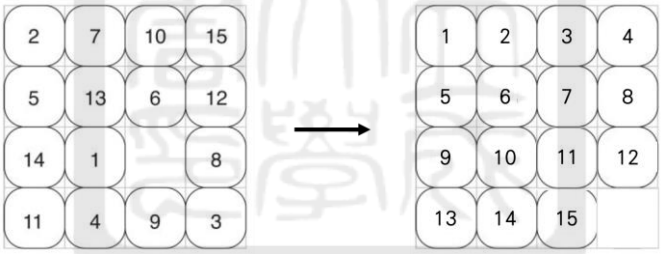


圖 1.6 數字推盤遊戲示意圖(Gue & Kim, 2007)

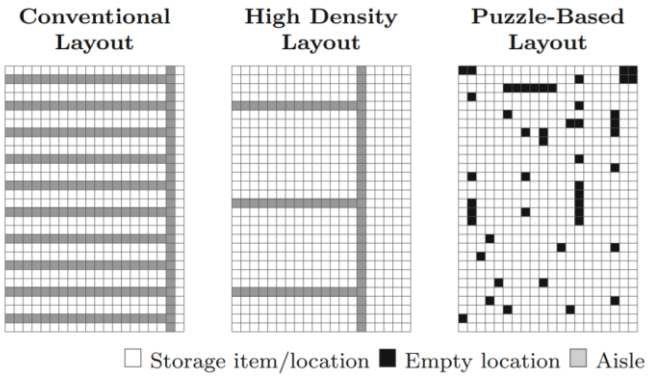


圖 1.7 傳統倉儲、有走道之高密度倉儲與推盤式倉儲示意(Yalcin et al., 2019)

1.2研究目的

在推盤式倉儲系統提出後，許多學者根據此系統設計取貨相關之演算，但皆分別在空格與目標的數量、移動限制與終點設定等面向簡化許多，且大多未將「時間」維度納入考量，僅探討個推盤間之移動路徑，無法由時間因子進一步判斷是否同時有多推盤位移與如何避免碰撞等情形，與實務應用仍有很大的落差，如Gue and Kim (2007)僅討論以單一空格取出單一目標物所需最短時間之演算法，而以多空格取出單一目標物(Gue and Kim, 2007; Taylor and Gue, 2008; Kota et al., 2015; Ma et al, 2021)及多目標物(Mirzaei et al., 2017; Yalcin et al., 2019)則無法保證最佳解。

除了每單位時間移動一個推盤至相鄰空格之外，「區塊位移」與「同步移動」亦是現實中常見的移動方式。區塊位移是指同一時刻多個相鄰推盤向同方向移動（如圖 1.8a-1），若不允許區塊位移，便只能各時刻逐一移動推盤（如圖 1.8a-2）。同步移動則是同一時刻系統中可有多個推盤進行位移（如圖 1.8b-1），反之則需逐一移動（如圖 1.8b-2）。目前討論區塊位移與同步移動的文獻相當有限。Gue and Kim (2007)探討如何使用單一空格在允許區塊位移的條件下，將單一目標送至終點；Yalcin et al. (2019)發展之演算法允許同步移動，透過多個空格將多目標物送至終點，但無法保證取得最佳解；Bukchin and Raviv (2021)設計之整數規劃數學模式，同時考慮區塊位移與同步移動，使用多個將單一目標送至終點，此文獻已相對其他文獻更加完備，但與現實仍有很大的落差。

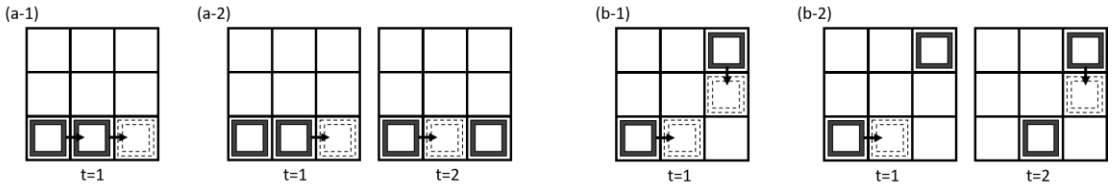


圖 1.8 區塊位移與同步移動示意圖

現實中仍有許多其他因素需納入考量，如機器人與推盤改變移動方向皆須耗費時間。同一商品可能放置在多個推盤中，單一推盤亦可能裝載多種商品。而在系統中，推盤抵達終點後，應在系統中等候下一次的訂單，而非直接離開系統。目前尚未有推盤式倉儲系統相關之文獻針對以上條件進行討論。因此本研究將在各推盤與商品儲存位置、揀貨站位置與訂單內容已知的條件下，以數學模式規劃最小化最遲完工時刻之推盤選取與移動路徑。考量到數學模式求解規模有限，且訂單的路徑規劃問題在現實中亦變動頻繁，本研究亦希望提出有效率且兼顧求解品質的演算法，以符合實際應用的需求。

1.3問題設定

問題分為基礎設定和延伸設定，基礎設定為各推盤僅有一種商品，已知欲出貨推盤，在可同步移動多個推盤但僅能向空格格點移動的條件下（即不允許區塊位移），將此推盤送至指定揀貨點，本研究之演算法將使用基礎設定，並以增進求解效率為主要目的。

另一方面，為考量更多現實情境，本研究亦討論延伸設定：系統中有數量不等之多種推盤，每種推盤可裝載多種商品組合，允許區塊位移的移動方式，並將轉向所需之時間列入考量。本研究將探討「選擇」哪些推盤，以及規劃推盤送至揀貨點的「路徑」。設訂單需求為甲、乙與丙各一份，左下角灰色格子為揀貨點。以圖 1.9 為例，同時選擇「甲乙」和「乙丙」推盤可滿足需求，只選擇「甲乙丙」推盤亦可滿足需求，本例子選擇「甲乙」和「乙丙」兩推盤，並規劃出合理的路徑，將出貨推盤送至揀貨點。

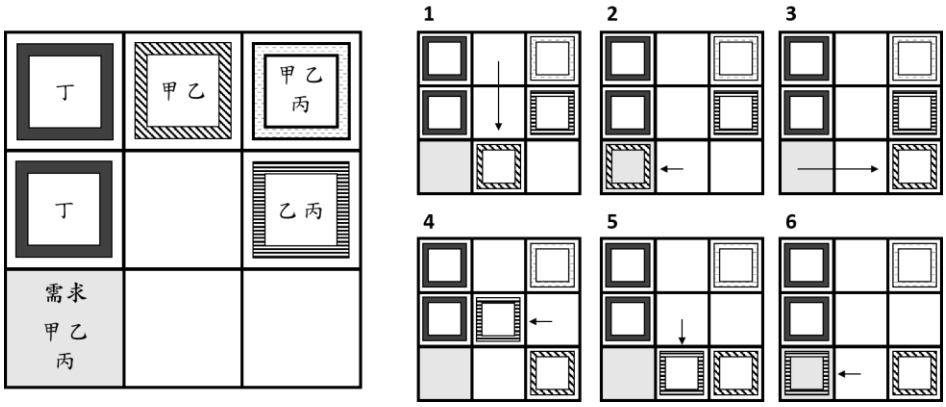


圖 1.9 問題描述

1.4論文架構

本論文之架構如下：第二章推盤式倉儲系統之相關文獻；第三章定義本研究之問題設定，並建立與說明整數規劃數學模式；第四章提出演算法以確保維持一定求解品質，同時加快求解速度；第五章進行數學模式與演算法之數值分析；第六章則是結論與未來研究方向的建議。

第二章 文獻探討

雖然已有許多文獻探討揀貨路徑規劃相關的議題，但皆是有區分為「走道」與「儲存區」的佈置設定，探討「推盤式倉儲系統」的文獻相當有限。本章將依推盤式倉儲系統的不同設定，依序探討單出貨推盤、多出貨推盤之演算法，再討論允許「區塊位移」與「同步移動」之文獻，如何將其實踐在演算法或數學模式中，最後再與本研究之問題設定進行比較。

2.1單一出貨推盤之相關文獻

Gue and Kim (2007)打破過往有走道的倉儲佈置格局，提出推盤式倉儲系統，透過空格位移將貨物移至目的地，以最大化空間利用率。此文獻討論在僅有一格空格的條件下，將單一出貨推盤移動至目的地所需之步數。貨物只能透過空格移動，因此如何將僅有的空格移動至出貨推盤欲移動之方向是本文討論的重點，如圖 2.1 中上圖所示，圖中欲將出貨推盤（黑色）往下移動一格，但空格（白色）此時卻在目標物右方，因此需將空格移動至出貨推盤下方，方能讓出貨推盤下移一格。文獻依空格位置與目標方向之關係，將常見之移動模式分為垂直向與反方向（如圖 2.1），若空格所在處與目標前進方向垂直，需三步方能將空格移至出貨推盤欲移動之位置；若兩者為反方向，則空格需移動五步方能抵達理想位置。

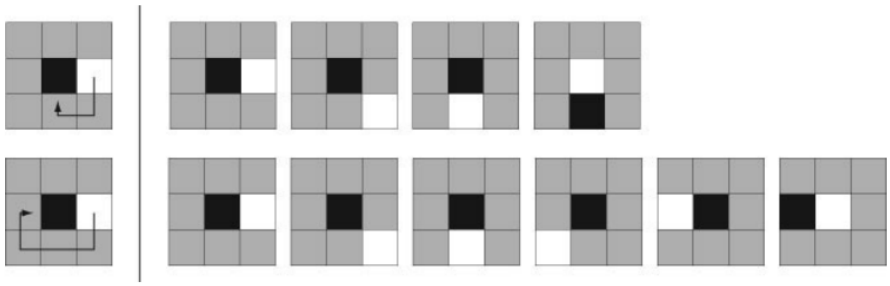


圖 2.1 垂直向位移與反方向位移說明(Gue and Kim, 2007)

圖 2.2 中，欲將出貨推盤移至左下角，相較同方向連續移動（如：先向左移動三步，再向下移動兩步），需進行三次反方向位移（五步）與一次垂直向位移（三步），共十八步；若照圖 2.2 的方式，兩方向交錯移動（向左移，再向下移，兩者交錯直到抵達邊界），僅需四次垂直向位移（共十二步）即可抵達終點。由此可知，兩方向交錯移動的方式可最小化單一空格運送單一目標至目的地所需之總移動步數，並依此計算出依此方式移動之最小總步數為 $6i+2j-13$ 步，i 為行數和列數中較大者，j 為較小者。

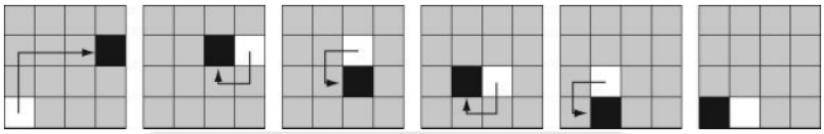


圖 2.2 交錯移動說明(Gue and Kim, 2007)

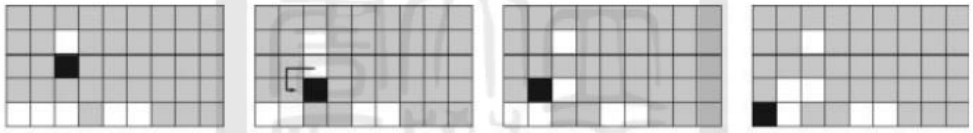


圖 2.3 多空格時路徑規劃案例 1 (Gue and Kim, 2007)
（黑色為出貨推盤，白色為空格，灰色為障礙物）

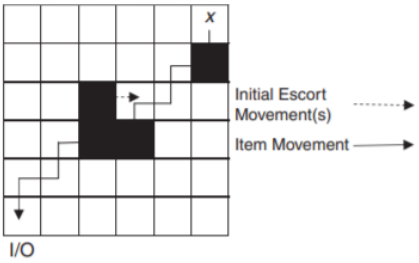


圖 2.4 多空格時路徑規劃案例 2 (Kota et al., 2015)
（黑色為空格，白色為障礙物，X 為出貨推盤初始位置）

有許多文獻嘗試發展以多空格將單一目標送至目的地之演算法，但皆未能保證取得最佳解(Gue and Kim, 2007; Taylor and Gue, 2008; Kota et al., 2015)，其共通

之邏輯為將空格移至目標物至終點的必經之路上，形成臨時的「通道」，使目標物盡早抵達目的地，如圖 2.3 和圖 2.4 所示。其中，圖 2.3 中的黑色為出貨推盤，白色為空格，灰色為障礙物；而圖 2.4 中的黑色為空格，白色為障礙物，X 為出貨推盤初始位置。

Bukchin and Raviv (2021)發展使用多空格移動單一出貨推盤，且允許區塊位移與同步移動之整數規劃數學模式與動態規劃演算法，並透過限制同一時刻可移動之物品數量以降低計算時間，得出最佳解或近似解。他們將此問題視為多元商品網路流量問題(Multi-commodity network flow)，將目標物視為一種商品（圖 2.5 中紅色物品與紅色流量），所有障礙物視為另一種商品（圖 2.5 中藍色物品與藍色流量），並將原始網路圖擴展為時空網路圖(time-expanded network)以表達各類商品在各時刻之流動情形（圖 2.5 之網路圖）。演算法以動態規劃為基礎設計，僅考量少量空格的情形及限制同時可移動之推盤數，減少狀態數以增進求解效率

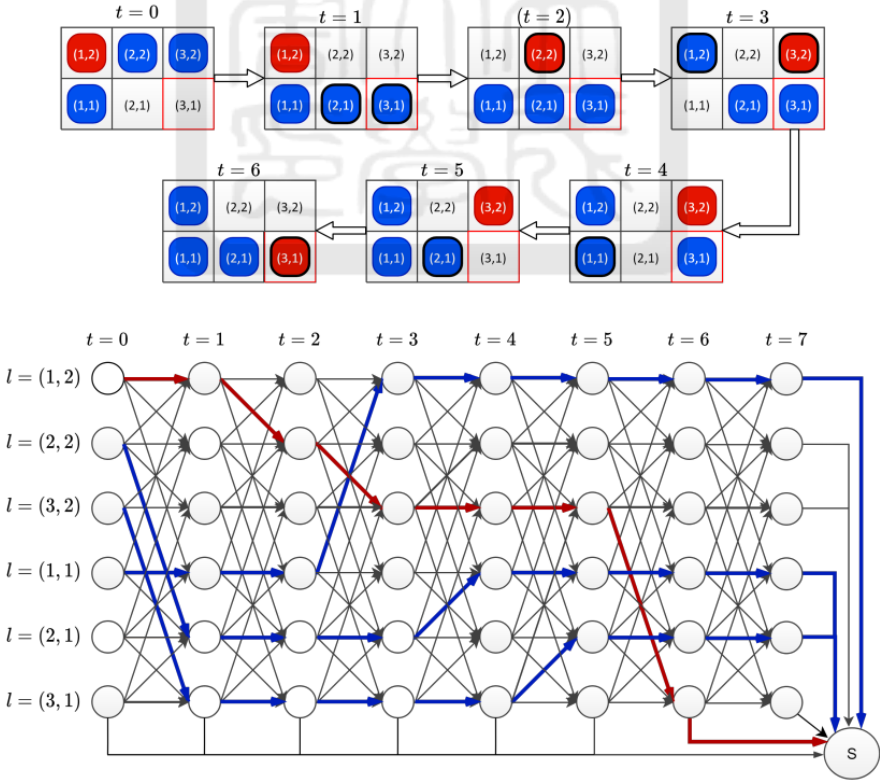


圖 2.5 數學模式網路圖說明(Bukchin and Raviv, 2021)

2.2多出貨推盤之相關文獻

Mirzaei et al. (2017)討論在單一空格的條件下，取出兩出貨推盤最短取貨時間。該文獻列舉有兩出貨推盤時，有三種可能的移動情形：(1)兩者分開移動、(2)兩者中間隔其他貨物一同移動（如圖 2.6a），以及(3)兩者相鄰一同移動（如圖 2.6b）。並於文獻中證明前兩者的效率皆不如第三者。基於以上證明，其探討如何選擇集合點，使兩出貨推盤移動至相鄰位置，再以前一小節所述 Gue and Kim (2007)之方法為基礎，將垂直向與反方向兩種移動方式，從單一出貨推盤時需三步與五步，調整成兩出貨推盤須五步（如圖 2.7a）與七步（如圖 2.7b），再以相似之計算方式，得出將出貨推盤皆送至目的地所需之最小步數。

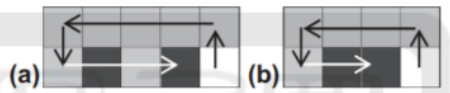


圖 2.6 一同移動兩出貨推盤之方式(Mirzaei et al., 2017)

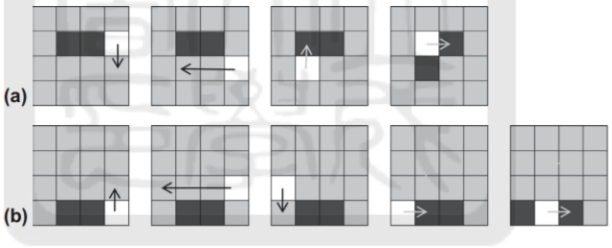


圖 2.7 垂直向位移與反方向位移說明(Mirzaei et al., 2017)

2.3區塊位移與同步移動之相關文獻

Gue and Kim (2007)亦討論到若要允許「區塊位移」，僅需稍微調整 2-1 所述之計算方式，垂直向位移仍是三步（如圖 2.8a），但反方向位移可由五步（如圖 2.8b）減少成四步（如圖 2.8c），再依與前述相似邏輯計算允許區塊位移之最小總移動步數。

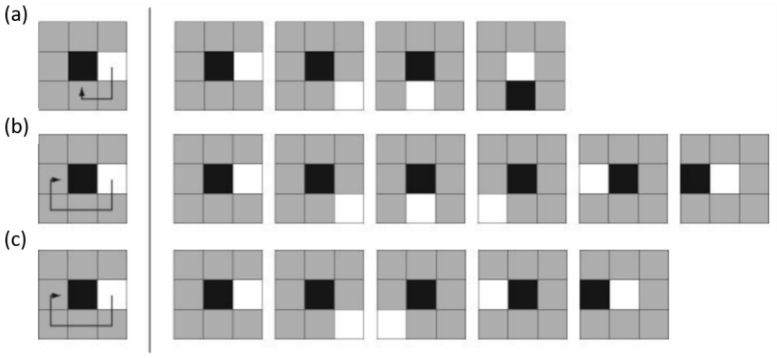


圖 2.8 允許區塊位移之垂直向位移與反方向位移(Gue & Kim, 2007)

Bukchin and Raviv (2021)亦提出允許區塊位移的數學模型，基於 2-1 節所提及的基礎，將原先僅允許移入空格的避撞限制式做了一些調整，允許移入非空格可能發生的碰撞分為三種情形：兩推盤互換位置（圖 2.9a）、兩推盤由垂直方向進出同一格點（圖 2.9b）、多個相鄰推盤向同方向移動（圖 2.9c，即區塊位移），為前兩者新增對應限制式，使此數學模式允許區塊位移的同時亦仍能避免碰撞。

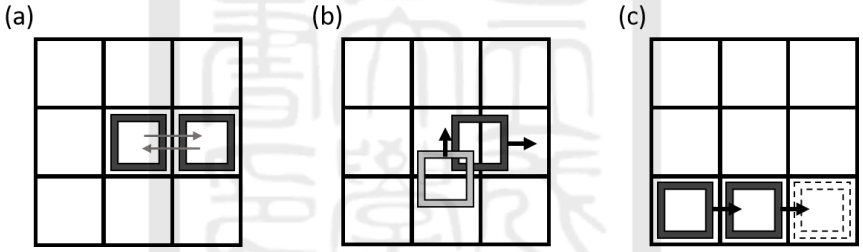


圖 2.9 允許移入非空格可能之碰撞情形

2.4小結

表 2.1 為推盤式倉儲系統與本研究之比較。本研究將發展整數規劃數學模式，考量機器人與推盤改變移動方向須耗費的時間，以及同一商品可能放置在多個推盤中，單一推盤亦可能裝載多種商品。推盤抵達終點後，在系統中等候下一次的訂單，而非直接離開系統。在各推盤與商品儲存位置、揀貨站位置與訂單內容已知的條件下，以數學模式規劃最小化最遲完工時刻之推盤選取與移動路徑。

表 2.1 推盤式倉儲系統相關文獻比較

作者	出貨推盤數/ 空格數/ 推盤商品數	移動設定	數學模式	演算法	最佳解
Gue & Kim (2007)	1/1/1	可區塊位移		V	V
	1/多/1			V	
Taylor & Gue (2008)	1/多/1			V	
Kota et al. (2015)	1/2/1			V	V
	1/多/1			V	
Mirzaei et al. (2017)	2/1/1			V	V
	多/1/1			V	
Bukchin & Raviv (2021)	1/多/1	可區塊位移 可同步移動	V	V	V
本研究	**多/多/多**	**可區塊位移 可同步移動 考慮轉向時間**	**V**	V	**V**

第三章 推盤式倉儲系統之揀貨路徑最佳化模式

本章節將說明本研究之問題定義與相關假設，並建立數學模式。我們依格點之相鄰關係建立網路圖，依時間將其展開為時空網路圖，並依此網路結構提出整數規劃數學模式，最佳化推盤在高密度環境中之揀貨路徑。

3.1問題描述

本研究將倉儲系統中各格點與其相鄰關係視為無向網路圖$G=(N,A)$，N表示網路圖的節點集合，A則是節線集合。需最小化從包含$|P|$種商品的$|R|$個推盤中，將訂單要求之商品送至指定揀貨點o所費時間。

問題分為基礎設定和延伸設定，基礎設定為各推盤僅有一種商品，已知欲出貨推盤，在可同步移動多個推盤但僅能向空格格點移動的條件下（即不允許區塊位移），將此推盤送至指定揀貨點。在延伸設定中考量更多現實情境，單一推盤可裝載多種商品，且有多個組合能滿足需求（如圖 3.1 所示），需挑選推盤並規劃移動至揀貨點的路徑，例如：共有甲、乙、丙與丁四種商品，一號推盤有甲和乙兩種商品，二號推盤有乙和丙兩種商品，三號推盤四種商品皆有，四號推盤則只有丁商品。若此時訂單需求甲、乙與丙商品各一份，將一號和二號推盤皆移動至揀貨點是一種滿足需求的組合，只移動三號推盤亦是一種選擇。而四號推盤無法提供此訂單所需商品，但仍在系統中佔有空間，而各類型推盤皆可能不只一個，再加上處於高密度的環境中，因此除了規劃「滿足訂單需求」之出貨推盤組合如何移動至揀貨點，其餘推盤如何移動方能「使出貨推盤組合更快抵達目的地」，亦是本問題之重點。

由於多個相鄰推盤同時向同方向移動（即區塊位移）不會造成碰撞發生，在延伸設定中允許區塊位移。此外，因為在現實中推盤改變移動方向需耗費時間（圖 3.2），本研究亦將此時間納入考量，若前一刻推盤是沿水平（鉛直）方向移動，

下次往鉛直(水平)方向移動前，需於原地停留至少一單位時間以改變推盤方向。

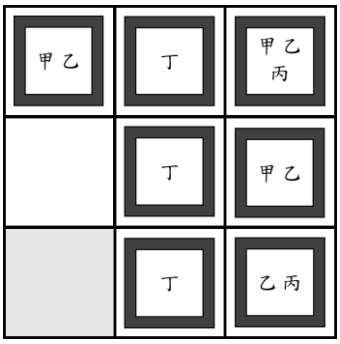


圖 3.1 倉儲系統示意圖
（左下角灰格表示揀貨點）

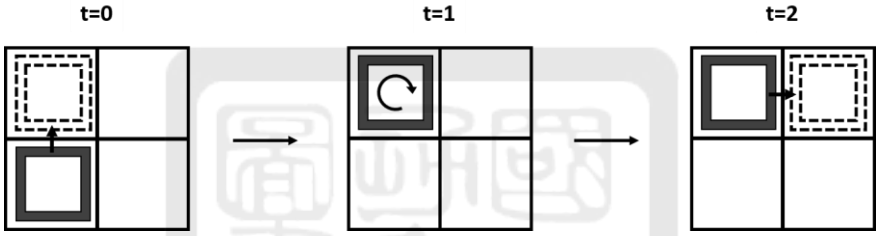


圖 3.2 轉向時間示意圖

3.2問題假設

以下為本研究數學模式之相關問題假設：

1. 每個格點皆為正方形。
2. 各推盤移動一格與改變方向皆須一時間單位。
3. 送至終點即視為完成訂單，不考慮訂單處理時間。
4. 同一訂單中各商品間無特定順序關係。
5. 各格點同時只能容納一個推盤。
6. 推盤僅可從 x 軸與 y 軸方向移動至相鄰格點。
7. 允許同一時間多個推盤進行位移，如圖 3.3 (b)所示。
8. 為避免碰撞，不允許同一時刻多推盤進入同一格點、兩推盤交換位置以及不同推盤由垂直方向進出同一格點之情形，如圖 3.3 (c)所示。

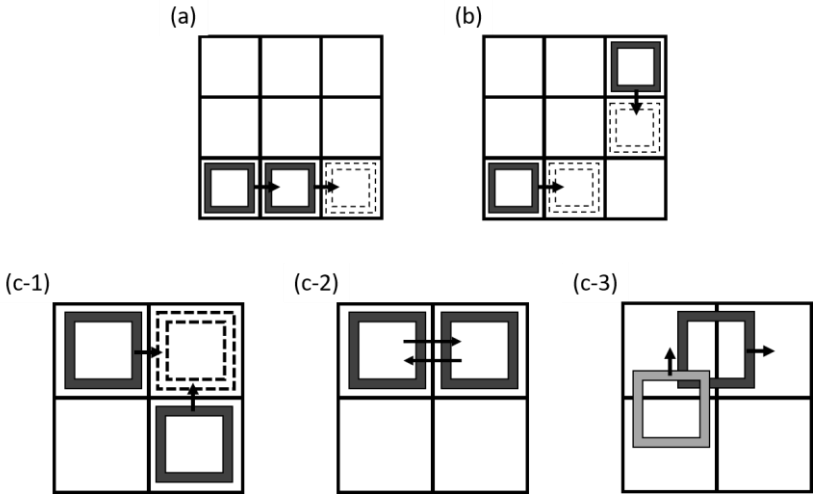


圖 3.3 假設說明圖示

3.3網路架構

在推盤式倉儲系統中，推盤需沿特定軌道移動，因此可將推盤的移動範圍視為各個格點，即無向網路圖的各節點，並可向上、下、左和右四個方向移動（如圖3.4），且移動所需時間皆為一時間單位。

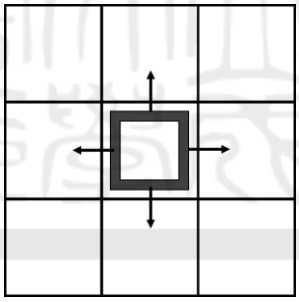


圖 3.4 推盤移動方向與格點示意

本研究以 3.1 節中提及之無向網路圖為基礎，加入時間軸，將其展開為時空網路圖。由使用者定義揀貨上限時間$|T|$，將時間離散化為$|T|$單位，每次的移動與轉向皆為一時間單位，需於時限內將訂單需求之產品送至揀貨點。

如圖 3.5 與 3.6 所示，時空網路由$|T|+1$個鉛直子層與$|N|$個水平子層構成，分別表示$|T|$個週期與$|N|$個格點。網路中每個節點表示各貨物在各時間點是否處於各格點的狀態，串聯網路中各節點即可推演出各貨物之路徑規劃。由於已知各推盤

之初始位置，因此本網路未設立虛擬起點；而在時限結束時，各推盤可能在系統中的任何格點，因此設定虛擬迄點 D，表達推盤的結束狀態並達到流量守恆。

1	2	3
4	5	6

圖 3.5 2×3 網格示意圖

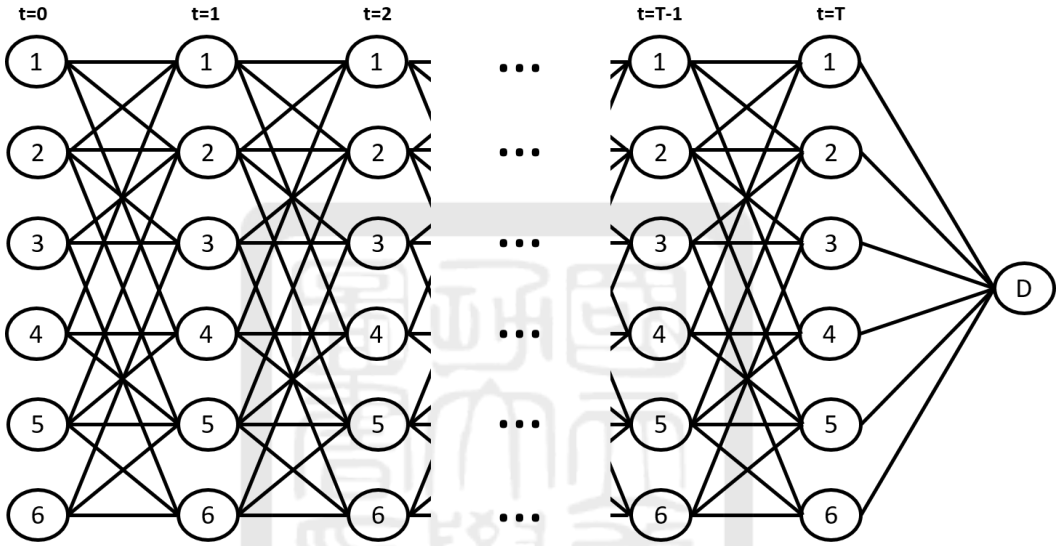


圖 3.6 時空網路圖—以 2×3 網格為例

3.4數學模式概念簡介

本研究依據兩種不同的邏輯建立數學模式，分別為「障礙物觀點之數學模式」與「空格觀點之數學模式」，本章節將會簡介這兩種數學模式之概念與差異。系統中各網格可劃分為三個類別，分別為「目標推盤」、「障礙物」與「空格」，呈現於圖 3.7 中，上方長方形表示系統格點隨時間之變化，下方網路圖對應格點狀態在數學模式依時空展開之網路圖狀態。藍色實心圓圈、灰色空心圓圈與紅色虛線三角形分別對應目標推盤、障礙物與空格。由於可以由目標推盤與障礙物之位移推得空格之移動路徑，因此建立數學模式時變數不需考量空格，此為障礙物觀點之數學模式（即圖 3.7 之左圖）。但由於此數學模式在空格數較少時容易花費

較長之運算時間，因此改為變數儲存空格而不儲存障礙物（如圖 3.7 右圖所示），以減少變數數量。障礙物觀點數學模式之避撞相關限制式較為直覺，因為變數接對應含有推盤之格點，僅需從「推盤移動相同格點會造成碰撞」出發，即可設計避撞限制式；而空格觀點數學模式因變數涵蓋具有實體推盤之目標推盤，以及用於搬運推盤，卻又具有與搬運工具不同移動邏輯之「空格」，因此須從不同的思維設計避撞限制，詳細於 3.6 節進行說明。

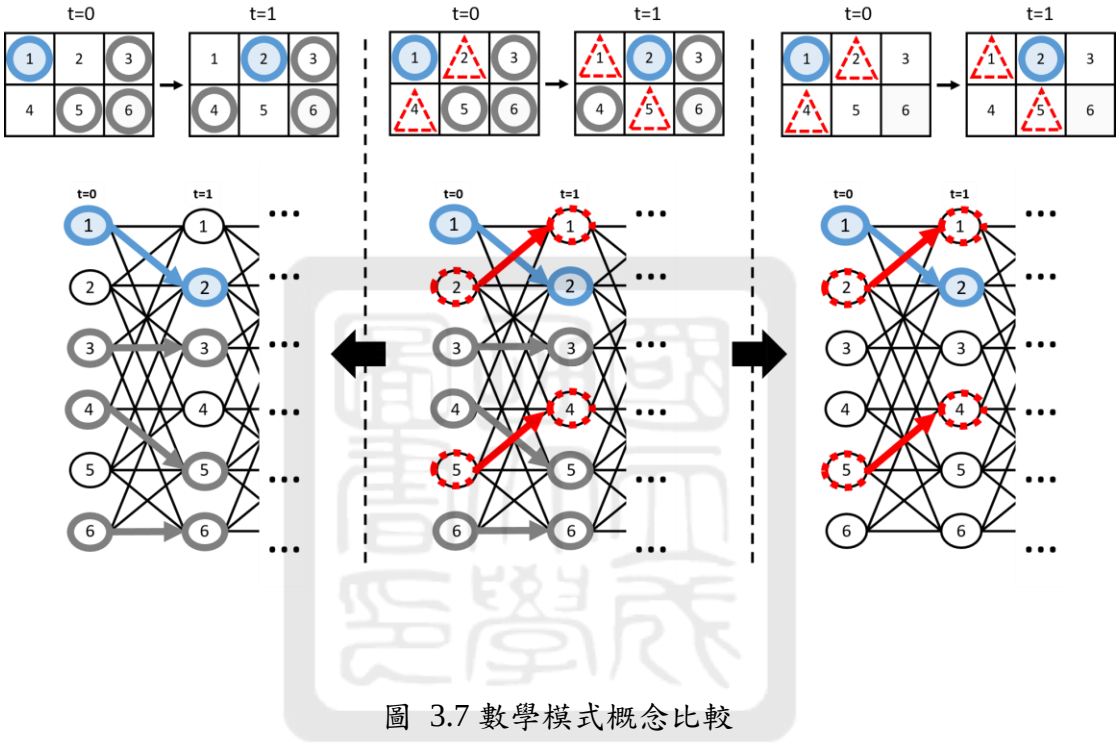


圖 3.7 數學模式概念比較

（左：障礙物觀點，中：所有格點類型，右：空格觀點）

3.5障礙物觀點之數學模式

在本章節將說明以障礙物為觀點建立之數學模式，各時刻出貨推盤與障礙物的流向儲存在變數中，並以此設計限制式。3.5.1 小節定義本模式使用之符號，包含集合、參數與變數，3.5.2 小節說明目標式設定，而 3.5.3 小節將介紹限制式。

3.5.1符號定義

集合

T	時間集合，$t=1\sim\|T\|$
R	各類推盤集合
P	各商品品項集合
N	網格格點集合
A	格點間路徑集合
δ_i^{in}	流入格點$i\in N$的節線集合，$\delta_i^{in}\subset A$
δ_i^{out}	自格點$i\in N$流出的節線集合，$\delta_i^{out}\subset A$
V_i	鉛直方向出入格點$i\in N$的路徑集合，$V_i\subset A$
H_i	水平方向出入格點$i\in N$的路徑集合，$H_i\subset A$

參數

M	極大常數
Num	總推盤數
D	虛擬迄點
o	揀貨點位置$o\in N$
$\|T\|$	揀貨時間上限
$\|R\|$	推盤種類數量
$\|P\|$	商品品項數量
$\|N\|$	格點數量
S_i^r	格點$i\in N$起始時是否有推盤$r\in R$
$pack_p^r$	推盤$r\in R$有商品$p\in P$為1；反之為0
d_p	需送至揀貨點之商品$p\in P$數量

變數

f_{ij}^{tr}	貨物$r\in R$在時刻t經過路徑$(i,j)\in A$為1；反之為0
w	所需商品送至揀貨點之最遲時刻，$w\in integer$

3.5.2目標式

本研究旨於提升在高密度環境下之揀貨效率，因此主要目標為最小化最遲完工時刻，意即最小化將所有指定商品品項皆送至揀貨點所需時間，如式(3.5.1)

$$Minimize \quad w \tag{3.5.1}$$

雖然本研究主要考量的指標為最遲完工時間，但在現實中，無謂之位移仍會造成不必要的能量耗損，如電量損失、機器磨損等，總移動距離如式(3.5.2)所示，將此式乘上微小懲罰值ε，與(3.5.1)合併為為本模式之目標式(3.5.3)。

$$Minimize \quad \sum_{t\in T}\sum_{r\in R}\sum_{(i,j)\in A, i\neq j} f_{ij}^{tr} \tag{3.5.2}$$

$$Minimize\ w+\varepsilon\sum_{t\in T}\sum_{r\in R}\sum_{(i,j)\in A, i\neq j} f_{ij}^{tr} \tag{3.5.3}$$

3.5.3限制式

本研究的限制式可分為流量狀態定義、避免碰撞之限制、轉向時間設定與完成目標要求，本章節將逐一說明。

首先是關於網路流之起訖點與能量守恆之定義。式(3.5.4)定義各推盤起始時所在格點，式(3.5.5)定義時間結束時，所有推盤將會從其所在格點移動至虛擬迄點D，最後式(3.5.6)定義各推盤在各時刻皆能在各格點保持流量守恆。

$$\sum_{(i,j)\in\delta_i^{out}} f_{ij}^{0r} = S_i^r,\ \forall r\in R; i\in N \tag{3.5.4}$$

$$\sum_{r\in R}\sum_{i\in N} f_{iD}^{|T|r} = Num \tag{3.5.5}$$

$$\sum_{(j,i)\in\delta_i^{in}} f_{ji}^{(t-1)r} = \sum_{(i,j)\in\delta_i^{out}} f_{ij}^{tr}, \forall t\in T; r\in R; i\in N \tag{3.5.6}$$

接下來是推盤在格點間移動的物理限制，以避免碰撞發生。在不允許區塊位移的基本設定中，式(3.5.7)限制每個格點同一時刻時最多一個推盤移入，式(3.5.8)要求推盤只能向空格移動。而在延伸設定中將移除式(3.5.8)，加入式(3.5.9)與式(3.5.10)以維持避撞限制。式(3.5.9)防止同一時刻兩推盤互換位置，最後式(3.5.10)和(3.5.11)分別限制同一時刻 x 軸（y 軸）方向移入且 y 軸（x 軸）方向移出情形。

$$\sum_{r\in R}\sum_{(j,i)\in\delta_i^{in}} f_{ji}^{tr} \le 1,\ \forall i\in N;\forall t\in\{0\}\cup T \tag{3.5.7}$$

$$\sum_{r\in R}\sum_{(j,i)\in\delta_i^{in}} (f_{ji}^{tr}+f_{ij}^{tr}) \le 1,\ \forall i\in N;\forall t\in\{0\}\cup T \tag{3.5.8}$$

$$\sum_{r\in R} (f_{ij}^{tr}+f_{ji}^{tr}) \le 1,\ \forall (i,j)\in A;t\in\{0\}\cup T \tag{3.5.9}$$

$$\sum_{r\in R}\sum_{(j1,i)\in H_i}\sum_{(i,j2)\in V_i} (f_{j1,i}^{t,r}+f_{i,j2}^{t,r}) \le 1,\ \forall i\in N;\forall t\in\{0\}\cup T \tag{3.5.10}$$

$$\sum_{r\in R}\sum_{(j1,i)\in V_i}\sum_{(i,j2)\in H_i} (f_{j1,i}^{t,r}+f_{i,j2}^{t,r}) \le 1,\ \forall i\in N;\forall t\in\{0\}\cup T \tag{3.5.11}$$

延伸設定中亦考量推盤改變前進方向時，需要一單位時間改變輪子走向。因為同一時刻各格點僅能容納一個推盤，可藉由限制格點的進出方向來進行轉向時間設定。式(3.5.12)與式(3.5.13)限制各格點若前一時刻有推盤從 x 軸（y 軸）方向進入，便無法在下一時刻馬上從此格點之 y 軸（x 軸）方向離開。

$$\sum_{r\in R}\sum_{(j1,i)\in H_i}\sum_{(i,j2)\in V_i} (f_{j1,i}^{t-1,r}+f_{i,j2}^{t,r}) \le 1,\ \forall i\in N;\forall t\in T \tag{3.5.12}$$

$$\sum_{r\in R}\sum_{(j1,i)\in V_i}\sum_{(i,j2)\in H_i} (f_{j1,i}^{t-1,r}+f_{i,j2}^{t,r}) \le 1,\ \forall i\in N;\forall t\in T \tag{3.5.13}$$

在本模式目標要求設定之部分，式(3.5.14)要求抵達揀貨點 o 之各商品數量需大於需求量，方可滿足訂單要求；式(3.5.15)則設定最遲完工時刻 w 為各推盤抵達揀貨點 o 的最遲時刻。

$$\sum_{t\in\{0\}\cup T}\sum_{r\in R}\sum_{(i,o)\in A} (pack_p^r * f_{io}^{tr}) \ge d_p,\ \forall p\in P \tag{3.5.14}$$

$$w+M(1-f_{io}^{tr}) \ge t,\ \forall t\in\{0\}\cup T;\forall r\in R;\forall (i,o)\in A \tag{3.5.15}$$

最後，式(3.5.16)與式(3.5.17)設定變數範圍。

$$f_{ij}^{tr}\in\{0,1\},\forall t\in\{0\}\cup T;\forall r\in R;\forall (i,j)\in A \tag{3.5.16}$$

$$w\in integer \tag{3.5.17}$$

3.6空格觀點之數學模式

有別於 3.5 節以障礙物為觀點，在變數中儲存障礙物的移動，在本章節將空格視為一種虛擬推盤，用其間接表示障礙物流向，以表達空格流向的變數建立數學模式。目前本模式僅適用於 3.2 節所述之基本設定，單一推盤僅裝載一種商品，計算在不允許區塊位移條件下，將出貨推盤送至揀貨點的最遲完工時刻。3.6.1 小節將定義本模式使用之符號，包含集合、參數與變數，3.6.2 小節說明目標式之設定，而 3.6.3 小節將介紹限制式。

3.6.1符號定義

集合

T	時間集合，$t=1\sim\|T\|$
R	各類推盤集合
N	網格格點集合
A	格點間路徑集合
δ_i^{in}	流入格點 $i\in N$ 的節線集合，$\delta_i^{in}\subset A$
δ_i^{out}	自格點 $i\in N$ 流出的節線集合，$\delta_i^{out}\subset A$

參數

M	極大常數
D	虛擬迄點
spa	空格對應之虛擬推盤索引值
Num^r	各推盤 $r\in R$ 數量
o	揀貨點位置 $o\in N$
$\|T\|$	揀貨時間上限
$\|R\|$	推盤種類數量
$\|N\|$	格點數量
S_i^r	格點 $i\in N$ 起始時是否有推盤 $r\in R$

d_r　　需送至揀貨點之推盤 $r \in R$ 數量

變數

f_{ij}^{tr}　　推盤 $r \in R$ 在時刻 t 經過節線 $(i, j) \in A$ 為 1；反之為 0

w　　出貨推盤送至揀貨點之最遲時刻，$w \in \mathbb{Z}^+$

3.6.2 目標式

本模式之目標式與 3.5.2 相似，皆以最小化最遲完工時刻 w 為主要目標，再加上避免無謂之位移的微小耗能懲罰值 ε，唯一的差別為使用空格流量變數便足以表達所有的位移耗能，調整為目標式(3.6.1)。

$$Minimize\ w + \varepsilon \sum_{t \in T} \sum_{(i,j) \in A, i \neq j} f_{i,j}^{t,spa} \tag{3.6.1}$$

3.6.3 限制式

首先是關於網路流之起訖點與能量守恆之定義。式(3.6.2)定義各推盤起始時所在格點，式(3.6.3)定義時間結束時，所有推盤將會從其所在點移動至虛擬迄點 D，最後式(3.6.4)定義各推盤在各時刻皆能在各格點保持流量守恆。

$$\sum_{(i,j) \in \delta_i^{out}} f_{ij}^{0r} = S_i^r,\ \forall r \in R; i \in N \tag{3.6.2}$$

$$\sum_{i \in N} f_{iD}^{|T|r} = Num^r,\ r \in R \tag{3.6.3}$$

$$\sum_{(j,i) \in \delta_i^{in}} f_{ji}^{(t-1)r} = \sum_{(i,j) \in \delta_i^{out}} f_{ij}^{tr}, \forall t \in T; r \in R; i \in N \tag{3.6.4}$$

接下來是推盤在格點間移動的物理限制以避免碰撞或不合理位移，此處為兩數學模式主要的差異，障礙物觀點數學模式中可直接限制格點流入量，在本模式需透過定義空格與空格、出貨推盤（變數直接表達）以及障礙物（透過限制空格變數間接表達）之間的關係進行避撞限制。式(3.6.5)限制每個格點同一時刻最多只能有一個出貨推盤或空格，不能同時存在兩個出貨推盤或兩個空格，亦不允許推盤和空格同時存在。式(3.6.6)定義出貨推盤僅能向有空格的格點移動，而空格可能自行移動（表示障礙物和空格交換位子），最後式(3.6.7)限制各格點各時刻

最多一個空格移入或移出，避免間接定義的障礙物發生超速等不合理位移。

$$\sum_{r\in R}\sum_{(j,i)\in\delta_i^{in}} f_{ji}^{tr} \le 1,\ \forall i\in N;\forall t\in\{0\}\cup T \tag{3.6.5}$$

$$\sum_{r\in R, r\neq spa} f_{j,i}^{t,r} \le f_{i,j}^{t,spa},\ \forall (i,j)\in A, i\neq j;\forall t\in\{0\}\cup T \tag{3.6.6}$$

$$\sum_{(i,j)\in\delta_i^{out}, i\neq j} f_{i,j}^{t,spa} + \sum_{(j,i)\in\delta_i^{in}} f_{j,i}^{t,spa} \le 1,\ \forall i\in N;t\in\{0\}\cup T \tag{3.6.7}$$

最後，與障礙物觀點數學模式相同，式(3.6.8)與式(3.6.9)分別設定需滿足訂單要求與最遲完工時刻 w 為各推盤抵達揀貨點 o 的最遲時刻，並於式(3.6.10)與式(3.6.11)設定變數範圍。

$$\sum_{t\in\{0\}\cup T}\sum_{(i,o)\in A} f_{io}^{tr} \ge d^r,\ \forall r\in R \tag{3.6.8}$$

$$w + M*(1-f_{io}^{tr}) \ge t,\ \forall t\in\{0\}\cup T;\forall r\in R;\forall (i,o)\in A \tag{3.6.9}$$

$$f_{ij}^{tr}\in\{0,1\},\forall t\in\{0\}\cup T;\forall r\in R;\forall (i,j)\in A \tag{3.6.10}$$

$$w\in\mathbb{Z}^+ \tag{3.6.11}$$

3.7初步執行結果

圖 3.8 為障礙物數學模式之延伸設定路徑規劃結果，共含有鞋子、裙子與上衣三種商品，而 A、B 與 C 三種推盤如圖中所示，各自包含不同的商品組合，而 O 推盤表示不含需求商品之其他推盤。左方的表格為各類推盤之初始格點位置，右下角灰色格點為揀貨點，需求商品為鞋子、裙子與上衣各一件。右圖為在允許區塊位移，並考量轉向時間的情況下，實際規劃之路徑。

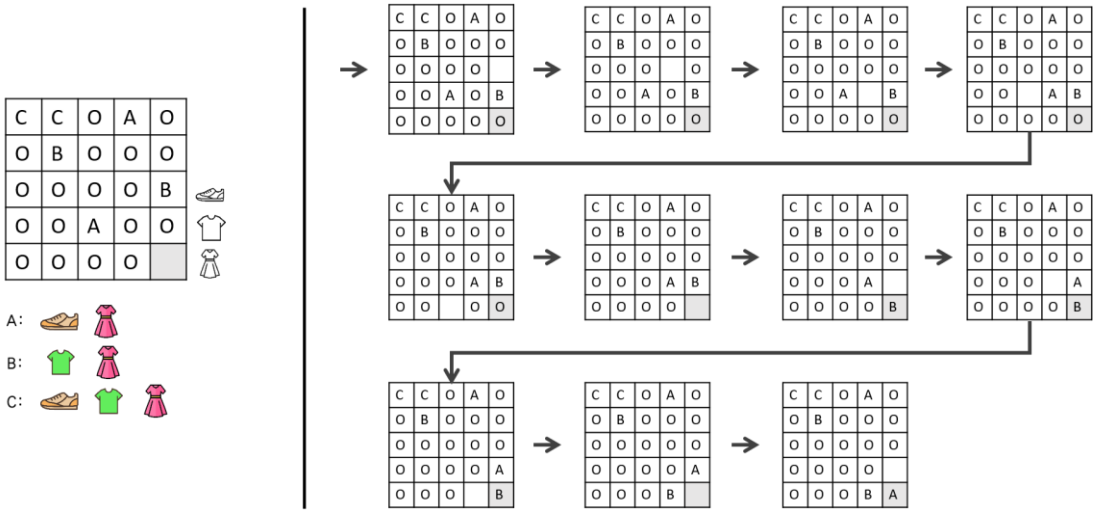


圖 3.8 障礙物數學模式之延伸設定執行結果

3.8小結

本章使用時空網路概念將網路圖展開，並分別依障礙物與空格為觀點建立數學模式以規劃推盤之路徑，這兩種模式在不同空格數量的條件下各有優劣。由於展開網路使變數與限制式數量大幅增加，若網路規模提升，將耗費過長的求解時間。若欲解決推盤式高密度倉儲系統取貨耗時長的問題，提升其應用於實務之可行性，需設計更有效率的演算法，在可接受的時間內得出近似最佳解。

第四章 推盤式倉儲系統之揀貨路徑最佳化演算法

第三章所提出之數學模式，在系統規模提升與空格數極少時，會花費許多時間求解，在實務上的應用價值較低，因此設計演算法在有限時間內取得足夠好的可行解。本章節將說明動態規劃演算法與滾動式求解演算法。

4.1動態規劃演算法

在動態規劃演算法中，每種推盤分佈狀態視為一個節點，以單位時間內可行之變化為節線，展開分佈狀態網路圖，如圖 4.1 所示，綠色推盤為出貨推盤，其餘為障礙物。各推盤起始時所在格點狀態為起始分佈狀態節點，所有出貨推盤位於揀貨點之分佈皆屬於終點分佈狀態節點，以廣度優先搜尋(Breadth-First Search, BFS）與深度優先搜尋（Depth-First Search）探索從起始狀態出發至終點狀態之最短路徑，並於 4.1.1 和 4.1.2 兩小節中說明。

本演算法的問題設定為 1.3 節提及之基本設定。而當空格數增加或同一時刻允許同步移動的推盤數越多時，展開之網路圖規模隨之膨脹，大幅影響演算法效率，因此本演算法僅考慮每時刻最多移動兩推盤之情形。

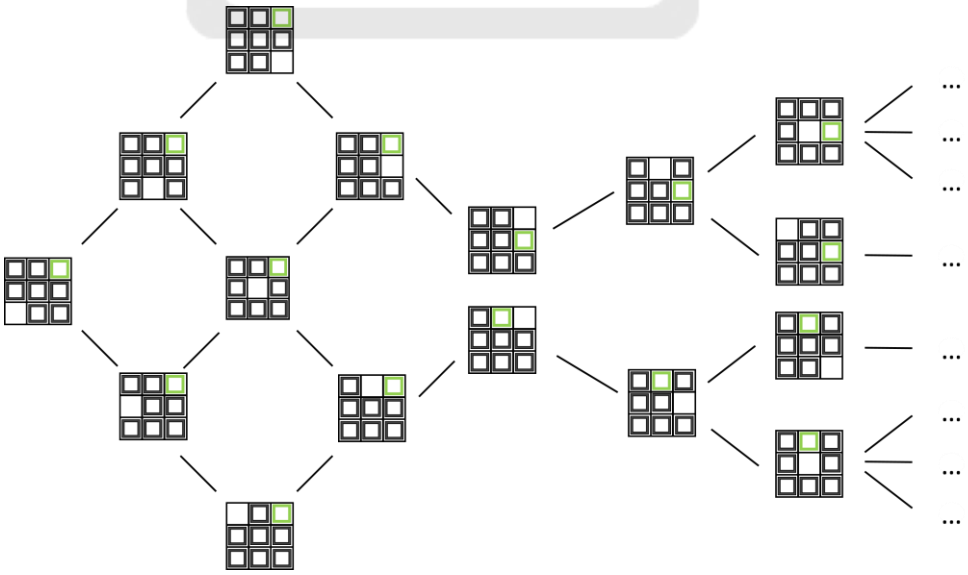


圖 4.1 格點狀態時空展開網路示意

4.1.1廣度優先搜尋探索

從起始分佈狀態出發，以廣度優先搜尋探索所有相鄰節點 *n*（表 4.1 為程式流程），記錄此節點與起始節點之最短距離與路徑，再判斷此節點是否屬於終點狀態。若屬於終點狀態，則結束探索；若不屬於，將其所相鄰節點加入待探索的節點列表 *NextList*，再探索下一個尚未探索之節點，重複探索直到觸及終點狀態。

由於在廣度優先搜尋中，會先探索相鄰之所有節點，而分佈狀態網路圖中所有節線距離相等（皆為一時間單位），因此觸及終點狀態的同時也表示找到最短距離之路徑。但也因此特性，需探索所有同一距離之可能性，需耗費較多時間才能觸及終點狀態。

表 4.1 廣度優先搜尋探索流程

Breadth-First Search

Data: *oriGraph,Graphs*

```
1:  let NextList be a queue
2.  NextList.enqueue(ori.Graph)
3. While ( queue ≠ null ) do
4.      cur = NextList.dequeue()
5.      For neighbors n of cur in Graphs do
6.          If n is not visited then
7.              distance of n = distance of cur+1
8.              previous graph of n = cur
9.              If n in ending state then
10.                 Break loop
11.             mark n as visited
12.             NextList.enqueue(n)
13.     End for
14. End while
```

4.1.2深度優先搜尋探索

與廣度優先搜尋相似，深度優先搜尋從起始分佈狀態出發，探索相鄰節點 *n*，記錄最短距離與路徑，重複此步驟直到觸及終點狀態（表 4.2 為其程式流程）。相異處為深度優先搜尋會優先向更長距離探索，直到抵達終點狀態或超出時間上

限 *maxT*，才會回到待探索的其他節點進行下一輪深度探索，並且只探索「可減少最短距離」之節點。

深度優先搜尋可快速找到一條從起始狀態到終點狀態可行路徑，但需確保所有節點皆為最短距離，才能確認其為最佳路徑，因此此探索方式較廣度優先搜尋耗時。每次得到新的可行路徑時會更新時間上限 *maxT*（即每次得到的可行路徑會逐漸縮短），良好的初始時間上限設定可增進求解效率與準確性。本研究以「時間上限更新次數 *iterationNum*」為限，在效率與精確度中取得平衡。

表 4.2 深度優先搜尋探索流程

Depth-First Search

Data: *oriGraph,Graphs*

let *NextList* be a stack
NextList.push(*ori.Graph*)
set *iterationNum*=0, *maxT*
While (*stack* ≠ *null*) **do**
    cur = *NextList*.pop()
    If distance of cur +1 is less than *maxT* **then**
        For neighbors *n* of *cur* in *Graphs* **do**
            If distance of *n* is less than distance of *cur* +1 **then**
                distance of *n* = distance of *cur*+1
                previous graph of *n* = *cur*
                If *n* in ending state **then**
                    update *maxT*
                    iterationNum+=1
                    If *iterationNum* reach limitation **then**
                        Break loop
                NextList.push(*n*)
        End for
If *iterationNum* reach limitation **then**
    Break loop

4.2滾動式求解演算法

使用整數規劃數學模式求解時，隨系統規模提升與空格數減少，運算時間會大幅成長，因此本研究亦發展滾動式求解演算法，將系統中的格點分成數個規模較小之子系統，縮小每次數學模式計算之系統規模，進而大幅降低運算時間，但也因此容易忽略部分可行解，而無法取得最佳解。

表 4.3 為本演算法程式流程，根據給定各子系統範圍 *subRanges*，從原圖 *graph* 中擷取屬於該子系統之範圍 *subGraph*，使用數學模式求得子系統之最佳解後，再更新目前累計之總完工時刻 *Total_makespan* 與總運算時間 *Total_runtime*，並將子系統完成任務時之格點分佈狀態更新至原圖對應範圍，再求解下一個子系統。

表 4.3 滾動式求解流程

Rolling

Data: *graph, subRanges*

1. set *Total_makespan* = 0, *Total_runtime* = 0
2. **For** *subRange* in *subRanges* **do**
3. load *subGraph* from *graph* based on *subRange*
4. do *IP Model*
5. update graph, *Total_makespan*, *Total_runtime* based on the model result
6. **End for**

如圖 4.2 所示，在同樣高密度的條件下，需將灰色出貨推盤由左上角初始位置，運送至右下角之黑色揀貨點。數學模式求解 10×10 系統時，需耗費數十分鐘甚或是數小時的運算時間，但若將 10×10 之系統拆解成兩個黑框中之 6×6 系統（圖 4.2 之黑框中白色部分，x 處為子系統之出貨推盤目的地），即可在五分鐘甚或是更短的時間內結束運算。

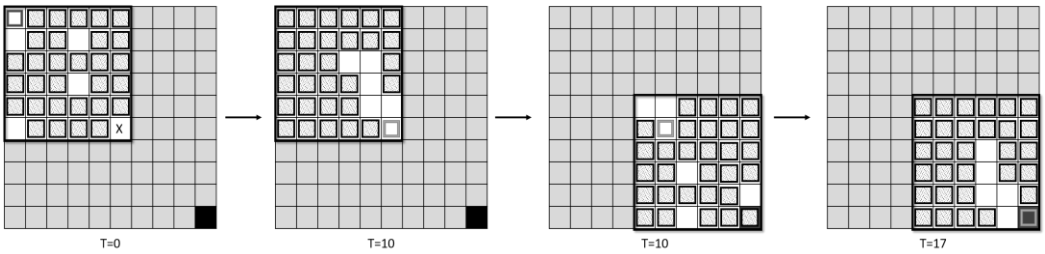


圖 4.2 滾動式求解示意圖

4.3小結

在動態規劃演算法中，展開分佈狀態網路圖，並以兩種探索方式從網路圖中找出最短路徑。廣度優先搜尋可保證以最短距離觸及終點狀態，但同時也需耗費較多時間才能抵達可行的終點狀態。深度優先搜尋能快速找到可行路徑，但需經過多次迭代改善求解品質。動態規劃演算法之效率容易受空格數與同一時刻允許同步移動的推盤數量上升影響。而滾動式數學模式求解可大幅減少運算時間，卻也因此可能無法取得最佳解，合適的初始空格位置分佈與系統分組範圍，可優化滾動式數學模式之求解品質。

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，包括求解規模、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，數學規劃模式則利用 Gurobi 9.3 版求解。

5.1測試資料與參數設定

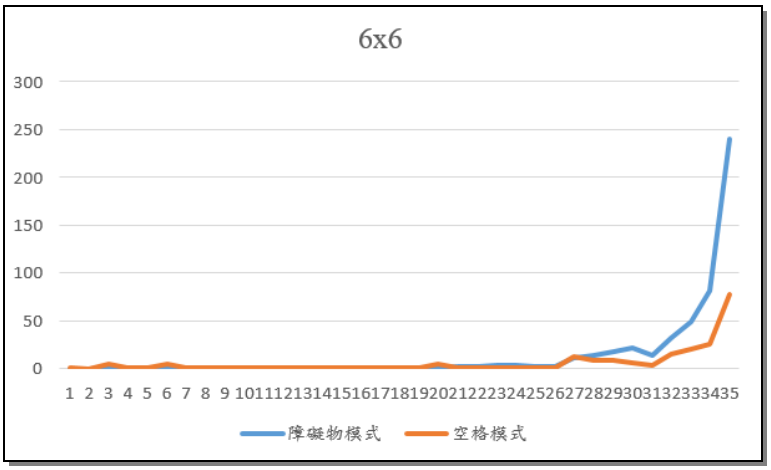
本研究根據給定參數產生測資，包含網格節點數 n 與系統中空格數 s，我們測試的網路圖皆為正方形之網格圖，因此節點數 n 皆為平方數；出貨推盤的起始位子固定為左上角，空格位置分為隨機組（隨機生成三十組取平均）與最差情形組（即最大化空格初始位置與出貨推盤距離，從右下角開始，依與出貨推盤之曼哈頓距離由遠而近排列，僅一組）。隨機組運算時間上限為 1200 秒，最差情形組運算時間上限為 10000 秒，超過時限之案例運算時間以「-」表示。最遲完工時刻極限 T 則根據 2.1 節提及之單一空格運送單一出貨推盤所需之最小時間公式來設定。

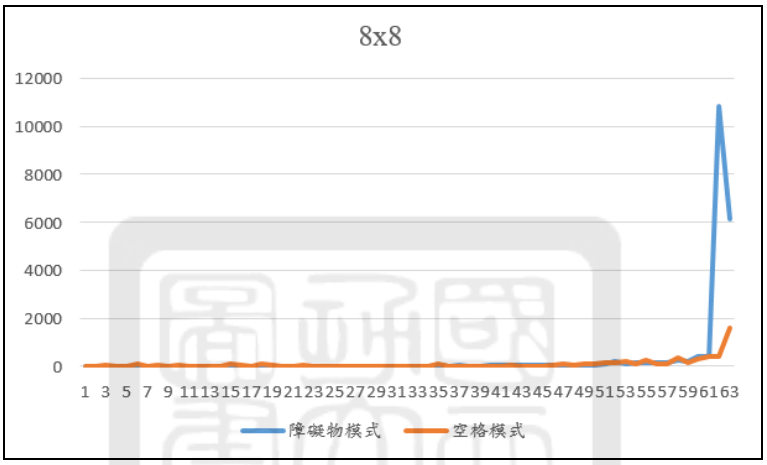
5.2兩數學模式求解結果與運算時間

表 5.1 與圖 5.1 中呈現兩數學模式在最差情形之求解結果與運算時間。由數據中可觀察到兩模式選出的最佳路徑具有相同的最遲完工時刻與移動次數，在不同空格數時卻有不同的求解效率。障礙物數學模式運算時間隨空格數減少而逐漸上升；空格數學模式的運算時間較浮動，但在空格數少時求解速度較快。

表 5.1 兩觀點數學模式最差情形求解結果

節點數 n	空格數 s	完工時刻	移動次數	障礙物數學模式運算時間(秒)	空格數學模式運算時間(秒)
36	35	10	10	0.031	0.119
	15	15	26	1.720	0.173
	10	16	35	2.478	0.943
	5	19	69	13.318	3.733
	3	24	52	48.920	20.251
	2	26	50	80.950	25.194
	1	37	37	240.561	77.377
64	63	14	14	0.082	0.326
	50	17	23	2.621	0.274
	40	19	31	6.637	2.527
	30	20	40	8.332	3.960
	20	22	54	21.200	0.375
	15	23	66	35.248	93.697
	10	26	75	135.145	79.706
	5	29	109	198.579	133.931
	4	31	94	391.127	276.437
	3	34	88	380.184	408.084
	2	38	74	-	388.092
	1	53	53	6135.824	1592.972
100	99	18	18	0.173	0.744
	80	22	32	11.700	4.641
	70	24	40	40.442	850.957
	60	25	49	54.588	784.053
	50	26	58	86.588	20.978
	40	27	69	86.003	731.738
	30	28	85	190.182	664.191
	20	30	109	373.314	582.880
	10	34	179	756.752	972.600
	5	39	149	2608.122	6668.803
	4	41	134	4048.809	3185.588
	3	44	124	2583.382	2498.841
	2	50	98	-	9630.219
	1	-	-	-	-





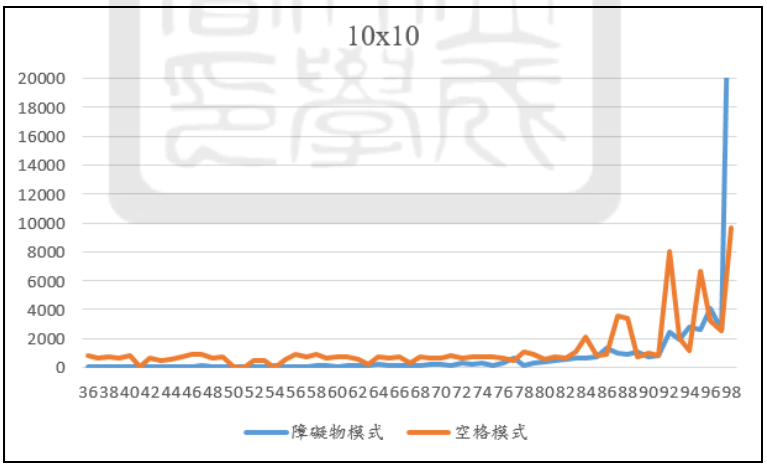


圖 5.1 兩數學模式在不同規模之空格數與執行時間比較
（x 軸為空格數，y 軸為運算時間）

表 5.2 呈現兩數學模式在隨機生成三十組空格初始位置之平均求解結果與運算時間。雖然在規模較大時，空格數學模式表現較不穩定，超出時間限制（1200 秒）的案例較多，且用時普遍較長，但可觀察到節點數 n=36 時，空格數學模式

運算效率有明顯優勢且用時夠少，因此 5.4 節中滾動式求解演算法的測試以 6×6 為子系統規模。

表 5.2 兩觀點之數學模式隨機三十組空格分佈求解結果

節點數 n	空格數 s	障礙物數學模式					空格數學模式				
		平均完工時刻	平均移動次數	運算時間		超時組數	平均完工時刻	平均移動次數	運算時間		超時組數
				平均(秒)	標準差				平均(秒)	標準差	
36	35	10.0	10.0	0.04	0.00	0	10.0	10.0	2.71	0.04	0
	10	11.0	22.5	1.53	0.59	0	11.0	22.5	2.97	0.77	0
	5	13.9	49.7	12.59	3.81	0	13.9	49.7	5.21	2.56	0
	3	18.0	41.5	39.91	12.82	0	18.0	41.5	19.86	9.81	0
	2	21.7	40.3	88.58	58.65	0	21.7	40.3	39.91	20.49	0
	1	31.7	31.7	231.05	81.88	3	31.6	31.6	69.02	59.32	2
64	63	14.0	14.0	0.09	0.01	0	14.0	14.0	0.36	0.03	0
	40	14.1	16.2	1.65	0.74	0	14.1	16.2	9.56	14.64	0
	20	14.6	27.8	7.10	2.67	0	14.6	27.8	86.14	35.59	0
	15	15.1	33.5	13.00	4.49	0	15.1	33.5	97.60	30.89	0
	10	17.7	52.6	118.46	46.49	0	17.7	52.6	252.64	181.94	0
	5	20.9	76.7	179.32	50.85	0	20.9	76.7	329. 75	148.66	0
	3	26.4	68.4	549.44	210.73	1	26.4	68.4	688.33	309.16	5
	2	31.2	60.4	636.31	252.85	16	32.7	62.7	729.52	317.65	12
	1	-	-	-		30	-	-	-		30
100	99	18.0	18.0	0.22	0.02	0	18.0	18.0	0.71	0.02	0
	70	18.2	19.5	6.22	3.17	0	18.2	19.5	129.10	176.86	0
	50	18.3	24.2	13.44	2.90	0	18.3	24.2	324.37	354.64	1
	40	18.3	27.0	20.16	7.01	0	18.3	26.8	676.34	255.93	2
	30	18.6	35.1	45.97	25.53	0	18.6	35.1	797.43	221.54	0
	20	19.4	50.5	138.20	78.00	0	19.36	50.0	822.67	160.25	5
	10	22.6	117.0	703.10	166.01	0	-	-	-	-	30
	5	26.0	109.0	987.07	70.35	28	-	-	-	-	30
	1~4	-	-	-	-	30	-	-	-		30

5.3數學模式與動態規劃演算法之比較

表 5.3 為數學模式與動態規劃演算法在最差情形之求解結果與運算時間比較，由於當同一時刻可移動的推盤數增加時，會使每單位時間可能擴展的狀態數大幅成長，因此演算法中僅允許同時最多移動兩個推盤。在空格個數少於二時，可求得與數學模式相同的最佳解，且在規模足夠大時，求解效率比數學模式佳。

表 5.3 兩觀點之數學模式與動態規劃最差情形求解結果

節點數	空格數	障礙物觀點數學模式		空格觀點數學模式		動態規劃演算法	
n	*s*	完工時刻	運算時間(秒)	完工時刻	運算時間(秒)	完工時刻	運算時間(秒)
36	3	24	48.92	24	0.17	25	190.70
	2	26	80.95	26	0.94	26	6.55
	1	37	240.56	37	3.73	37	0.18
64	3	34	380.18	34	79.71	36	408.08
	2	-	-	38	133.93	38	130.63
	1	53	6135.82	53	276.44	53	34.873
100	3	44	2583.38	44	2498.841	-	-
	2	-	-	50	9630.219	50	816.73
	1	-	-	-	-	69	97.15

5.4數學模式與滾動式求解演算法之比較

圖 5.2 為使用兩數學模式進行滾動式求解演算法之求解品質與運算時間比較，橫軸為空格數，縱軸分為為最遲完工時刻與運算時間（秒）。系統規模為 10×10，總格點數 n=100。初始推盤分佈隨機產生，並選取初始子系統含有至少一空格之案例，確保有可行解。數學模式滾動式求解之子系統規模固定為 6×6，共求解兩次，兩個子系統範圍之左上角座標分別為[1,1]與[4,4]，重疊範圍四格，包含目標推盤與至少一格空格。

由於求解範圍與分佈相同，因此兩數學在滾動式求解演算法之求解品質相同。從圖 5.2 中可觀察到，當空格足夠多時，求解品質較穩定；當空格數較少時，

演算法與最佳解之誤差較不穩定。因為最佳解需使用之空格數有限，多餘的空格對完工時間並無幫助，因此系統中的空格數足夠多時，子系統可分到足夠空格之機率愈大，因而求解結果為最佳解或是非常接近最佳解；然而，系統空格數較少時，空格分佈於子系統範圍之機率減小，因此子系統求解時能使用之空格數較浮動，演算法求解品質較不穩定。在運算時間的部份，當空格數極少時，所需求解時間大幅成長，而規模越大之系統成長幅度亦越大，因為 6×6 系統所需之運算時間遠低於 10×10 系統，因此滾動式求解演算法可明顯減少運算時間，而空格觀點數學模式在 6×6 系統運算時間穩定，且空格數極少時運算時間明顯較障礙物觀點數學模式少，因此此數值測試中，空格觀點數學模式之運算時間表現較佳。

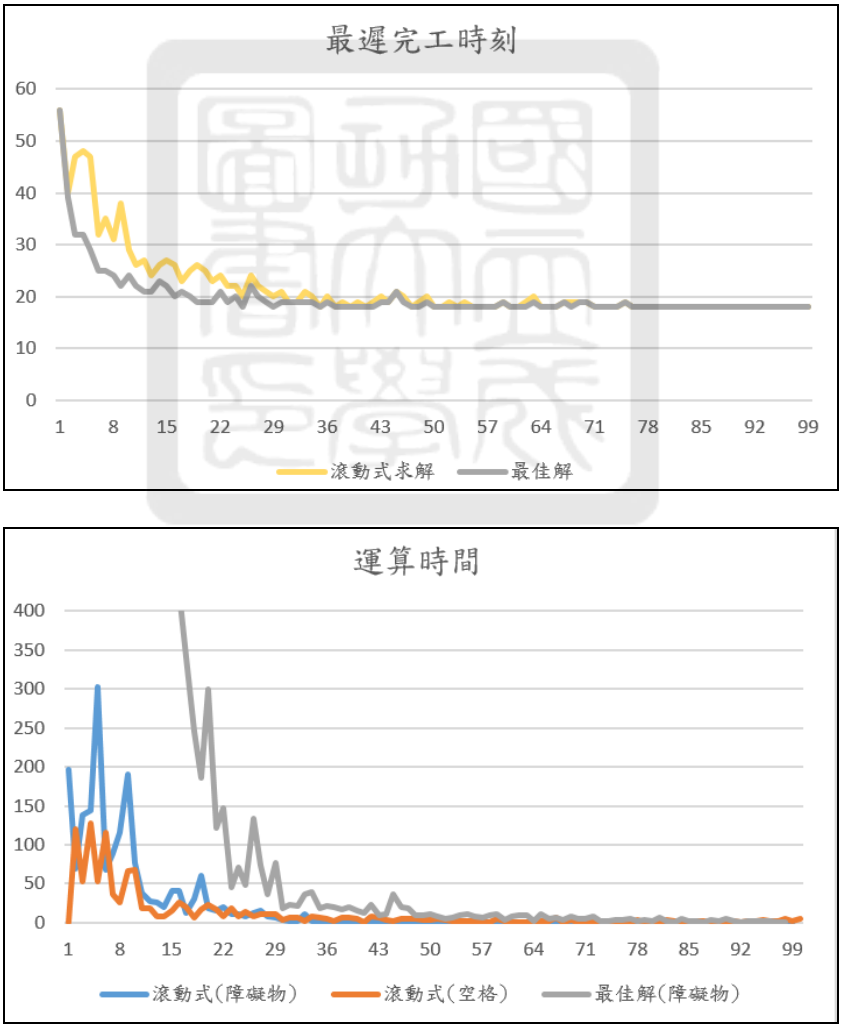


圖 5.2 滾動式求解之求解品質與運算時間
（x 軸為系統中空格數，y 軸為運算時間）

雖然本章節目前僅以 10×10 為案例進行數值測試，但滾動式求解演算法亦可應用於更大的案例，如圖 5.3 中所示，將 22×22 之系統劃分為以下五個 6×6 之子系統，並且安排合適的空格數在子系統中，此演算法可於短時間內計算出更大系統規模之最佳揀貨路徑。

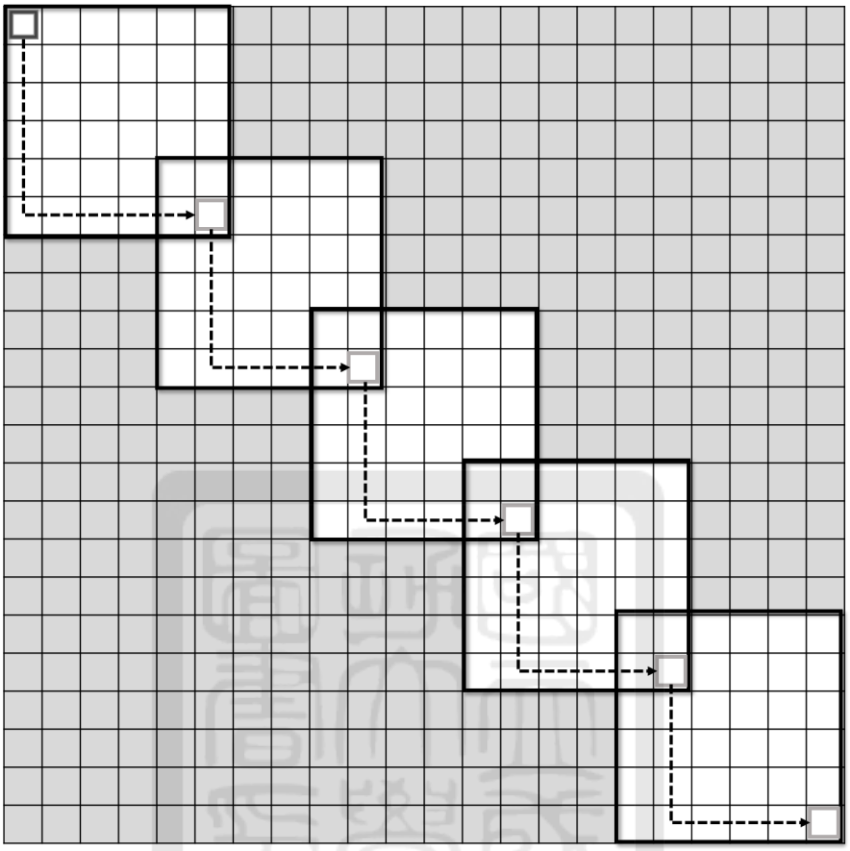


圖 5.3 滾動式求解大型案例示意（22×22）

5.5小結

本章節比較兩數學模式與兩演算法間之優缺點，障礙物數學模式在空格數多時求解效率高且運算時間穩定，空格數學模式在空格數較少時可節省運算時間，而動態規劃演算法在空格數一或二格時，可以高效率求得最佳解。以上三者皆可得到最佳解，但隨著系統規模擴大與空格數減少，需耗費大量運算時間，因此發展滾動式求解演算法，藉由劃分子系統的方式執行數學模式，以提升求解效率。由數值測試結果得知，滾動式求解演算法確實可大幅減少運算時間，但空格的位置分佈、子系統範圍劃分方式，以及子系統涵蓋之空格數，皆可能對運算時間與結果造成影

響，因此求解品質較不穩定。尤其在空格數較少時，若有空格未被涵蓋於子系統範圍（即空格被浪費，變相減少空格數），會與最佳解有較大誤差。擴大子系統範圍時雖然可改善求解品質，但運算時間亦隨之增加，因此需找到效率與求解品質之平衡點。未來將測試不同的系統與子系統之規模，與調整子系統間重疊範圍，觀察其對求解時間與效率之影響。

第六章 結論與未來研究方向建議

6.1結論與貢獻

由於電子商務蓬勃發展，商品品項日益豐富，儲存空間需求增加，需要更充分地利用倉儲中的閒置空間或走道空間。然而，當系統的密度提升時，揀貨路徑之規劃難度亦隨之提升，規劃時不僅需考慮出貨推盤至揀貨點的路徑，亦需考慮在因密度提升使空格與走道等運送空間被壓縮的條件下，如何應用有限的閒置空間協助出貨推盤移動，由於閒置空間的規劃邏輯並不直覺，因此規劃路徑相當複雜。為了提升高密度儲存方式在現實中的可行程度，本研究旨於發展規劃儲存系統揀貨路徑之數學模式與演算法，規劃倉儲系統中各推盤在各時刻移動路徑，在有限時間將出貨推盤移動至揀貨點。

雖然有許多探討揀貨路徑規劃相關議題之文獻，但大多囿於有走道之佈局模式，而探討推盤式倉儲系統的文獻中，大多存在以下三個問題：

1. 假設過於限縮，如系統中僅有單一出貨推盤、單一空格等。
2. 無法保證取得最佳解。
3. 未考量時間因素，僅可規劃「移動路線」而非「逐時位子變化」。

本研究針對以上三點進行調整，並依此建立兩數學模式與兩演算法。在假設方面，本研究允許以多個空格於有限時間內將多個出貨推盤運送至揀貨點，並且允許推盤中裝載多種商品，將每種商品組合視為一類推盤，可從各類推盤中挑選合適的推盤送至揀貨點完成訂單需求。使用整數規劃數學模式求解，可以取得最佳解。數學模式與演算法皆有依時空展開，因此可將「時間因素」納入考量，讓問題不只是「移動路線」而是「各推盤在各個時間點的位子變化」，因而可考慮避免碰撞、利用時間差提前移動以舒緩壅塞，並且探討「區塊位移」與「同步移動」之相關設定，計算確切等候時間等涵蓋時間考量的規劃。

由於障礙物觀點之數學模式求解效率有限，運算時間隨空格數增加而大幅成長，因此嘗試使用以空格為觀點之邏輯建置另一個數學模式，期望在空格數少時有較高的求解效率，在實測中雖然在空格數少時，確實較障礙物觀點之模式有效率，但仍需耗費大量時間，因而發展演算法以改善求解效率。

本問題在演算法設計上有兩大困難點，分別是「時間維度」與「空格應用」，具有時間維度才能知道是否碰撞與如何利用時間差排除阻礙，而推盤需藉由空格才能移動，卻無法將空格視為搬運工具─因為有推盤移動到空格上時，兩者位子會交換，而非空格載著推盤一起移動。雖然動態規劃演算法將各推盤分佈圖依時間建構網路圖，但隨著同一時刻可移動的空格數增加，網路之節線數亦大幅成長，因此求解效率也相當有限。

基於數學模式與動態規劃之效率問題，本研究發展滾動式求解演算法，將倉儲系統依範圍劃分為數個子系統（各子系統間範圍可重疊），使用數學模式求得第一個子系統之最佳解，將格點狀態依最佳解更新，再求解下一個子系統，重複此流程直到出貨推盤抵達揀貨點。此方法可大幅減少運算時間，但空格的位置分佈、子系統範圍劃分方式，以及子系統涵蓋之空格數，皆可能對規劃結果造成影響，因此求解品質較不穩定，空格數較少時易與最佳解有較大的誤差。增加子系統涵蓋範圍，雖然可改善求解品質，但運算時間亦隨之增加，因此需找到求解效率與求解品質之平衡點。

綜合以上敘述，本文主要貢獻為以下三點：

1. 考量更多元的倉儲系統設定

在問題設定上更符合現實，相對於文獻僅大多僅針對單一出貨推盤，以及少量空格之問題設定，本研究允許推盤含有一或多種商品，系統中有多個裝載相同或相異商品組合之推盤，可從包含出貨商品之推盤組合中，選取合適的推盤逐一送至揀貨點，以最小揀貨時間滿足訂單需求之商品。

2. 將時間維度納入考量之路徑規劃

相對於大多數文獻僅規劃「移動路徑」，本研究建構橫軸為時間單位、縱軸為空間格點、相鄰兩層的空間可移動連結為節線之「時空網路」，並依此網路發展混合整數規劃數學模式與演算法，因為將時間維度納入考量，可規劃出「各時刻之推盤位子變化」，而不僅只是規劃出無時間維度之移動路徑，因而可針對同時移動的推盤數與碰撞限制等進行更細緻的設定，除移動步數外，亦可求出完工所需時間。

3. 提出新的求解方法並進行數據測試

提出多種數學模式與演算法來處理推盤式倉儲系統揀貨路線規劃問題，皆有取得最佳解之能力，依空格數由多至少，運算效率依次為分別為障礙物觀點數學模式、空格觀點數學模式與動態規劃演算法；而滾動式求解演算法在合適的參數調配下，可以減省運算時間，動態規劃演算法亦可在短時間內取得可行解，但求解品質受最大時限設定影響。

6.2未來研究方向建議

本研究提出兩數學模式與兩演算法，其中數學模式與動態規劃演算法求解效率有限，而滾動式求解之求解品質不穩定，因此在求解方法上還有許多進步空間。在未來可嘗試不同的初始位置分配、調整各個推盤每次移動所需的耗能比例，與優化滾動式求解之子系統範圍劃分方式與預處理邏輯，例如：刻意選擇涵蓋較多空格的區域做為子系統範圍、將空格預先移動至子系統範圍中，或是允許使用子系統範圍外之空格等，觀察這些調整對各方法的求解結果與運算時間之影響，並從中發想優化數學模式與演算法之方向。問題延伸應用可思考如：三維的儲存空間，預先得知隔日取貨時程以預先對推盤位置進行排程等，皆是未來可能的發展方向。

參考文獻

Yalcin, A. (2017). *Multi-agent route planning in grid-based storage systems* (Doctoral dissertation, Europa-Universität Viadrina Frankfurt).

Taylor, G. D., & Gue, K. R. (2008). The effects of empty storage locations in puzzle-based storage systems. In *IIE Annual Conference. Proceedings* (p. 519). Institute of Industrial and Systems Engineers (IISE).

Gue, K. R. (2006). Very high density storage systems. *IIE Transactions*, 38(1), 79-90.

Gue, K. R., & Kim, B. S. (2007). Puzzle-based storage systems. *Naval Research Logistics (NRL)*, 54(5), 556-567.

Li, J. T., & Liu, H. J. (2016). Design optimization of amazon robotics. *Automation, Control and Intelligent Systems*, *4*(2), 48-52.

Mirzaei, M., De Koster, R. B., & Zaerpour, N. (2017). Modelling load retrievals in puzzle-based storage systems. *International Journal of Production Research*, 55(21), 6423-6435.

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.

Kota, V. R., Taylor, D., & Gue, K. R. (2015). Retrieval time performance in puzzle-based storage systems. Journal of *Manufacturing Technology Management*, 26(4): 582–602.

Bukchin, Y., & Raviv, T. (2021). A comprehensive toolbox for load retrieval in puzzle-based storage systems with simultaneous movements. DOI: 10.13140/RG.2.2.32086.98884

Yunfeng, M. A., Haoxun, C. H. E. N., & Yugang, Y. U. (2022). An efficient heuristic

for minimizing the number of moves for the retrieval of a single item in a puzzle-based storage system with multiple escorts. *European Journal of Operational Research*, 301(1), 51-66.