國立台東大學

資訊管理學系

115(級)畢業專題報告書

適用於 Solidity 智能合約之多對多配對演算法比較與實作分析

指導教授:廖國良

學生:周芮莛(11112136)

中 華 民 國 114 年 5 月

誌謝

本研究得以順利完成，首先誠摯感謝廖國良教授。在專題執行的過程中，老師始終給予明確的階段性目標與進度規劃，透過定期的進度檢查，確保研究能按部就班地推進：而在研究遭遇瓶頸以及感到迷惘時，老師亦總能適時給予明確的方向與指引，協助我釐清思緒、排除困難。

也感謝資管系諸位師長三多年來的教導，並培養了我獨立思考與解決問題的專業能力，讓我有能力面對並克服研究過程中的種種挑戰。同時，也感謝家人與朋友在求學期間給予的無條件支持與包容，讓我能全心投入於學業與專題製作。

本專題之完成，除個人之努力外，更有賴於師長與親友的全力支持，最後，再次向所有在研究過程中提供協助與指導的人員致謝。因為有您們的支持與鼓勵，本專題才能順利完成。

摘要

本研究旨在探討多對多配對演算法於 Solidity 智能合約中的實作可行與效能比較。選擇 Gale-Shapley(GS)穩定婚姻演算法與 Top Trading Cycle(TTC)交換循環演算法作為核心比較對象，旨在評估並比較兩者在鏈上環境的實際表現、技術限制與應用適配性。分別設計與實作兩種演算法的 Solidity 智能合約，透過 Remix 與 Hardhat 等開發與測試工具，針對幾項關鍵指標進行功能驗證與效能評估。

研究結果顯示，在效能測試中(n=3)，TTC 演算法所消耗的 Gas 與執行時間均低於 GS 演算法，顯示 TTC 具備較高的鏈上執行效率。然而，在實作複雜度上，GS 的邏輯較為清楚、難度普通，而 TTC 因需判斷封閉循環，實作難度較高。整體而言，本研究證實了將此類配對演算法導入智能合約的可行性。GS 演算法可保障配對穩定性，適用於求職媒合、志願填報等情境；TTC 演算法則因 Gas 消耗較低，更適用於資源輪轉與數位資產互換等重視鏈上效率的應用。

關鍵字： 智能合約、Solidity、多對多配對、Gale-Shapley 演算法、Top Trading Cycle 演算法、Gas 消耗

目錄

圖目錄

表目錄

第一章　緒論

此章節為本專題計畫書之緒論，內文撰述研究背景、研究動機、研究目的、研究步驟的詳細規劃。

1.1 研究背景

2008 年，Satoshi Nakamoto 提出了比特幣（Bitcoin）的運作協定與相關技術[21]，掀起了加密貨幣與區塊鏈發展的浪潮。比特幣改變了傳統貨幣的交易模式，也引入了區塊鏈作為核心技術，其去中心化、密碼學加密、共識機制與時間戳記等方式，確保資料的安全性、透明性與不可篡改性[3]。

近年來，區塊鏈技術已從最初的加密貨幣領域逐步擴展，成為許多創新應用的基礎平台。在 2015 年以太坊（Ethereum）的推出，標誌著區塊鏈技術進入 2.0 階段，其核心在於引入智慧合約（Smart Contracts）的概念[4,14,24]。智能合約（Smart Contracts）是儲存在區塊鏈上的自動執行協定，其條款直接寫在程式碼中，允許在滿足特定條件時，無需第三方中介即可自動且可信任地執行預設的動作或協議。這種特性極大的擴展了區塊鏈的功能，使去中心化的應用更加廣泛，開發者得以在區塊鏈上設計出可自動執行的協定，將區塊鏈的應用範圍從單純的價值轉移拓展到更廣泛的協作與自動化情境。智能合約的出現，進而衍生出供應鏈追蹤、數位身分認證、版權保護、投票系統、醫療資料管理等多元化的應用[1,20]。

去中心化（Decentralization）作為區塊鏈技術最核心的優勢，更是為其在非金融應用中帶來更多價值[21]。在區塊鏈架構下，資訊的儲存與驗證不再需要依賴單一中心機構，而是可以透過分散在各節點間的共識機制來保障資料的真實性與完整性，智能合約正是實現這種去中心化應用邏輯的關鍵機制。這樣的特性讓區塊鏈特別適用於需強調信任、公平與抗審查性的情境，舉例來說:投票系統能減少選舉舞弊的風險、創作者作品版權認證、供應鏈管理等[20]。

區塊鏈的未來發展不只拘泥於金融交易，目前已經發展出許多於不同領域的應用，顯示區塊鏈作為一種資訊技術，正逐漸深入我們日常生活中各種需要信任機制的場景，智能合約正是驅動這一轉變和實現

這些創新應用模式的核心動力之一，擁有持續發展的潛力。

1.2 研究動機

隨著社會發展與科技演進，日常生活中出現越來越多需要解決的配對與分配問題，這些需求多數具備多對多的特性，例如:群眾募資平台的項目與資金方配對、學生實習機會分發、醫療器官移植的資源對應等。

儘管配對演算法在數理與經濟領域已發展成熟，當今在區塊鏈上的實際應用有限，尤其在智能合約層面，對於演算法的選擇與實作效能仍缺乏深入探討。

區塊鏈技術以其去中心化、匿名性與交易確認機制等優勢，在資料記錄與系統安全性方面有很大的發展空間，若能將配對演算法與智能合約技術結合，不僅可提升配對過程的透明化與信任度，也有助於建立公平且可自動執行交易的環境。

1.3 研究目的

基於上述研究動機，本研究主要目的為探討幾種比較知名的的多對多配對演算法於智能合約中的可行性以及效率。

透過對不同配對演算法的實作與分析，多方面評估各演算法在執行時所面臨的實際挑戰與技術限制，包含資料結構設計的複雜度、Gas 成本消耗[12]、配對穩定性、公平性、以及系統的可擴展性等關鍵指標。

1.4 研究步驟

本研究流程詳細說明如下:

1. 建立研究方向

   初步訂定研究主題，確認探討探討幾種比較知名的多對多配對演算法於智能合約中的可行性以及效率。

2. 進行文獻探討

   蒐集並整理智能合約以及多對多配對相關之理論與應用文獻，包含配對機制設計、各種代表性演算法的工作原理與適用情境，以及區塊鏈與智能合約在資源分配、資料管理、去中心化應用等領域的發展現況與挑戰，作為研究的理論依據與實作參考。

3. 技術分析

   針對區塊鏈技術架構與 Solidity [22]智能合約的開發環境進行分析，探討在鏈上實作配對演算法時可能遇到的關鍵技術議題，包括資料結構設計、交易成本（Gas）控制[12]、執行流程管理與狀態儲存效率等，並預先考量可能的技術限制。

4. 合約實作

   根據前面的分析結果，選定幾種具有代表性的多對多配對演算法進行智能合約實作，撰寫合約，模擬各種配對邏輯於區塊鏈環境中的實際運作方式，驗證不同演算法在智能合約中的可實作性與執行效率，並根據實驗結果進行其效能與資源使用之分析。

5. 資料彙整

   整合文獻探討、技術分析、合約實作內容，進行多面向的比較，針對各個配對演算法在區塊鏈環境下的實作可行性、效率表現、公平性進行總結，並歸納出適合於區塊鏈交易應用的配對演算法，提出可能的應用場景與未來的研究方向，作為後續研究的實際參考依據。

第二章 文獻探討

本章內容為相關文獻的探討,主要包括區塊鏈、乙太坊、智能合約、多對多配對。

2.1 區塊鏈

2.1.1 區塊鏈的起源

區塊鏈技術的起源可以追溯到 2008 年,中本聰(Satoshi Nakamoto)發表的比特幣白皮書,比特幣作為第一個應用區塊鏈技術的加密貨幣,它的誕生標誌著去中心化和分散式帳本技術的開始[21]。區塊鏈是藉由密碼學與共識機制等技術建立與儲存龐大交易資料串鏈的對等網路系統[3]。按照時間順序將數據區塊以順序相連的方式組合的一種鏈式數據結構,每一個區塊包含了前一個區塊的加密雜湊、相應時間戳記以及交易資料,這樣的設計使得區塊內容具有難以篡改的特性[4,5,21]。

2.1.2 區塊鏈的主要特性

基於時間戳記的鏈式區塊結構,分布式節點的共識機制、基於共識機制的激勵機制和靈活可編程的智能合約,是區塊鏈技術最具代表性的創新點[8]。以下為其三個主要特性。

1. 去中心化

   消除單一節點故障的風險,因為數據被分散在各個節點中,攻擊者需要攻擊多個節點才能破壞系統,使得系統具備可信性與安全性。但因其系統結構較為複雜,執行速度可能較慢,且交易成本較高[21]。

2. 不可竄改

   區塊鏈中的每一筆資料一旦寫入後,就不可再更動,只要資料被驗證完就永久的被寫入,其中的技術是透過雜湊演算法,透過一對一的函數來確保資料不會輕易被竄改,雜湊演算法很容易被驗證,但破解的難度極高,無法輕易回推出原本的數值,資料也就無法竄改,每個區塊得出的值也會被放進下一個區塊中,使得區塊間的資料皆被正確的保障。如果交易記錄包含錯誤,則必須新增交易來扭轉錯誤,且兩個交易都是可見的[16]。

3. 智能合約

以太坊引入了智能合約概念[14,20]，是一種自動執行的程式，用以儲存資料、執行合約內容以及進行驗證，允許用戶在區塊鏈上建立和運行各種應用程序，用「分散式」與「一致性」的演算法來保證他的去中心化，可以在沒有第三方介入的情況下執行合約，也能隨時追蹤合約的執行狀況，並透過自動執行來減少執行成本[4,5]。

2.1.3 區塊鏈運作原理

交易驗證過程中，區塊鏈利用數位簽章和哈希函數等加密技術[21]。數位簽章確保交易的真實性和完整性，只有擁有相應私鑰的用戶才能生成有效的簽章，而網路中的其他節點則使用公鑰來驗證簽章的有效性。哈希函數則將交易資料轉換為固定長度的哈希值，確保資料的唯一性和不可逆性，如圖 1 所示。

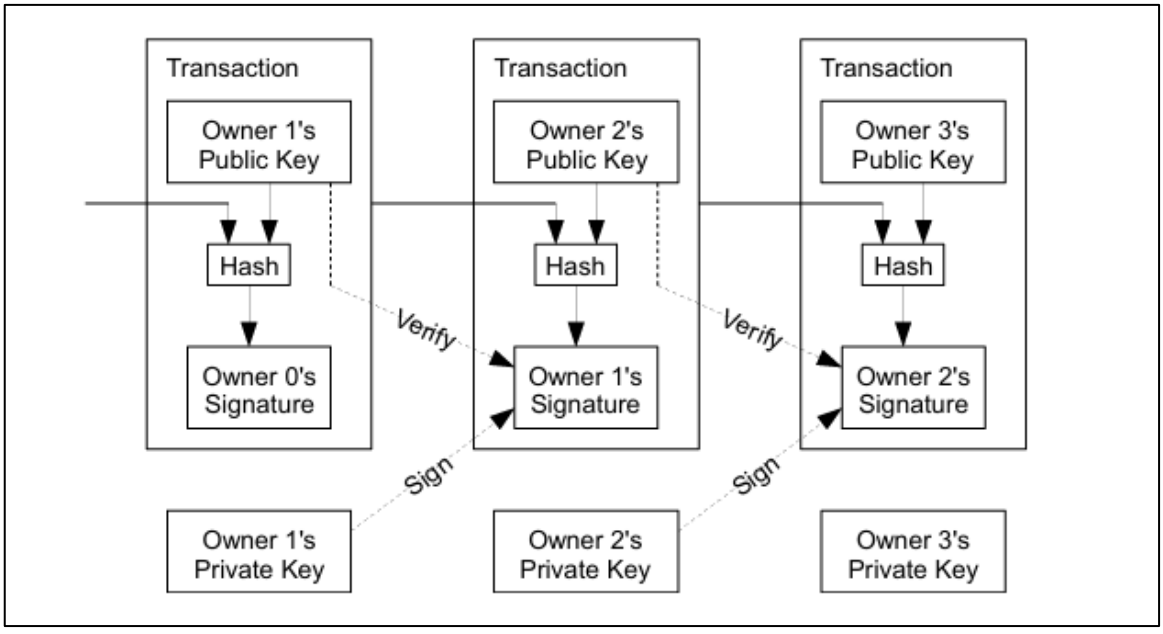

圖 1　電子貨幣的數位簽章[21]

每個區塊都包含前一個區塊的加密哈希值，這種鏈接方式確保區塊鏈的不可篡改性，任何對已存在區塊資料的更改都會導致後續區塊的哈

希值發生變化，從而被網路中的其他節點檢測到，保障交易的順序性和時間戳記的可靠性，防止惡意行為者插入或修改交易資料[18,19]，如圖 2 所示。

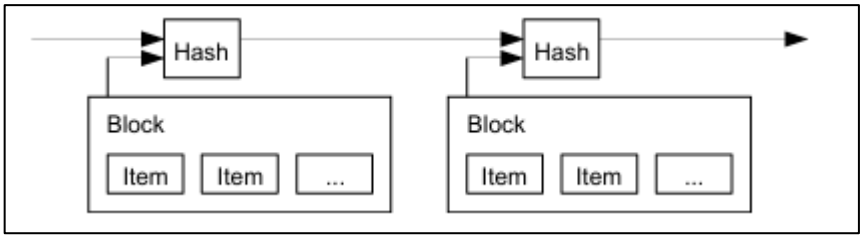

圖 2　區塊添加時間戳記進行哈希值加密[21]

而為了在區塊鏈架構下完成時間戳記，需要利用「工作量證明」機制建立時間序列。每個節點要不斷嘗試不同的隨機數（稱為 nonce），直到找到一個結果，當某個節點成功找到這樣的結果，就表示它完成了足夠的「工作量」，可以將其區塊加入區塊鏈中。其他節點只需進行一次哈希運算，即可驗證對錯。若驗證失敗，則必須重新開始整個計算過程，如圖 3 所示。

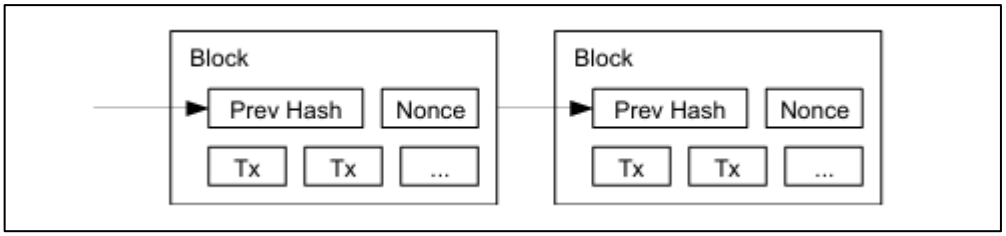

圖 3　工作量證明[21]

2.2 乙太坊

於 2014 年由 Vitalik Buterin 受比特幣啟發而提出的概念，乙太坊（Ethereum）是一個去中心化、開放開源碼的公共區塊鏈平台 [14,24]。提供去中心化乙太坊虛擬機（Ethereum Virtual Machine, EVM）來處理對等合約。並於 2015 年推出，具有圖靈完備的程式語法，這種語言的引入提供了極高的靈活性，開發者能夠使用專用編程語言在其區塊鏈上部屬智能合約[2,15]。

2.3 智能合約

智能合約（Smart Contract）是一種部屬於區塊鏈平台的智慧型協定，當中內含了程式碼函式，可根據預先定義之條件執行特定交易行為，亦能與其他合約進行互動、做決策、儲存資料及傳送乙太幣等功能 [20,22]。智慧型合約提供驗證及執行合約內所訂立的條件，當合約條件一旦被滿足，智能合約會自動執行對應的操作，所有參與者可以立即透過區塊鏈查驗執行結果，無須倚賴第三方仲介進行仲裁，進而降低信任成本及時間延遲，提升交易效率[26]。IBM 於其區塊鏈平台中指出：「智能合約可在交易參與方之間建立共享與可信的運作機制，顯著提高跨企業流程的效率與透明度」[17]。且透過區塊鏈的特性，合約的程式內容無法被竄改，可以保證執行過程無法舞弊，因此智能合約也經常應用於數位產權的保護以及交易[1]。

然而，智能合約也面臨一些挑戰，例如:合約安全漏洞、語意錯誤、執行不可逆等問題。2016 年著名的 The DAO 專案，即是因為智能合約

漏洞，大規模的資金被抽走造成重大損失，此事件也說明了智能合約在缺乏完善審核與監管機制下所面臨的風險與挑戰[15]。

2.4 多對多配對

區塊鏈技術以其去中心化和不可篡改的特性，在各個領域得到廣泛應用。然而，隨著應用場景的複雜化，傳統的一對一或一對多的交易模式已難以滿足需求，因此，多對多配對機制成為提升區塊鏈交易效率和靈活性的重要發展方向。

2.4.1 Gale-Shapley 演算法

由 David Gale 與 Lloyd Shapley 於 1962 年提出[13]，旨在解決穩定婚姻問題。該問題產生於人數相同的兩個群體相互配對，群體中每位成員都有對另一組成員偏好排序的配對情境中，找到一種配對方式，使得不存在任何一對男女配對中彼此有其他更喜歡的對象而不滿意當前配對結果的情形發生，即所謂的「穩定配對」[10]。此演算法運作原理如下，以穩定婚姻問題舉例說明:

通常假設其中一方作為主動提案方，另一方則為被提案方。目前有一組男人與一組女人要進行配對，採用男人主動邀請女人的形式。

設有兩組參與者：

男性 $M = \{ m_1, m_2, ..., m_n \}$

女性 $W = \{ w_1, w_2, ..., w_n \}$

每位 $m \in M$ 與 $w \in W$ 對另一方有完整的偏好排序。令：

$pref_m(w)$ 表示男性 m 對女性 w 的偏好（數值越小越偏好）

pref_w(m) 表示女性 w 對男性 m 的偏好

定義一組配對為集合：

$M = \{ (m_i, w_j) \mid m_i \in M, w_j \in W \}$

若存在配對 M 中的兩對 (m, w') 與 (m', w)，使得：

m 偏好 w 高於 w'： $pref_m(w) < pref_m(w')$

w 偏好 m 高於 m'： pref_w(m) < pref_w(m')

則稱 (m, w) 構成一對不穩定配對（Blocking Pair），配對 M 即為不穩定配對。若在所有配對中不存在任何不穩配定對，則該配對集合稱為穩定配對（Stable Matching）。

演算法整體流程描述如下(以男性作為主動提案方):

第一回合，每個人都會選擇自己的第一志願，但這時候會有兩種情況發生:

1. 女人未被其他男士邀請，因此接受該男士的邀請。
2. 女人已被其他男士邀請，便會將兩者做比較，並選擇比較心儀的那位。

第一回合結束後，有些人已成功配對，有些人依舊單身。

第二回合,單身的男士從未拒絕過他的女士中選擇第一志願進行邀請，這時，便會出現第一回合中的兩個情況其中之一，同樣的模式持續運作，直到所有人都配對成功[7,11]。

此演算法的穩定性來自於其「延遲接受」的特性，被提案方可以不用再無法更改的情況下立即訂定選擇，而是能在多方選項中持續保留最優者。這種機制有助於減少潛在的不穩定因素，使最終配對結果中不會存在任意一對組合想要私下脫離目前配對對象而結成新組合的情形。

此外，演算法的結果具有提案方最佳性，也就是說，在所有可能的穩定配對中，演算法所產出的結果是對提案方最有利的解。這一特性在制度設計中尤其關鍵，特別是在教育分發、醫療資源配置、以及區塊鏈智能合約等需強調公平與透明的應用場景。

2.4.2 Top Trading Cycle 演算法

由 Herbert Scarf 和 Lloyd Shapley 於 1974 年提出[25]。是一種不使用金錢即可交易不可分割物品的演算法，使用於房屋交換，器官移植配對此類交換型配對市場[23]。此演算法的核心概念在於透過每個參與者志願序所製成的表格進行循環交易，保證最終結果的配對穩定性，運作過程如下:

詢問每個人想交換的志願序,不須列出志願序低於自身的（維持現狀比交易好)，列出下列表格：

志願序	A	B	C	D	E
1	C	A	B	E	A
2	A	B	C	D	B
3					C
4					D
5					

表 1　TTC 演算法參與者偏好志願序

畫出每個人指向自己最高志願序的箭頭，並觀察能否形成一個環，若可以形成一個環，將環內的參與者去除，剩下的參與者持續前述步驟。

從以上圖表格得知各參與者第一志願，並畫出指向:

A → C

B → A

C → B

D → E

E → A

第一志願中，B → A, A → C, C → B, 形成循環，將 A,B,C 去除。

接下來的志願序中(對於每個人最好的志願序中)，剩下 D,E 進行配

對

D → E→ D，形成循環，將 D,E 去除。

最終結果顯示，每位參與者皆獲得其偏好列表中較前順位之資源[9]。

第三章 研究方法

本研究旨在比較兩種多對多配對演算法在智能合約中的執行效率與優劣。包括智能合約的撰寫、部署與測試，並透過 Remix IDE 進行模擬執行與實驗觀察。期望能結合理論與實作模擬，全面檢視研究假說的可行性與實際應用情況。

3.1 研究工具

本研究實作部分採用 Solidity 語言進行智慧合約開發。在開發初期，首先使用 Remix IDE 搭建合約雛型，並針對演算法邏輯進行驗證；隨後的效能評估階段，則導入 Hardhat 進行測試，藉由其自動化測試環境來簡化重複測試流程，以利分析合約在不同條件下的執行效率。

3.1.1 Solidity

Solidity 是一種合約式導向的程式語言，用來撰寫智能合約，專門用於 Ethereum（以太坊）和其他 EVM（Ethereum Virtual Machine）兼容的區塊鏈上開發去中心化應用（DApps）。受 JavaScript 的影響，其語法簡潔且適合初學者。其為以太坊虛擬機(EVM)之通用標準語言。因此採用 Solidity 撰寫智能合約，確保程式碼與 EVM 運作邏輯的相容性，以利驗證配對演算法，同時增加合約在實務應用的彈性。

3.1.2 Remix IDE

本研究選用 Remix IDE 作為主要的智能合約開發工具。Remix 為一款基於瀏覽器的整合開發環境（Integrated Development Environment,

IDE)，內建了 Solidity 編譯器、合約部署介面、交易模擬以及除錯工具，無須額外安裝本地開發套件即可使用，降低開發環境的建置成本，且直觀的操作介面有助於開發者快速上手。

3.1.3 Hardhat 開發框架

Hardhat 為一套專為以太坊軟體開發而設計的開發環境，提供部署合約、運行測試和調試程式碼等服務，且內建本地開發網路，開發者可無須連接外部節點和消耗真實代幣的情況下，模擬完整的區塊鏈交易行為與挖礦流程。

3.2 研究流程

本研究之整體流程可分為兩個階段：第一階段聚焦於配對演算法之運作原理與邏輯流程，第二階段則著重於智能合約的開發與實作效能比較分析。此架構設計旨在兼顧理論與實務層面，確保演算法不僅在數學上成立，也適用於區塊鏈環境，全面評估其在去中心化應用的可能。

第一部分：配對演算法邏輯理解與轉換

針對 Gale-Shapley（GS）演算法與 Top Trading Cycle（TTC）演算法進行文獻探討，詳細理解兩種演算法在多對多配對問題中的運作邏輯、決策流程與穩定性原則，為後續智能合約撰寫建立明確的邏輯架構，並透過分析兩種演算法的配對性質差異，制定評估兩合約效能參考指標。

第二部分：Solidity 智能合約實作與效能比較

進入合約實作階段，以 Solidity 語言撰寫對應 GS 與 TTC 演算法之智能合約。實作重點在於解決鏈上資源有限之特性，對資料結構與迴圈機制進行最佳化設計，以有效減少 Gas 消耗並確保合約可於實際環境執行。

合約開發初期，先透過 Remix IDE 驗證基礎邏輯，後續效能實驗階段則採用 Hardhat 開發環境進行模擬配對，透過自動化腳本執行部署及配對交易，產出結果從而全面評估不同演算法在鏈上執行之適配性與執行效率。

為評估兩種演算法於區塊鏈環境下的實用性與效能表現，設定以下三個關鍵指標作為比較依據：

1. Gas 消耗量（Gas Consumption）：反映智能合約執行成本，為衡量效率與資源使用的重要指標。
2. 執行時間（Execution Time）：即合約完成整體配對流程所需時間，有助於評估演算法於鏈上應用時的即時性。
3. 配對完整性與穩定性：演算法是否能在合約限制下完成所有配對，並檢視其配對結果是否符合理論上應有之穩定或最適性質。

第四章 多對多配對智能合約設計與實作測試

本章將說明本研究如何從設計智能合約、進行部署與測試，進而針對配對演算法進行效能評估的完整過程，對應第三章所建立之研究架構，實際驗證 Gale-Shapley 與 Top Trading Cycle 演算法於區塊鏈環境下之應用性與可行性。

4.1 Gale-Shapley 演算法智能合約設計

在本研究的前述章節中，已針對 Gale-Shapley 延遲接受演算法的理論基礎與運作流程進行說明。該演算法的核心在於「提案方依據偏好依序提出配對申請，而被提案方則保留目前好感較高的提案者，並可在後續輪次中進行更替」，整體過程會在有限步驟內完成一組穩定配對。

基於上述理論，本研究在 Solidity 智能合約中設計相應的結構與執行邏輯，合約設計的重點如下：

1. 參與者建模與資料結構

   配對狀態需具備儲存性與可查詢性，採用 mapping 資料結構記錄每位 proposer 與 receiver 的偏好清單、配對對象與提案進度。

   使用兩個 address[]陣列分別儲存 提案方(proposer)與接受方(receiver)的列表。偏好清單以 mapping(address => address[])的結構進行記錄，以便鏈上查詢與配對操作。配對狀態紀錄於雙向 engagements 映射中，並使用 engaged 映射追蹤每個地址的配對狀態：

```
mapping(address => address) public engagements;
mapping(address => bool) public engaged;

mapping(address => address[]) public proposerPreferences;
mapping(address => address[]) public receiverPreferences;

mapping(address => uint) public proposerNextProposalIndex;
```

圖 4　本研究中 GS 演算法智能合約程式內容截圖-1

2. 偏好清單上傳

利用 setProposerPreferences()與 setReceiverPreferences()函式，使用者（由合約擁有者代表）可將偏好清單上傳至鏈上。並同時透過 addProposers()與 addReceivers()建立參與者列表。模擬現實中去中心化平台上，使用者事先輸入偏好參與媒合的過程。

在 addProposers()與 addReceivers()函式中，分別使用 for 迴圈將輸入的參與者地址陣列逐一寫入合約內部的 proposers[]與 receivers[]陣列。這種設計可確保所有參與對象在鏈上均有可追蹤的身份紀錄，也方便後續配對邏輯中以迴圈依序調用各提案者進行操作。

```
constructor() {    1229716 gas 1204200 gas
    owner = msg.sender;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Only owner can call this.");
    _;
}

function addProposers(address[] memory _proposers) public onlyOwner {    infinite gas
    for (uint i = 0; i < _proposers.length; i++) {
        proposers.push(_proposers[i]);
    }
}

function addReceivers(address[] memory _receivers) public onlyOwner {    infinite gas
    for (uint i = 0; i < _receivers.length; i++) {
        receivers.push(_receivers[i]);
    }
}
```

圖 5　本研究中 GS 演算法智能合約程式內容截圖-2

3. 提案流程及偏好判斷邏輯

主配對流程由 matchProposers()函式負責控制，透過 while 迴圈與內部 for 迴圈反覆檢查 proposer 是否仍有尚未成功配對者，並依偏好順序提出配對申請。而為對應理論中「保留最佳選擇」的原則，因此設計 prefers()函式比較 receiver 偏好清單中兩位提案者的優先順序，以確保演算法穩定性特性不被破壞:若 receiver 尚未配對，則接受該 proposer；若 receiver 已配對，則執行 prefers()函式判斷其

偏好，若更偏好新 proposer，則進行替換配對。

```
while (freeProposerExists) {
    freeProposerExists = false;

    for (uint i = 0; i < proposers.length; i++) {
        address proposer = proposers[i];

        if (engaged[proposer]) {
            continue;
        }

        freeProposerExists = true;

        address[] memory prefs = proposerPreferences[proposer];
        uint proposalIndex = proposerNextProposalIndex[proposer];

        if (proposalIndex >= prefs.length) continue;

        address receiver = prefs[proposalIndex];
        proposerNextProposalIndex[proposer]++;
```

圖 6　本研究中 GS 演算法智能合約程式內容截圖-3

4. 配對完成與結果查詢

配對流程完成後，所有配對資料皆可透過 engagements[address]查詢，也可使用 getPartner()函式查詢某地址對應的配對對象，具備良好的可追蹤性與鏈上透明性。

```
function getPartner(address participant) public view returns (address) {
    return engagements[participant];
}
```

圖 7　本研究中 GS 演算法智能合約程式內容截圖-4

4.2 Top Trading Cycle 演算法智能合約設計

Top Trading Cycle(TTC)演算法的核心概念在於：「所有參與者基於偏好指向其最想要的資源，系統則根據指向關係找出封閉交換環，並一次性完成該環內所有成員的資源交換」整體過程將在有限輪次內逐步消除所有尚未配對的參與者，最終完成完整的資源媒合。基於上述邏輯，本研究將 Top Trading Cycle 演算法轉化為模組化設計架構，將參與者偏好與資源所有權邏輯清楚分離，以提升可讀性與合約擴展性。以下為本研究實作之 TTC 合約邏輯與資料結構設計重點說明。

1. 參與者紀錄與偏好結構設計

合約中透過 participants 陣列紀錄所有參與者地址，偏好清單則使用 mapping(address => address[]) preferences 記錄每位使用者對資源的喜好排序。

```
address[] public participants;
mapping(address => address[]) public preferences;
mapping(address => bool) public hasTraded;
```

圖 8　本研究中 TTC 演算法智能合約程式內容截圖-1

此外，使用 ownership 映射紀錄每位使用者最初持有的資源。

```
mapping(address => address) public ownership;
```

圖 9　本研究中 TTC 演算法智能合約程式內容截圖-2

2. 配對流程控制與封閉環邏輯

主邏輯 executeTTC()中，合約透過以下邏輯模擬 TTC 運作流程：

(1) 所有尚未交易者指向其偏好清單中最高順位之資源。

(2) 查找指向關係是否形成封閉循環（cycle）。

(3) 若存在循環，則其成員依據指向順序進行資源交換。

(4) 交換完成後，將已交易者移除，進入下一輪。

```
        address current = start;

        while (!inCycle(cycle, current, index)) {
            cycle[index++] = current;
            current = pointTo[current];
        }
```

圖 10　本研究中 TTC 演算法智能合約程式內容截圖-3

3. 最終配對紀錄與查詢功能

配對完成後，會將每位使用者最終獲得的資源記錄於 finalMatch，供查詢與驗證使用。

```
mapping(address => address) public finalMatch;
```

圖 11　本研究中 TTC 演算法智能合約程式內容截圖-4

```
function getMatchedItem(address user) public view returns (address) {
    return finalMatch[user];
}
```

圖 12　本研究中 TTC 演算法智能合約程式內容截圖-5

4.3 合約測試

本研究採用 Hardhat 作為 Solidity 智能合約的本地測試框架。測試腳本使用 JavaScript 撰寫，並透過 ethers.js 套件呼叫 Solidity 合約，模擬使用者操作流程。並結合 Mocha 測試框架與 Chai 驗證工具(由 Hardhat 預設整合) 檢查合約是否正確執行配對邏輯，並驗證每位參與者是否獲得正確的配對結果。

下列為實際執行介紹:

1. 建立測試專案資料夾

建立測試專案根目錄。(TTC 合約測試為範例)

```
C:\Users\USER>mkdir ttc-hardhat
```

圖 13　本研究合約測試執行終端機指令截圖-1

```
C:\Users\USER>cd ttc-hardhat
```

圖 14　本研究合約測試執行終端機指令截圖-2

2. 初始化 Node.js 專案環境

使用該指令建立 package.json，作為 Hardhat 所需依賴管理環境。

```
C:\Users\USER\ttc-hardhat>npm init -y
Wrote to C:\Users\USER\ttc-hardhat\package.json:

{
  "name": "ttc-hardhat",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

圖 15　本研究合約測試執行終端機指令截圖-3

3. 安裝 Hardhat 與相關工具

安裝 Hardhat 及測試所需的基本工具。

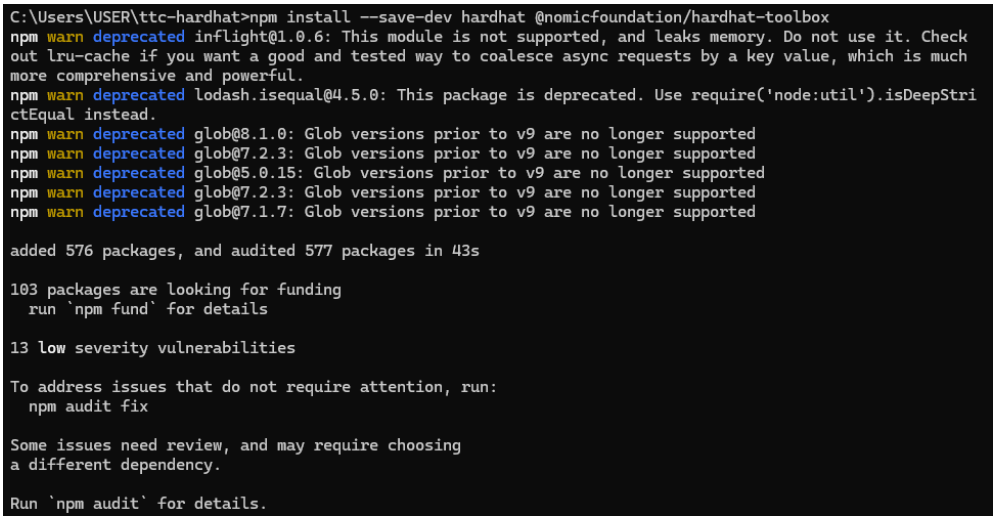

圖 16　本研究合約測試執行終端機指令截圖-4

4. 初始化 Hardhat 專案架構

建立 Hardhat 專案基礎結構，並新增合約以及測試腳本放置資料夾。

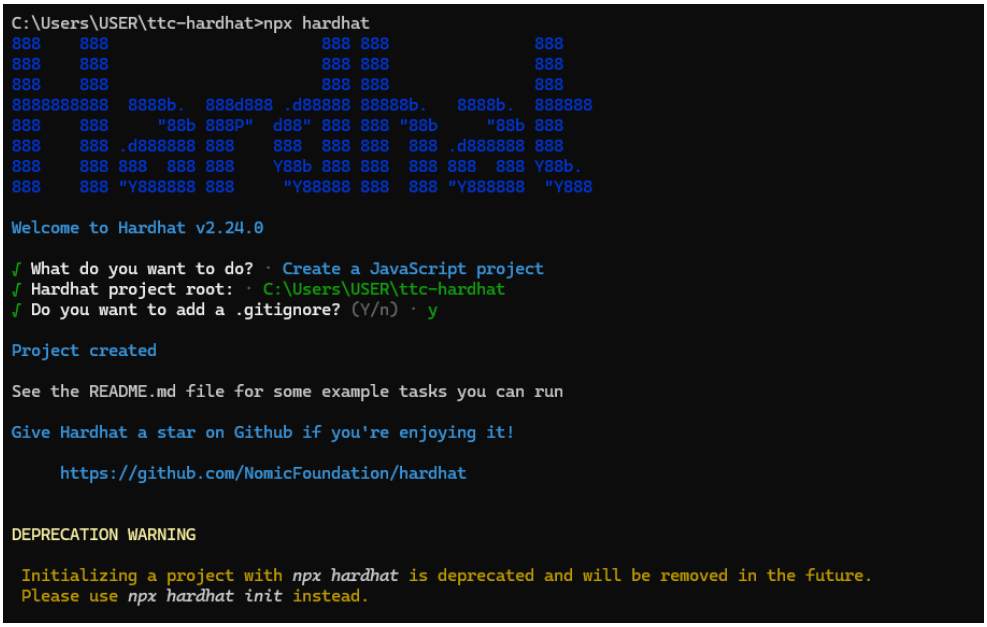

圖 17　本研究合約測試執行終端機指令截圖-5

將所撰寫好的合約以及腳本分別放入資料夾。

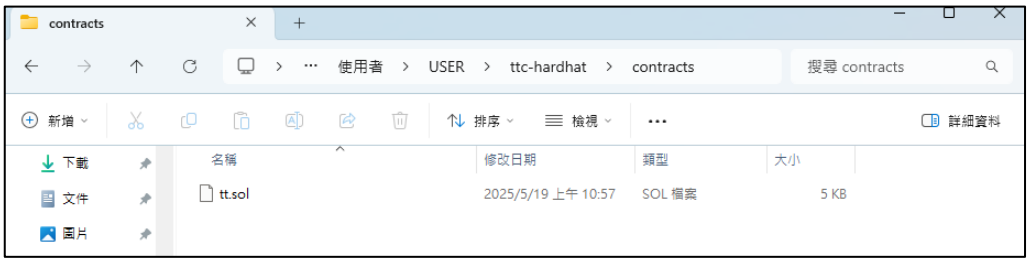

圖 18　本研究合約測試專案資料夾截圖-1

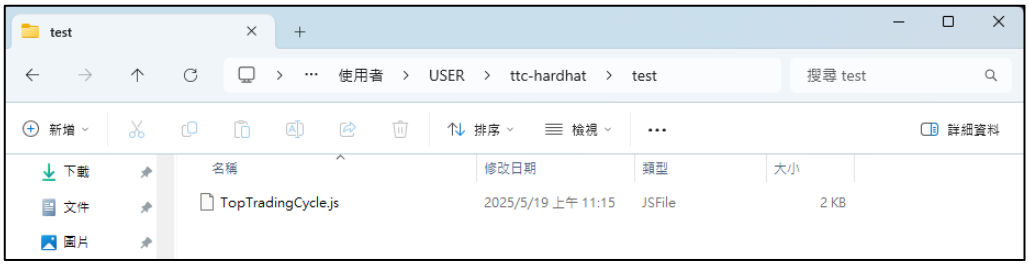

圖 19　本研究合約測試專案資料夾截圖-2

5. 執行測試

輸入指令啟動本地測試流程，終端機將會輸出測試執行情況、配對結果與 Gas 使用量等資訊，並標示通過與否。

```
C:\Users\USER\ttc-hardhat>npx hardhat test


  TopTradingCycle
A got: 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc
B got: 0x976EA74026E726554dB657fA54763abd0C3a0aa9
C got: 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65
    ✓ should assign new items correctly using TTC


  1 passing (451ms)
```

圖 20　本研究合約測試執行終端機指令截圖-6

第五章 結果和建議以及未來規劃

5.1 結論

本研究實作並比較兩種多對多配對演算法在智能合約的應用並觀察其運作過程與效率表現。

以下為兩種不同演算法進行模擬配對測試的結果:

Gale-Shapley 延遲接受演算法，輸出結果如下:

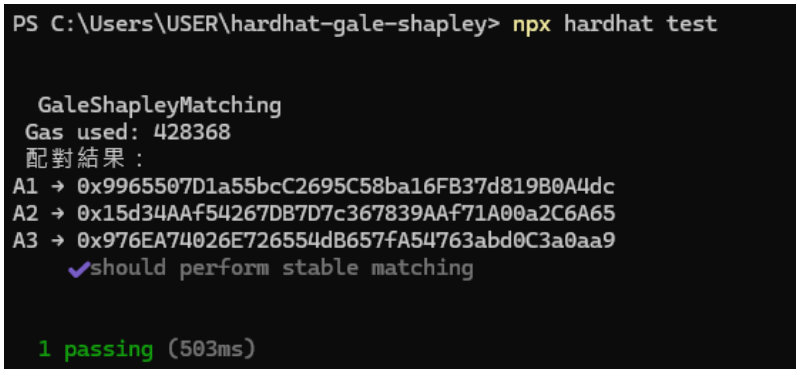

圖 21　本研究 GS 合約測試執行終端機指令結果截圖

- Gas used: 428,368。
- 執行時間：505ms。
- 所有提案方皆完成穩定配對，無人可提升偏好。

Top Trading Cycle 交換循環演算法

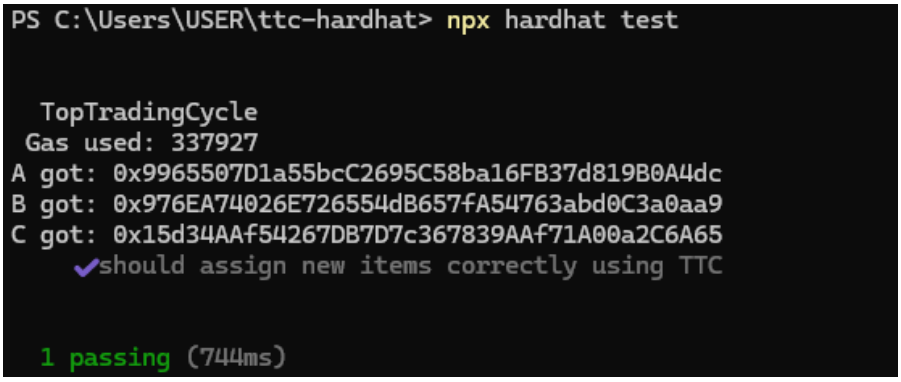

圖 22　本研究 TTC 合約測試執行終端機指令結果截圖

- Gas used:337,927。

- 執行時間：457ms。
- 所有參與者皆完成交換，已從封閉循環成功移除。

合約執行結果比較分析:

分析項目	GS 演算法	TTC 演算法	比較結論
Gas 消耗量	428,368	337,927	TTC 消耗較少 Gas，效能更高。
執行時間	505ms	457ms	TTC 輪次結束較快，耗時較短。
配對完整性	全員配對完成	全員配對完成	兩者皆可保證全員配對。
配對穩定性	滿足穩定配對條件	所有封閉交換循環皆完成，無遺漏	皆符合各自演算法目標。
合約邏輯複雜度	依提案方偏好控制	需判斷封閉循環與模擬交換結構	TTC 較為複雜。
合約實作難度	邏輯清楚、資料結構明確，難度普通	需模擬交換循環並追蹤邏輯，難度較高	TTC 實作難度高於 GS。

表 2　本研究整理之 GS 與 TTC 演算法合約執行結果之比較

實作結果顯示：

1. Gale-Shapley 演算法可保證其配對穩定性，且雙方皆有偏好表達，更能保障公平性，適用於求職媒合、志願填報等配對情境。
2. Top Trading Cycle 演算法,可在有限循環輪次內完成資源封閉交換,Gas 消耗相對較低，適用於資源輪轉、器官交換與數位資產互換等實際資源物件交換應用。

兩者皆可於區塊鏈環境中穩定實現媒合功能，而 TTC 演算法在測試中較 GS 演算法節省約 21%的 Gas 成本，顯示其在處理交換循環上的經濟優勢：基於實驗數據與演算法特性，本研究針對更大規模應用與實務開發提出建議，主張開發者應採取「交易邏輯優先，技術規模為輔」的決策原則。

5.2 建議與未來展望

5.2.1 建議

1. 區塊鏈具備不可篡改特性，合約一旦部署即無法修改：未來若要正式部署合約，應先透過 Hardhat 或 Remix IDE 進行模擬測試，預先排除邏輯漏洞同時確保合約運作效能。

2. 現階段多對多配對機制多屬學術理論模型，尚未有成熟的實際應用案例，可進一步探討將此演算法導入 NFT 組合交易或去中心化資產交換等具體場景，驗證其在真實商業環境中的可行性與經濟效益。

5.2.2 限制

1. 本研究僅以 n=3（三組配對者）為範例進行測試，未針對大規模參與者情境進行交易壓力測試。

2. 測試過程僅以錢包地址作為資源代號進行配對，合約邏輯側重於配對演算法的實現，未包含實體資產（如房屋、醫療資源）的代幣化和所有權移轉機制。

3. 本研究範疇僅限於智能合約層級的後端邏輯驗證，未開發前端互動介面，無法評估一般用戶在實際操作流程中的體驗。

參考文獻

1. 王振軒(2019)。混成式跨鏈第三方託管交易之研究 – 以智慧合約實作。國立政治大學資訊管理學系碩士論文。
2. 林億雄，馬瀰嘉(2021)。以太坊區塊鏈與去中心化金融數據分析之研究。《智慧科技與應用統計學報》 19 卷 1 期 ，p. 1-19。
3. 黃浩哲(2024)。區塊鏈簡介與技術探討。資料來源:https://master.get.com.tw/it/column/detail.aspx?no=913134
4. 黃立群，鄭宇，黃曉濤(2022)。區塊鏈智能合約。北京:電子工業出版社，第一版。
5. 顏裕(2017)。以 Solidity 語言實作之多對多配對雛型系統。國立中央大學資訊管理學系碩士論文。
6. 蘇柏菖(2019)。多對多配對配合智能合約之雛型架構：以高等教育分發問題為例。國立中央大學資訊管理學系碩士論文。
7. Anne033(2020)。稳定婚姻问题：Gale–Shapley 算法。資料來源:https://blog.csdn.net/Anne033/article/details/107560938?spm=1001.2101.3001.6650.4&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-4-107560938-blog-132856170.235%5Ev43%5Epc_blog_bottom_relevance_base8&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-4-107560938-blog-132856170.235%5Ev43%5Epc_blog_bottom_relevance_base8&utm_relevant_index=8
8. aurora920114(2023)。什麼是區塊鏈?。資料來源:https://ithelp.ithome.com.tw/m/articles/10319698
9. JasonLiu(2021)。Top Trading Cycle(最適循環交易法)。資料來源:https://hackmd.io/@zhu424/Top_Trading_Cycle
10. m0_57781768(2024)。盖尔-沙普利算法（Gale-Shapley Algorithm）在多对一匹配中的应用。資料來源:https://blog.csdn.net/m0_57781768/article/details/139415456

11. sinTaro kevin(2025)。穩定配對演算法學習草記。資料來源:https://hackmd.io/@GoldxTree/rJlQChTFY
12. Bernhard K Meister, Henry CW Price(2024). Gas fees on the Ethereum blockchain: From foundations to derivative valuations. Frontiers in Blockchain, No.7, 1462666.
13. Gale, D. and Shapley, L. (1962) College Admission and the Stability of Marriage. American Mathematical Monthly, Vol.69, pp.9-15.
14. Gavin Wood(2014). Ethereum: A Secure Decentralized Generalized Transaction Ledger. Retrieved from:https://ethereum.github.io/yellowpaper/paper.pdf
15. Global X Research Team(2022). Exploring the Disruptive Potential of Smart Contracts. Retrieved from :https://www.globalxetfs.com/exploring-the-disruptive-potential-of-smart-contracts/
16. IBM. Blockchain Introducton. Retrieved from:https://www.ibm.com/think/topics/blockchain#What+is+blockchain%3F
17. IBM. What are smart contracts on blockchain? Retrieved from :https://www.ibm.com/think/topics/smart-contracts
18. Marc Pilkington(2016). Blockchain Technology: Principles and Applications. . In F. X. Olleros & M. Zhegu (Eds.). Research handbook on digital transformations, pp.225–253. Edward Elgar Publishing.
19. Merkle Tree in Blockchain: What It Is and How It Works. 2024. Retrieved from :https://www.investopedia.com/terms/m/merkle-tree.asp
20. Nick Szabo(1996).Smart Contracts: Building Blocks for Digital Markets. Extropy Journal of Transhuman Thought, No.16.
21. Satoshi Nakamoto(2008).Bitcoin: A peer-to-peer electronic cash system. Retrieved from :https://bitcoin.org/bitcoin.pdf

22. Solidity. Retrieved from :https://docs.soliditylang.org/en/v0.8.29/#

23. Top trading cycle. Wikipedia. Retrieved from :https://en.wikipedia.org/wiki/Top_trading_cycle

24. Vitalik Buterin(2014). Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. Retrieved from :https://ethereum.org/en/whitepaper/

25. Lloyd Shapley,Herbert Scarf(1974). On cores and indivisibility. Vol.1, No.1, pp. 23-37.

26. Zubin Pratap(2022). Reentrancy Attacks and The DAO Hack. Retrieved from :https://blog.chain.link/reentrancy-attacks-and-the-dao-hack/