萬用啟發式演算法 期末程式實作報告

論文: An ant colony optimization based on local search for the vehicle routing problem with simultaneous pickup-delivery and time window

第 4 組 314704055 賴繹鈞、314704025 黎瑄宜、112701006 吳昭泓、314704042 柯卉翎

1. 前言

隨著電子商務和即時物流的需求越來越高，企業在安排每天的配送路線時，需要同時兼顧成本、效率和顧客的時間要求。車輛途程問題 (Vehicle Routing Problem, VRP) 成為物流計畫中最重要的議題之一。不過，實際的配送情況通常比傳統 VRP 更複雜，需要向顧客回收貨物，因此必須同時處理「送貨」和「取貨」兩種需求。

本研究所探討的 VRPSPDTW (考慮同時取貨、送貨與時間窗的車輛途程問題)。這個問題因為同時加入容量限制、時間窗限制、取送貨順序等多項條件，因此求解難度非常高，屬於 NP-hard 問題，文獻中也相對較少被討論。

根據論文中的文獻整理，過去處理 VRPSPDTW 的方法有基因演算法(GA)、模擬退火 (SA)、大型鄰域搜尋 (ALNS)、禁忌搜尋 (TS) 等啟發式方法，幾乎沒有看到用蟻群演算法 (ACO) 解決 VRPSPDTW 的研究。作者也在論文中指出，ACO 目前應用在 VRPTW 或 VRPSPD，卻沒有相關研究直接用在 VRPSPDTW，形成研究缺口。

此外，雖然 ACO 非常適合處理路徑搜尋類的問題，但傳統 ACO 有 positive feedback 的特性，容易陷入局部最佳，尤其在像 VRPSPDTW 這種限制更多、結構更複雜的情境中，效果可能不如預期。因此，作者提出了一個改良版本的 ACO，稱為 ACO-DR (Ant Colony Optimization with Destroy & Repair)。

這個改良演算法加入了兩個主要想法：

1. 方向性隨機轉移規則：把時間窗的緊迫程度、服務時間等因素納入螞蟻的決策，使螞蟻更聰明地選擇下一個服務對象。
2. 破壞與重建 (Destroy & Repair) 策略：藉移除部分節點再重新建構的局部搜尋機制，於協助演算法跳脫局部最佳、強化解的多樣性與探索能力。

2. 數學模式

2.1 集合與參數

符號	定義
$V = \{0,1,2,\dots,n\}$	所有客戶的集合
$V_0 = \{1,2,\dots,n\}$	車輛與所有客戶的集合, i.e. $V_0 = V \cup \{0\}$
$K = \{1,2,,,,.,m\}$	車輛的集合
σ	目標式中車輛派遣成本相對於行使成本的權重
g_d	每一台車的派遣成本
g_t	單位距離的行駛成本

C	車輛的最大容量
w_i	客戶 i 所需的服務時間，$\forall i \in V$
p_i	客戶 i 需要被載回車庫的貨物量，$\forall i \in V$
d_i	客戶 i 需要從車庫取得的貨物量，$\forall i \in V$
a_i	客戶 i 最早抵達時間，$\forall i \in V$;從車庫的最早出發時間，if $i=0$
b_i	客戶 i 最晚抵達時間，$\forall i \in V$; 從車庫的最晚抵達時間，if $i=0$
t_{ij}	節點 $i \in V_0$ 行駛至節點 j$\in V_0$ 的時間
c_{ij}	節點 $i \in V_0$ 行駛至節點 j$\in V_0$ 的距離

2.2 決策變數

符號	定義
x_{ijk}	=1 若車輛 $k \in K$ 由節點 $i \in V_0$ 行駛至節點 j$\in V_0$，否則為 0
S_i	車輛 $k \in K$ 抵達客戶 $i \in V$ 的時間
L_i	服務完客戶 $i \in V$ 後的車輛載重
$L^0{}_k$	車輛 $k \in K$ 離開車庫時的初始載重

2.3 目標式與限制式

$$\textit{Min.} \quad \sigma \sum_{j \in V} \sum_{k \in K} g_d \, x_{0jk} + (1-\sigma) \sum_{i \in V_0} \sum_{j \neq i \in V_0} \sum_{k \in K} g_t \, c_{ij} x_{ijk} \tag{1}$$

$$\textit{s.t.} \quad \sum_{i \neq j \in V_0} \sum_{k \in K} x_{ijk} = 1 \qquad \forall j \in V \tag{2}$$

$$\sum_{j \in V} x_{0jk} = 1 \qquad \forall k \in K \tag{3}$$

$$\sum_{i \neq h \in V_0} x_{ihk} = \sum_{j \neq h \in V_0} x_{hjk} \qquad \forall h \in V_0, k \in K \tag{4}$$

$$S_i + w_i + t_{ij} - M(1 - \sum_{k \in K} x_{ijk}) \leq S_j \qquad \forall i \in V, j \neq i \in V \tag{5}$$

$$a_i \leq S_i \leq b_i \qquad \forall i \in V \tag{6}$$

$$S_i \geq a_0 + t_{0i} \sum_{k \in K} x_{0ik} \qquad \forall i \in V \tag{7}$$

$$S_i \leq b_0 - (t_{i0} + w_i) \sum_{k \in K} x_{i0k} \qquad \forall i \in V \tag{8}$$

$$L^0{}_k = \sum_{j \in V} d_j \sum_{i \neq j \in V_0} x_{ijk} \qquad \forall k \in K \tag{9}$$

$$L_j \geq L^0{}_k - d_j + p_j - M\,(1 - x_{0jk}) \qquad \forall j \in V, k \in K \tag{10}$$

$$L_j \geq L_i - d_j + p_j - M\,(1 - \sum_{k \in K} x_{ijk}) \qquad \forall i \in V, j \neq i \in V \tag{11}$$

$$L^0{}_k \le C \qquad \forall k \in K \qquad (12)$$

$$L_j \le C \qquad \forall j \in V \qquad (13)$$

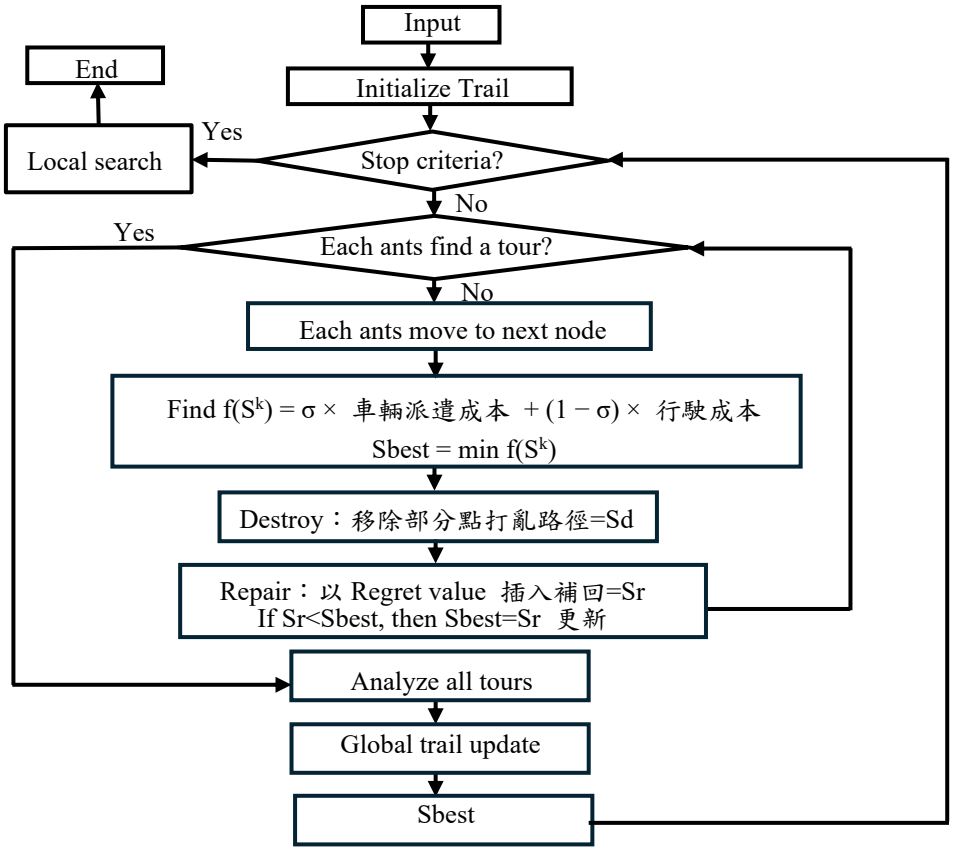
$$x_{ijk} \in \{0,1\} \qquad \forall i \ne j \in V_0, k \in K \qquad (14)$$

目標式 (1) 最小化車輛派遣與行駛成本的加權總合。限制式 (2) 限制每個客戶僅被拜訪一次，限制式 (3) 限制每台車僅可離開車庫一次，限制式 (4) 則確保流量守恆。限制式 (5) 依車輛路徑順序累積每個客戶的抵達時間。限制式 (6) 至 (8) 確保客戶與車庫的時間窗被滿足。限制式 (9) 設定各車初始載重為路線所有客戶從車庫取得貨物量總和。限制式 (10) 與 (11) 累積各路線載重。限制式 (12) 與 (13) 限制載重不可超過車輛最大容量。限制式 (14) 宣告 x_{ijk} 為二元變數。

首先，我們將論文 two-index 的 S_{ik} 改成更簡潔的 single-index S_i。其次，論文限制式 (11) 累積路線客戶載重，誤植 L_i 為$L^0{}_k$，我們將其更正。最後，論文列式未嚴格處理車輛需要於車庫最晚抵達時間前返回的限制；我們在限制式 (8) 與 (9) 顯式表達出來。

3. 蟻群演算法

1. Initialize Trail 設定
   演算法進行最多 200 次迭代，並設定 ACO 各種參數，含螞蟻的數量 (m)、費洛蒙重要度 (α)、距離重要度 (β)、時間範圍權重 (γ)、服務時間權重 (δ)、費洛蒙蒸發率 (ρ)、控制 exploration 與 exploitation 門檻 (r_0)、費洛蒙強度 (Q)、destroy 移除的客戶數 (L)、控制移除客戶相關性高低的參數 (D)。
2. 螞蟻根據隨機轉移規則構造路線（3.1 詳細說明）
3. 計算每隻螞蟻的總路線成本以及找出 Sbest
   目標式：$Min.\ \ \sigma \sum_{j \in V} \sum_{k \in K} g_d\, x_{0jk} + (1-\sigma) \sum_{i \in V_0} \sum_{j \ne i \in V_0} \sum_{k \in K} g_t\, c_{ij} x_{ijk}$
   本論文的目標是降低使用車輛以及總行駛成本最低，其中又以車輛數為首要降低目標，所以目標式中車輛權重 σ = 0.6 比總行駛成本高。
4. 破壞機制 (Destroy)：
   移除目前最佳解 (Sbest) 的部分節點，打亂部分結構得到 Sd。(3.4 詳細說明)
5. 修復機制 (Repair)：
   再把被拿掉的節點依照 regret value 重新插入補回得到 Sr。
   If Sr < Sbest, then Sbest = Sr 更新。(3.5 詳細說明)
6. 重複步驟直到完成 200 次的迭代
7. 在好的路徑上增加費洛蒙濃度(3.3 詳細說明)
8. 透過 Local Search 局部微調
9. 得到最佳解（成本最低的路徑）結束演算法



3.1 隨機轉移規則 (Random Transfer Rule)

Random Transfer Rule 是每隻螞蟻在構造路線過程中，如何選擇下一個拜訪客戶的機制。假設螞蟻目前位於客戶 i，針對下一位拜訪的候選客戶 j，本篇論文定義其分數為

$$Score_j = {\tau_{ij}}^{\alpha}(t)\,{\eta_{ij}}^{\beta}(t)\,(\frac{1}{width_j})^{\gamma}\,(\frac{1}{wait_j})^{\delta}$$

$\tau_{ij}(t)$ 為邊 (i, j) 在時間 t 的費洛蒙量，費洛蒙量多的邊常出現在先前迴圈最佳解中，因此值得 Exploit。啟發式資訊$\eta_{ij}(t)$ 定義為 (i,j) 距離的倒數，表示距離近的節點值得接續拜訪。最後兩項分別為客戶 j 時間窗寬度與服務時間的倒數，論文認為時間窗寬度大表示不急迫，而服務時間長表示車輛易在此擱置，而不希望成為下一個拜訪節點。

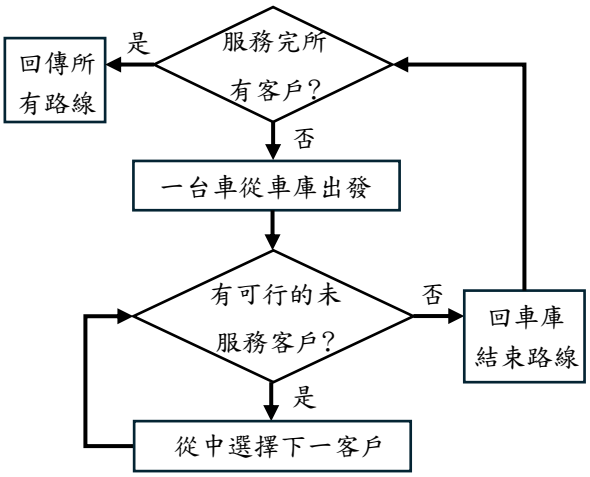
每次螞蟻決定下一位客戶時，會藉由一個 0 到 1 的隨機亂數 (r)，決定 Explore 或 Exploit，公式如下。

$$j = \begin{cases} arg\ max\,\{Score_j\}, r < r_0 \\ Select\ by\ Roulette\ Wheel, with\ probability\ p_{ij} = \dfrac{Score_j}{\sum_k Score_k}, otherwise \end{cases}$$

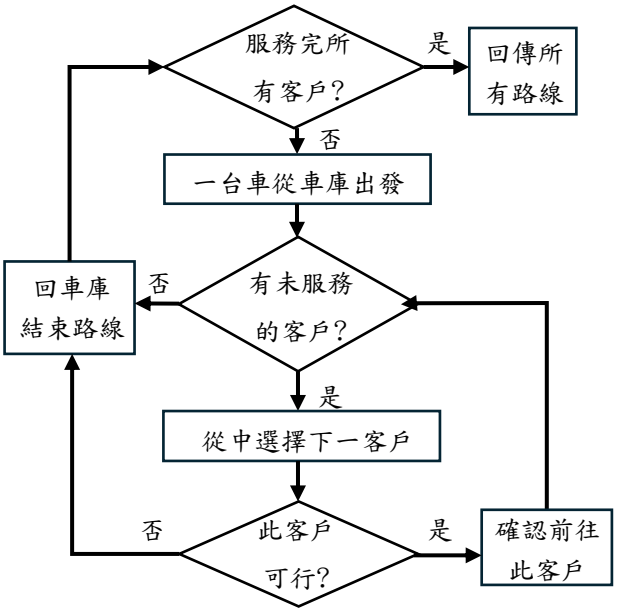
然而，論文並未詳細說明遇到不可行解的處理方式，僅說明演算法只接受可行解，沒有設計對應的不可行懲罰機制。我們查閱了 VRPTW、VRPPD 與 VPRSPDTW 的相關文獻，發現有兩種可能的執行流程。

第一種流程為 Molina et al. (2020) 所採用。一開始螞蟻從車庫開一台車出發，此時若有未服務且可行的客戶，則對上述客戶計算分數，並以亂數決定 Explore 或 Exploit；若未服務客戶都不可行，則螞蟻立即返回車庫，重新開另一台車出發。演算法執行直到所有客戶均被拜訪後結束。基於此流程，我們額外發想新的分數計算方式，改以「客戶最晚抵達時間 (時間窗上限)−車輛的預計抵達時間」取代時間窗寬度，來衡量急迫程度。

第二種流程為 Wu et al. (2023) 所採用。與上述不同的是，螞蟻每次針對所有未服務客戶計算分數，選擇下一位客戶後，再檢查其可行性。若可行則確定拜訪，若不可行則立即返回車庫，重新開另一台車出發後再重新選擇。此演算法觀點是：若每次只從可行客戶挑選，滿載的車輛為了騰出空間，可能選擇較遠但卸載貨物量較多的客戶。



流程圖一：第一種流程



流程圖二：第二種流程

3.2 可行性檢查

論文並未詳細說明可行性檢查的演算法實作方式，但由數學模式可知，需檢查抵達客戶的時間是否在時間窗內，以及車輛載重是否未超過最大容量限制。我們希望能夠以局部更新的方式實作，避免每次 Random Transfer，都重新計算時間與累積載重。

針對時間，我們只要以 S_i 紀錄目前末端客戶 i 的抵達時間。當嘗試拜訪下位客戶 j，則依照上式 (5) 檢查 $S_j = S_i + w_i + t_{ij}$ 是否在客戶 j 的時間窗內；然而，還需檢查服務完客戶 j 後立即回到車庫，是否可在其最晚抵達時間前返回，因若加入 j 導致回程超時，則繼續拜訪其他客戶也必然無法準時返回車庫，整條路線無論如何都不可行。

車輛載重限制比較複雜。當嘗試拜訪下一位客戶 j 時，由上式 (9) 可知：客戶 j 的卸載量 d_j 需要在車庫就裝載到車輛上，直至抵達客戶 j，此段路程的車輛載重皆須更新。我們於是以兩個變數 *bottleNeck* 與 *currentLoad*，分別記錄車輛從車庫到客戶 i 的最大與服務完畢後目前載重，皆初始化為 0。當欲拜訪下一位客戶 j 時，需執行以下步驟。

1. $bottleNeck \leftarrow bottleNeck + d_j$，並檢查是否超過車輛最大容量。
2. $currentLoad \leftarrow currentLoad + p_j$
3. $bottleNeck \leftarrow max\{bottleNeck, currentLoad\}$，再次檢查是否超過車輛最大容量。

以下我們將以數學歸納法證明上述演算法可確保路線可行性。

a. **induction basis**：當僅有一個客戶，bottleNeck＝車輛從車庫到此客戶的載重＝此客戶的卸載量；服務完後的目前載重為此客戶的裝載量。
b. **induction hypothesis**：假設此演算法對 n 個客戶的路線成立 ($bottleNeck$ 與 $currentLoad$ 都計算正確，且路線的車輛載重可行性判斷亦正確)。
c. **induction step**：欲拜訪第 $n+1$ 個客戶 (不失一般性假設已拜訪的客戶為 $1, 2, ..., n$，服務此客戶前，每條邊載重分別為 $L_{01}, L_{12}, ..., L_{n-1,n}, L_{n,n+1}$，$currentLoad^{old} = L_{n,n+1}$)。
   1. 考慮此客戶卸載量，因為從車庫到此客戶的每條邊都增加 d_{n+1}，$bottleNeck^{new} \leftarrow max\{L_{01}+d_{n+1}, L_{12}+d_{n+1}, ..., L_{n,n+1}+d_{n+1}\} = max\{L_{01}, L_{12}, ..., L_{n,n+1}\}+d_{n+1} = bottleNeck^{old}+d_{n+1}$。若 bottleNeck 未超過車輛容量，每條邊載重也必定未超過。
   2. 考慮裝載量 $currentLoad^{new} \leftarrow (L_{n,n+1}+d_{n+1}) - d_{n+1}+p_{n+1} = currentLoad^{old} + p_{n+1}$
   3. 服務完此客戶後，$bottleNeck^{after} \leftarrow max\{bottleNeck^{new}, currentLoad^{new}\} = max\{L_{01}+d_{n+1}, L_{12}+d_{n+1}, ..., L_{n-1,n}+d_{n+1}, L_{n,n+1}+p_{n+1}\}$，此即為 $bottleNeck$ 定義。同理，若 bottleNeck 未超過最大容量，每條邊的載重也必定未超過。

3.3 費洛蒙更新

原論文的費洛蒙更新方式即是採用傳統 ACO 的蒸發與補強機制。首先，所有邊的費洛蒙會依據蒸發率ρ進行衰減，使整體費洛蒙量變為$(1-\rho)\tau_{uv}$，以避免過度累積而集中於少數邊，保持演算法的探索能力。接著，針對目前找到的最佳可行解S^{best}，計算其路徑長度$L(S^{best})$，並依據下式產生費洛蒙增量：

$$\Delta\tau = \frac{Q}{L(S^{best}) + \varepsilon}$$

其中費洛蒙總量Q為常數、實作上則加入極小值ε為防止除以零。之後，將費洛蒙增量$\Delta\tau$加到S^{best}中所有邊(u, v)上，使下一輪的螞蟻更傾向選擇這些相對較佳的路徑。本次程式實作即先進行矩陣整體蒸發，再沿著最佳路線逐邊累加$\Delta\tau$，完成費洛蒙更新。

3.4 破壞 (Destroy)

破壞的目的在於自現有解中移除一部分具有高度相關性的客戶，以製造局部「空洞」，讓後續的重建階段能重新配置，從而跳脫局部最佳。此方法的核心概念是「相關度」：若兩位客戶距離相近，或原本位於同一路線（同一台車輛）上，則視為高度相關，在破壞時較有可能被一起移除。在論文中採用的兩客戶i與j的相關度定義為：

$$R_{ij} = \frac{1}{C'_{ij} + V_{ij}}$$

其中C'_{ij}為標準化後的距離；若兩者在同一條路線（同一車輛），則$V_{ij}=0$，否則$V_{ij}=1$。因此，距離越近或同車者，其相關度越高。

具體操作流程如下：首先，從所有客戶中隨機選取一名作為種子，加入移除集合R。如R的人數尚未達到設定的移除量L，則持續重複以下步驟：自R中隨機挑選一名客戶r，計算r與所有尚未移除客戶集合U的相關度，並由高到低進行排序。而後利用參數D，自序列中選擇關聯性第random$(0,1)^D$作為下一名移除客戶。當D越大，其結果越接近 0，也意味著越容易挑選到相關度較高的客戶。此過程將重複直到移除L名客戶為止。最終，將集合R中的客戶自所有路線中刪除，得到破壞後的新解$S^{destroy}$。

3.5 重建 (Repair)

重建階段負責將被移除的客戶逐一插回既有路線或建立新路線，以恢復完整可行解。該論文採用基於後悔值（regret value）的插入策略。針對每一個待插入客戶，計算其在所有可行位置的插入成本（Insertion Cost，即目標函數值）之變化，同時，因「開啟新路線」亦屬可行位置，故也將其列為可選方案之一。

計算完成後，將每位客戶「最佳」與「次佳」插入位置之目標值差異定義為後悔值，越大代表若不立即插入，未來將付出更高成本，故優先處理，將其插回最佳（目標函數值最小）位置；若存在客戶僅有一個可行插入位置 (即開新路線)，其後悔值設定為 (-1)，也就是此客戶優先級最低，等待後續迴圈是否有轉變為可行的可能。每次迭代皆選擇後悔值最大的顧客插入，並持續更新路線，直到所有移除顧客被放回為止，得到重建後的新解S^{repair}。若此新解S^{repair}優於原S^{best}，則會更新成為新S^{best}，取得更高品質可行解。

4. 數據實驗

我們使用的資料集來自 https://github.com/senshineL/VRPenstein (Liu, 2021)，與本篇論文所採用的資料集 (Wang and Chen, 2012) 相同。資料欄位如表 1，客戶數量為 100，每台車載重限制為 200 單位。

表一：資料集欄位名稱與解釋

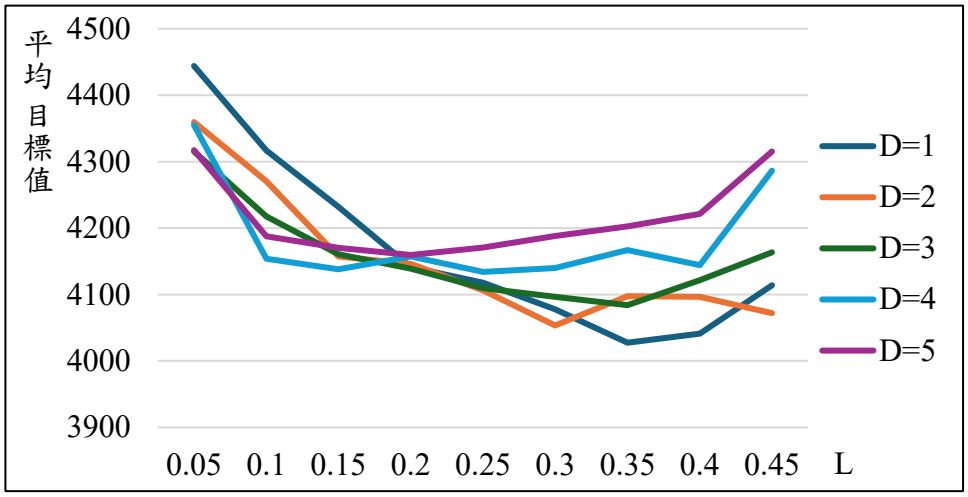
英文欄位	中文欄位	英文欄位	中文欄位
CUST NO.	客戶代號	PDEMAND	逆向需求
XCOORD.	X 座標	READY TIME	最早開始服務時間
YCOORD.	Y 座標	DUE DATE	最晚開始服務時間
DDEMAND	客戶需求	SERVICE TIME	服務時間

4.1 論文未說明的超參數調校

L 為因應不同大小資料集而設計的破壞比例，本研究將 L 定義為被移除樣本數占總樣本數的比例，並設定超參數搜尋範圍為 0.05 至 0.45，間距為 0.05；D 則設定為 1 至

5，間距為 1。為降低隨機性影響，本研究對每組 (L, D) 超參數組合皆進行 10 次重複實驗並取其平均目標值進行比較。

根據圖一，在 D＝1 下，L＝0.35 明顯取得所有組合中最低的平均目標值，為本研究整體表現最佳的超參數設定，故本研究後續實驗皆採用 L＝0.35 與 D＝1 進行測試。



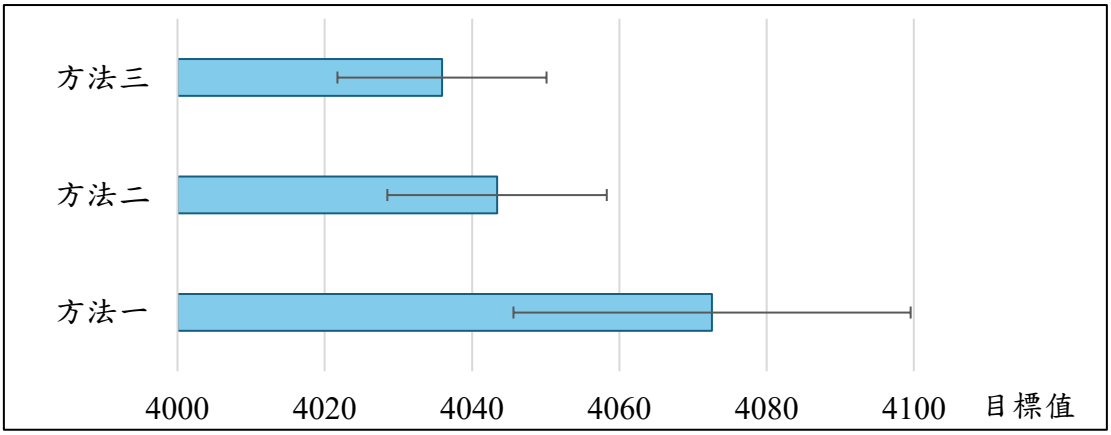
圖一：L 與 D 的參數調校結果

4.2 不同 Random transfer 規則和分數計算的結果比較

根據上文，共有三種 Random transfer 規則和分數計算方法，如表二所示。三種方法進行 10 次重複試驗，方法三在平均表現與穩定性上皆優於方法一與方法二，為最適合的方式。後續實驗因此採用方法三作為本研究的 Transfer 機制和分數計算。

表二：不同規則與分數計算說明

方法一	Random Transfer 第一個流程+論文分數計算
方法二	Random Transfer 第二個流程+論文分數計算
方法三	Random Transfer 第一個流程+時間急迫度分數計算



圖二：不同規則與分數計算的平均目標值

4.3 各資料集的實驗結果

本節將資料集依時間窗長短分為兩組：表三為時間窗較短的資料集，表四則為時間窗較長的資料集。兩表分別比較 Gurobi 與本研究所提出之 ACO-DR 演算法在使用車輛數、 總行駛距離及運算時間上的表現。

表三：Type-1 資料集 Gurobi 與 ACO-DR 結果比較

資料集	Gurobi			自行實作 ACO-DR			論文 ACO-DR	
	車輛	總距離	時間	車輛	總距離	時間	車輛	總距離
rdp101	19	1650.8	12.59	19	1669.9	87.57	20	1650.06
rdp102	-	-	3600+	18	1536.45	113.25	18	1462.12
rdp103	-	-	3600+	15	1305.45	86.82	14	1227.62
rdp104	-	-	3600+	10	1048.83	100.82	10	1003.58
rdp105	17*	1491.67*	3600+	16	1440.18	107.26	14	1383.06
cdp101	12*	966.84*	3600+	11	994.79	109.74	11	970.30
cdp102	-	-	3600+	11	940.32	120.73	10	964.56
cdp103	-	-	3600+	11	1079.95	134.75	10	913.46
cdp104	-	-	3600+	10	1100.24	146.01	10	1043.21
cdp105	11*	1028.38*	3600+	11	1013.46	112.66	11	989.59
rcdp101	16*	1689.76*	3600+	16	1708.8	57.64	15	1658.58
rcdp102	-	-	3600+	15	1624.5	60.73	13	1531.25
rcdp103	-	-	3600+	13	1447.27	66.05	12	1312.07
rcdp104	-	-	3600+	11	1230.27	71.94	11	1183.80
rcdp105	-	-	3600+	15	1623.22	62.98	14	1569.55

時間單位:秒

表四：Type-2 資料集 Gurobi 與 ACO-DR 結果比較

資料集	Gurobi			自行實作 ACO-DR			論文 ACO-DR	
	車輛	總距離	時間	車輛	總距離	時間	車輛	總距離
rdp201	-	-	3600+	6	1264.52	79.79	5	1236.13
rdp202	-	-	3600+	6	1196.88	93.45	4	1098.90
rdp203	-	-	3600+	4	1038.57	108.54	3	946.77
rdp204	-	-	3600+	3	831.75	140.55	3	771.89
rdp205	-	-	3600+	4	1086.01	105.19	3	1002.98
cdp201	3	591.56	3.83	3	591.56	129.45	3	591.56
cdp202	3*	591.56*	3600+	3	591.56	167.66	3	591.56
cdp203	4*	635.5*	3600+	3	600.21	187.03	3	591.17
cdp204	-	-	3600+	3	608.94	257.9	3	590.60
cdp205	3	588.88	838.07	3	588.88	263.01	3	588.88
rcdp201	-	-	3600+	6	1404.58	108.56	4	1383.79
rcdp202	-	-	3600+	6	1284.49	100.2	4	1197.36
rcdp203	-	-	3600+	4	1066.17	125.59	4	984.59
rcdp204	-	-	3600+	4	888.88	219.3	3	809.53
rcdp205	-	-	3600+	6	1265.15	153.65	4	1370.54

時間單位:秒

註：Gurobi 求解時間上限設定為 3600 秒，表中標記–之案例，表示時間內未能找到可行解；*表示時間內已找到可行解，但未能證明最佳解 (Gap > 0)；未標記者則為在時限內成功找到理論最佳解 (Gap = 0)。

4.4 結論

本研究所探討的 VRPSPDTW 問題因包含時間窗、同時取送、容量限制等多項約束，屬於典型的 NP-hard 問題。在有限計算時間內，透過本研究提出的 ACO-DR 演算法，能有效取得品質良好的可行解。

本研究針對 Random Transfer、分數計算方式與篩選超參數後，對 30 組資料集進行比較。Gurobi 在求解時間限制 3600 秒內僅能對其中 8 組產生可行解，其餘案例皆無法於時限內證明可行。在成功求解的少數案例中，ACO-DR 與 Gurobi 解品質相近；此外，除 cdp201 與 rdp101 外，其餘資料集 ACO-DR 均能在更短的運算時間內取得品質更佳的可行解，顯示本研究方法在 VRPSPDTW 問題上具備快速收斂與產生高品質解的能力。

5. 參考文獻

Liu, S., Tang, K., & Yao, X. (2021). Memetic search for vehicle routing with simultaneous pickup-delivery and time windows. *Swarm and Evolutionary Computation, 66*, 100927. https://doi.org/10.1016/j.swevo.2021.100927

Molina, J. C., Salmeron, J. L., & Eguia, I. (2020). An ACS-based memetic algorithm for the heterogeneous vehicle routing problem with time windows. *Expert Systems with Applications, 159*, 113379. https://doi.org/10.1016/j.eswa.2020.113379

Wang, H. F., & Chen, Y.Y. (2012). A genetic algorithm for the simultaneous delivery and pickup problems with time window. *Computers & Industrial Engineering, 62* (1), 84–95. https://doi.org/10.1016/j.cie.2011.08.018

Wu, H., & Gao, Y. (2023) An ant colony optimization based on local search for the vehicle routing problem with simultaneous pickup–delivery and time window. *Applied Soft Computing, 139*, 110203. https://doi.org/10.1016/j.asoc.2023.110203

Wu, H., Gao, Y., & Wang, W. (2023). A hybrid ant colony algorithm based on multiple strategies for the vehicle routing problem with time windows. *Complex & Intelligent Systems, 9*, 2491–2508. https://doi.org/10.1007/s40747-021-00401-1