國立暨南國際大學資訊管理學系研究所

碩士論文

開發改良式螞蟻演算法求解連續型最佳化問題

Developing an enhanced ant colony search algorithm for continuous optimization problems

指導教授：游子宜 博士

研究生：陳以承

中華民國 102 年 7 月

國立暨南國際大學碩士論文考試審定書

資訊管理 學系（研究所）

研究生 陳以承 所提之論文

開發改良式螞蟻演算法求解連續型最佳化問題

Developing an enhanced ant colony search algorithm for continuous optimization problems

經本委員會審查，符合碩（博）士學位論文標準。

學位考試委員會

楊建成 委員兼召集人

洪嘉良 委員

游子宜 委員

中華民國 一〇二 年 六 月 二十一 日

致謝

很快的兩年的研究生崖即將告一段落，在這說長不長說短不短的時間裡，要感謝的人實在太多了，很感激這些人在這段時間的陪伴，讓我的研究所生活更多采多姿，謝謝每位親朋好友的支持與鼓勵。

首先要謝謝我的指導教授－游子宜老師，在這兩年的時間裡，教會了我許多東西，從一個完全不會寫程式的我，慢慢的教導與鼓勵，使我有了現在的成果，更感謝老師在許多時候的叮嚀，讓我們在求學之餘，更可學到許多人情世故，與生活的小細節。

更要感謝我的家人，感謝爸爸從小到大的栽培以及提供我良好的學習環境，讓我可以無憂無慮的念完研究所，感謝媽媽永遠總在第一時間的關心及打氣，讓我的心總是可以暖暖的，感謝姊姊提供了我許多，做學生的態度及方法，總是聽著我遇到困境時的抱怨，謝謝你們，讓我雖然身在遠方，但是總不會忘記，我有一個溫暖且幸福美好的家。

最後要感謝的是一路陪我相伴的同學，杰哥、世宇，感謝你們在實驗室的許多幫忙，以及一同遇到的許多問題，終於要苦盡甘來了。感謝逸哲、鴻鴻、春春、阿幹學長，冠伃學姊，在我還是研一的時候，教導了我許多課業方面的問題。感謝芳宜學妹，佳興學弟，在實驗室帶來的歡樂。感謝小馬、蕾蕾研究所兩年的陪伴，讓我可以好好的傾訴與放鬆，感謝阿玫在我遇到許多問題時，總是陪我著一起面對，感謝研究所的朋友，讓我這兩年的生活豐富且順利。

感謝每位關心我，幫助我的人，因為有你們，才會有現在的我，之後將帶著大家的祝福與期盼，繼續讓我未來的生活發光發熱，實在太感謝你們了。

論文名稱：開發改良式螞蟻演算法求解連續型最佳化問題

校院系：：國立暨南國際大學管理學院資訊管理學系　　頁數：69

畢業時間：一〇二年七月　　學位別：碩士

研究生： 陳以承　　指導教授：游子宜 博士

論文摘要

最佳化的應用在日常生活中比比皆是。所謂的最佳化是利用數學方法在函式限制式中找到最佳最好的數值。然而真實世界最佳化問題常常是高階且非線性的數學問題，傳統的數學理論跟方法無法求出確實的理論解。近年來啟發式演算法逐漸受到重視，因為這個演算法可以用來解決複雜的最佳化問題。在許多不同的啟發式演算法中，螞蟻演算法(ACO)在非連續型問題中是常被使用的熱門方法之一。本研究將演化式策略加入螞蟻演算法中，開發並提出運用於實數的連續型螞蟻演算法。經過不同的實測，本方法在穩定度跟精確度上都有優質的表現，證實本研究的方法可以運用到大維度的最佳化問題。

關鍵詞：螞蟻演算法(ACO)、非連續型問題、演化式策略

Title of Thesis：Developing an enhanced ant colony search algorithm for continuous optimization problems

Name of Institute：Department of Information Management, College of Management, National Chi Nan University Pages：69

Graduation Time：07/2013 Degree Conferred：Master

Student Name：Yi-Chen Chen Advisor Name：Dr. Yzu-Yi Yu

ABSTRACT

Optimization applications can be found easily in our daily life. Optimization is a mathematical practice that focuses on the finding the functional minima (or maxima) subject to functional constraints. However, most real life applications are high orders and nonlinear after converting to mathematical formula. Traditional and theoretical mathematical approaches fail to the solutions to these problems. On the other hand, heuristic algorithm has gained more attentions as it can be used to fit this requirement and solve the complex optimization problems. Among the heuristic algorithms, the Ant Colony Optimization (ACO) is one of the popular methods use for optimization problems with discrete domain. In this study, we embed the evolutionary mechanism and develop an enhanced method to make ACO suitable for continuous optimization problem. Various benchmark problems have been tested to verify the robustness and accuracy. The results show this proposed approach can be applied to large dimensional optimization problem with continuous domain.

Keywords: Ant Colony Optimization (ACO), discrete domain, Evolutionary Strategy

目　錄

表目錄

圖目錄

第一章　緒論

1.1　研究動機與背景

全球因為金融海嘯的關係，導致通貨膨脹、物價上漲，許多中小企業都面臨裁員甚至是經營不下去的問題。因此如何在有限的資源與成本下，取得最佳的利益就變得相當重要，這也是最佳化問題(optimization problems)的一種應用。

凡舉我們日常生活中就有許多關於最佳化的問題，在飲食方面，飲料瓶身的厚度該如何設計就是一例。太薄則可能無法忍受碰撞或是壓力的擠壓，然而太厚可能會浪費太多的資源而造成成本的上升。而在科技業方面，切割一塊大型面板，該如何在一定的面積下切割出各種不同形狀大小的面板，達到最少剩餘的材料，就能有效地降的成本，減少不必要的浪費，替公司省去許多開銷，這也是逆向的幫公司賺錢。在工廠的排程方面，有許多機器的製產流程的安排，如可降低未使用的機器，儘可能的讓每台機器同時運作，減少機器等待時的浪費，提高製程的效益等等都是最佳化的應用。而最佳化問題在經濟面就更為重要了，每個人都希望能賺取更多的錢，因此需要將資金做有效的投資，然而高報酬的投資策略就會有高風險的可能，而低風險的策略，投資報酬率相對也會較低，如何能在最低的風險之下，賺取最多的錢，就是最佳化投資一個典型的範例。

而傳統典型的最佳化問題，最經典的就是旅行銷售員問題(traveling salesman problem，簡稱 TSP)。此問題的定義為一個推銷員要走過 n 個城市推銷商品，但每個城市都需經過，但不可重複，最後再回到原點。在這些的條件下，找到最短成本的路徑。因為油價常常不穩定的再調漲，導致公車和計程車的票價也跟著漲，因此不管是在跑業務的銷售員，或是便利超商的物流車配送等，都希望能經過最佳化的路線，才能減少大量的油費與時間。還有許多問題都和最佳化息息相關，由此可知最佳化的問題在我們日常生活中是多麼的重要。然而這些最佳化問題往往可以用很複雜的數學式

子表示，且無法用傳統的數學方法(微積分)求得精準的解。所以這就是為什麼最佳化問題在現在成為這麼熱門的研究原因。

現今欲解決這些問題可以使用暴力法(brute force method)、以 TSP 問題為例，可以把所有路徑一一找出來，在找尋最短路徑，但此方法有一個很嚴重的缺陷，當城市數量不多時，可以輕而易舉地找出最佳路徑，但是當城市數量變多，條件限制變複雜時，此方法會變得相當龐大，計算量相對提高許多，造成很多資源的浪費，及時間的消耗，因此絕大部分的研究希望能避免使用此方法。另一種方法則是使用啟發式演算法(mata heuristic algorithm)，像是基因演算法(Genetic Algorithm 簡稱：GA)、螞蟻演算法(Ant Colony Optimization 簡稱：ACO)，這類的演算法，都是藉著之前的資料、或經經驗法則提供之後的決策選擇，再不斷的更新選擇方式，經過多次的選擇與比較之後找到最佳的解決方式。因此挑選一個良好的演算法相當的重要，好的演算法能有效地解決問題，在相同的資源下能減少更多的時間，能求得更精確地值。

1.2 研究目的

本研究將使用螞蟻演算法，透過模仿自然生物的習性，利用螞蟻群的覓食行為時，在經過的路徑留下費洛蒙，而較短的路徑將會留下較多的費洛蒙，由此特性找到最佳路徑。而螞蟻演算法已經被使用多年來解決各領域的問題，因此具有大量的理論基礎，然而傳統的螞蟻演算法，還是有部分的缺點，像是在問題較複雜，可選擇的路線較多時，可能會因為有太多螞蟻走在次佳的路線留下過多的費洛蒙，因此提早陷入區域最佳解的麻煩，相對的，也可能會因為螞蟻數量不夠，使得最佳的路線被拜訪的機會相對降低，因此導致蒸發速率過快，而無法有效的留下費洛蒙提供之後的螞蟻找尋最佳路徑。此外這個演算法還有一個問題，當城市數量變大，相對的螞蟻數量也會隨著變多時，每隻螞蟻需要走的路徑相對也會變長，再加上每隻螞蟻都須走完，由此可知螞蟻演算法，會因為螞蟻的數量及程式的數目過多，而導致需要較長的搜尋時間，使得整體的執行時間迅速的增加，大幅降低了程式的效能。

但傳統的螞蟻演算法，主要解決非連續、離散型、非實數的問題，但在真實世界上最佳化的問題，通常是高維度、非線性、限制式多且複雜的問題。雖然現在有許多的套裝軟體，可以提供運算求解，但是套裝軟體仍有許多缺陷無法突破，像是價格方面，因為有版權的關係，使得在始用軟體時，需額外付出一筆費用，且套裝軟體在遇到高維度、資料量龐大時，求解的精準度及穩定性會變地很差，獨立執行10次，可能每次的答案都不相同，而最麻煩的地方是不能修改程式碼，這樣在自變數的制訂及參數的設定會有很多不方便的地方。而針對這些問題，本研究將提出改良後的螞蟻演算法，使用在高為度且連續的函數，希望可以提出一個具有穩健性(Robust)的演算法，提供往後學術研究及產業界的使用。

1.3 全文結構

本研究倚賴相關文獻中的理論為基礎，改良傳統的螞蟻演算法，嘗試減少演算法的計算時間，並且求得更精準的解，提供本研究的演算法能解決不同維度的問題上。而本研究綱要概述如下：

第一章、 緒論

簡略介紹本研究的背景與目的，提出問題的定義，並簡單的說明使用的方法和使用的工具以及預期達到的效果，及對社會的貢獻；在本章最後將簡述全文架構。

第二章、 文獻探討

介紹螞蟻演算法的背景、並以旅行銷售員的問題為例，做簡單的說明。介紹 Dorigo 在 2008 年提出－連續型螞蟻演算法，並使用簡單的範例說明。

第三章、改良的連續型螞蟻演算

對於 Dorigo 在 2008 年提出提出的連續型螞蟻演算法作改良，主要針對固定的費洛蒙值，以及計算標準差的公式。

第四章、 實例分析

將使用不同的 benchmark 函式驗證，本研究所使用的演算法對於不同問題的求解能力及精確度。

第五章、 結論與建議

本章將所有的實驗結果作總結，並且討論建議未來的研究的方向。提出加入 GPU 的方法，及安裝步驟。

第二章　文獻探討

2.1　蟻群演算法(Ant Colony Optimization)

2.1.1　文獻回顧

在解決非線性高維度的最佳化問題上，過去就有許多的演算法被學者提出與研究，如早期的演化式計算(Evolutionary computation)。而其中有幾個代表的演算法，如：基因演算法(GA)、演化策略(ES)、演化規劃(EP)等，主要目標都在避免過早收斂至區域最佳解(local optima)以及能迅速的找到全域最佳解(global optima)。有一些演算法是仿造生物特性的行為去設計，例如：蟻群演算法(ACO)、飛鳥演算法(PSO)、蜂群演算法(BCO)等，而這些演算法被稱為：群體智慧演算法(Swarm Intelligence Algorithm)，主要是用來解決非決定性的問題，如：旅行銷售員(TSP)[6]、著色問題(GCP)，解決了許多無法經過數值演化的問題。

其中螞蟻演算法(Ant Algorithm，簡稱 AA)，最早是 M. Dorigo 等人在 1991 年所提出[4]，主要是模擬群體螞蟻的覓食行為，透過每隻螞蟻在尋找食物與走回巢穴的狀況，找出最短路徑，而設計出來的一種最佳化演算法。而 Dorigo 等人也在 1996 年的在 IEEE 發表了一篇期刊[5]，文章內容主要說明 AA 的步驟流程與各參數的設定，並使用 TSPLIB[23]的例子與 GA、SA、ES 等啟發式演算法做比較，結果都顯示螞蟻演算都優於其他演算法。而 Yao 、Alam 等人在 2012 年提出了 Diversity Guided Evolutionary Programming(DGEP)[15]，裡面使用了一個新型的突變策略，稱為 Diversity Guided Mutation(DGM)，而 DGM 同時擁有兩種搜尋方式：1.探索(exploration) 2.開採(exploitation)，讓解能夠跳出區域最佳解及找到全域最佳解。

而在螞蟻演算法中，最早被提出的模型為螞蟻系統(Ant System 簡稱：AS)，而 Dorigo 在 1997 年提出了螞蟻群系統(ant colony system 簡稱：ACS)，主要針對 AS 提

出三項改良：(1) 新增了狀態轉移的規則，(2) 改變費洛蒙路徑的更新方法，(3) 增加區域費洛蒙更新的規則。而 Stutzle 也在 1997 年提出了最大-最小螞蟻系統(max-min AS, 簡稱 MMAS)[8]，主要也是針對 AS 提出四項改良：(1) 只允許最佳螞蟻留下費洛蒙，(2) 設定路徑上的費洛蒙上下界，(3) 初始費洛蒙濃度均設定為上限值 (4) 加入路徑費洛蒙的平滑機制，而以上的方法都統稱為蟻群最佳化演算法(Ant colony optimization 簡稱：ACO)[7]。

以上兩位學者，都針對費洛蒙的公更新，提出了改良，對於往後使用螞蟻演算法的學者有了很大的貢獻，因為針對 AS 所提出的改良，可以避免螞蟻因為走過相同的路線，造成累積過高的費洛蒙，導致陷入區域最佳解的問題。

而螞蟻演算法也被使用在解決許多組合最佳化的問題，如：調度問題、車輛路徑問題、分配問題、設置問題、影像處理、資料探勘等，有此可知螞蟻演算法求解於整數形、非連續的問題是一個良好且被很多人使用與研究的演算法。

2.1.2 螞蟻演算法介紹

螞蟻演算法(Ant Colony Optimization)，簡稱 ACO，主要是模擬群體螞蟻的覓食行為，透過每隻螞蟻在尋找食物與走回巢穴的狀況，找出最短路徑。而每隻螞蟻在走過的路上會留下費洛蒙(pheromone)，而之後的螞蟻會根據之前螞蟻所留下來的費洛蒙，選擇要走的路線，而較多螞蟻走過的路，費洛蒙的量會越來越濃，而較少的走的路，會因為費洛蒙的蒸發而越來越淡。因此，路線較短的路，被走過的機率相對較大，相對的費洛蒙量也會較濃，進而找到最佳路徑。

而在 ACO 裡的重要的機制與參數名詞如下：

- 費洛蒙(Pheromone)

  螞蟻溝通的介質，而每隻螞蟻在行走時都會留下且可感知其他螞蟻所留下的費洛蒙，而當路徑因為螞蟻走過或是蒸發而有所更新的路徑，稱為費洛蒙路徑(Pheromone Trail)。

而費洛蒙的更新可分為兩類：

- 全域費洛蒙更新(global update)

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\Delta(i,j) \in L \tag{1}$$

其中，1-ρ 為費洛蒙的殘留因子，0< ρ <1，$\Delta\tau_{ij}^{k}$ 是第 k 隻螞蟻，留下來的費洛蒙，被定義為：

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{C_k} & \text{if e}\,(i,j)^{k} \text{ is gobal - best solution} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

其中，參數 Q 為一個常數，在早期的 AS 論文中設定為 1。C_k 為最佳螞蟻所走過的路徑總長，因此只有當代最佳解的路線才會更新螞蟻走過的費洛蒙，全域費洛蒙的主要目的是找出最短的路徑長。

- 區域費洛蒙更新(local update)

$$\tau_{ij} = (1-\omega)\cdot\tau_{ij} + \omega\cdot\tau_0 \tag{3}$$

其中，0<ω<1，τ_0 為極小但不為零的費洛蒙初始值，當一隻螞蟻經過一斷路徑的時候，減少一些費洛蒙，目的是避免在同一代裡的螞蟻走過相同的路。

- 蒸發機制(Evaporation Mechanism)

如同之前所敘，螞蟻的溝通方式以及找尋路線，是依造先前螞蟻所留下的的費洛蒙濃度而所選擇，而如果只有不斷累積的方式，可能會造成所有螞蟻都走向相同的路徑，而陷入停滯(Stagnation)的問題，無法找尋其他可能更好的路徑。因而導致演算法提早陷入區域最佳解，因此才有蒸發的機制，如之前提到的參數 ρ，即為費洛蒙的衰退參數，其值越大，代表衰退的幅度越大則可能需花費較多的時間找尋最佳解。若值過小，代表容易累積費洛蒙濃度，則容易陷入區域最佳。值固定，會因為問題的搜尋空間、路徑長度不相同，因此無法有效地找到最佳解。

- 狀態轉換機率(State Transition Probability)

當初始的每隻螞蟻在路線建構時，會依照機率規則選擇下一個可行的點，稱為狀態轉換，而選擇的機率規則如下：

$$P_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\tau_{ij})^{\beta}}{\sum_{l \in N_i^k}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}} \qquad if \quad j \in N_i^k \tag{4}$$

其中，$d_{i,j}$ 為點 i 與 j 兩點間的距離，啟發式訊息又稱為能見度(visibility)，代表螞蟻選擇下個點的偏好，通常以距離的倒數表示，即 $\eta_{i,j} = 1/d_{i,j}$ ，其值越高則被選到的機率越大。而 α 和 β 兩個參數是影響費洛蒙跟啟發式訊息的重要參數，N_i^k 是螞蟻 k 在城市 i 時鄰近還可以拜訪的點，之外的其他點都為 0，這是為了預防走過的點又在被走過一次，$\tau_{i,j}$ 為點 i 與點 j 兩點間殘留下來的費洛蒙量。

而參數 α 和 β 的設定相當重要，如果參數設定不當，容易造成停滯不前的情況，若 α=0 時，則為貪婪法(greedy)，螞蟻則會選擇路線較短的前進；若 β=0 時，則為隨機搜尋法(random search)，螞蟻將不考慮路線的長短，只依循費洛蒙的濃度前進，但初始路徑上的費洛蒙量均一樣，並沒有經過累積佳而有所不同，固為隨機搜尋。良好的選擇機制，可以加快收斂速度，避免提早陷入區域最佳解。

2.1.3 演算法流程

- 初始化及參數設定

設定每條路徑的費洛蒙起始值、費洛蒙的衰退參數、螞蟻數量、迭代次數、α 和 β 參數值。

- 路線建構

將螞蟻隨機放在任一節點上，並依照之前的轉換機率往下一個節點前進，要注意的是，螞蟻並非選擇機率較高的路徑前進，而是會隨機產生一個值，以那個機率來選擇下個節點，直到每隻螞蟻都完成他的旅程。

- 更新費洛蒙

當螞蟻完成一趟完整個旅程，稱為一次迭代(Iteration)，在 ACS 中，只會對當代最佳解的螞蟻所經過的路徑進行費洛蒙的更新：

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\Delta\tau_{ij}^{best} \quad \forall(i,j)\in T^{global-best} \tag{5}$$

- 找尋最佳解與最佳路徑

先將第一代每隻螞蟻的最佳解與最佳路徑，替換為全域最佳解，往後的每代在與全與最佳解作比較，如果有更好的解，變更新最佳解，即最佳路徑。

- 終止條件

當達到起始設定的最大迭代次數或是已達到全域最佳解時，即可停止。

傳統螞蟻演算法解離散型問題，流程圖如圖 1 所示：

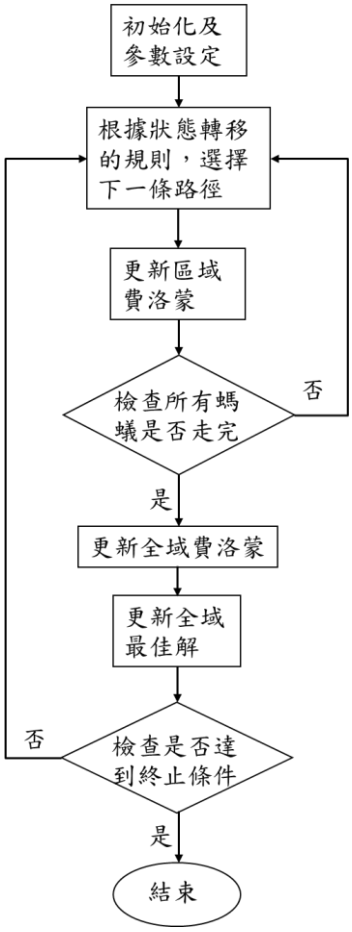


圖 1 離散型螞蟻演算法流程圖

2.1.4 離散型螞蟻演算法簡易範例

本節將使用傳統的螞蟻演算法，解決旅行銷售員的問題為例，做簡單的操作，觀察每條路徑在螞蟻經過時，費洛蒙的變化。而典型的 TSP 問題，要求所有點都必須經過且在不能重複的情況下，找出最短路徑為何。

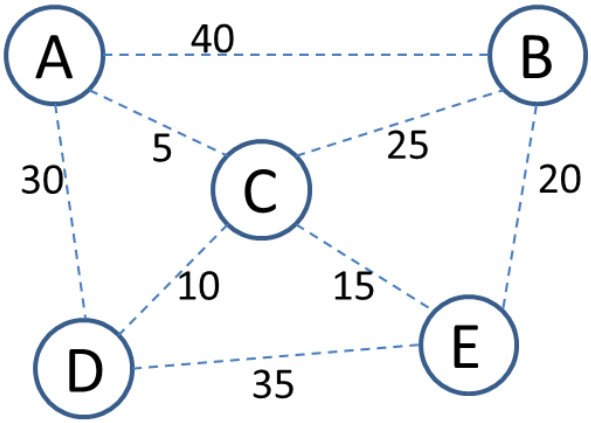


圖 2 城市路徑圖

表 1 各路徑距離與費洛蒙初始值

路徑	距離	費洛蒙
AB	40	0.001
AC	5	0.001
AD	30	0.001
BC	25	0.001
BE	20	0.001
CD	10	0.001
CE	15	0.001
DE	35	0.001

假設第一隻螞蟻 a_1 起始點為 A，則 $R_1[A]$，依照之前的狀態轉移求出可行走路徑之機率，並隨機產生一亂數值，假設為 0.01 故選擇路徑 AB，走完後並更新區域費洛蒙值，及陣列 $R_1[A,B]=40$，而無法行走的路徑，或走過之路徑，機率為 0，如表 2 的

BC、BE、CD、CE、DE。而第一代路徑上的費洛蒙均一樣，故每次更新區域費洛蒙的值，將不會改變。如表 2 所示。

表 2 螞蟻 a_1 在第一代的第一次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0.01497544	0.001
AC	0.95842818	0.001
AD	0.02659638	0.001
BC	0	0.001
BE	0	0.001
CD	0	0.001
CE	0	0.001
DE	0	0.001

接著假設第二隻螞蟻 a_2 起始點為 C，則 $R_2[C]$，依照之前的狀態轉移求出可行走路徑之機率，並隨機產生一亂數值，假設為 0.83 故選擇路徑 CD，走完後並更新區域費洛蒙值，及陣列 $R_2[C,D]=10$，而無法行走的路徑，或是走過之路徑，機率為 0。結果如表 3 所示。

接著假設第三隻螞蟻 a_3 起始點為 D，則 $R_3[D]$，依照之前的狀態轉移求出可行走路徑之機率，並隨機產生一亂數值，假設為 0.92 故選擇路徑 DE，走完後並更新區域費洛蒙值，及陣列 $R_3[D,E]=35$，而無法行走的路徑，或是走過之路徑，機率為 0。結果如表 4 所示。

接著再回到螞蟻 a_1，繼續依照狀態轉移的機率，往下個城市前進，而需注意的，點 A 已經走過，因為不可重複經過，故路線 AB 之機率為 0，而無法行走的路徑亦為 0。隨機產生一亂數值，假設為 0.28 故選擇路徑 BC，走完後並更新區域費洛蒙值，及陣列 $R_1[A,B,C]=65$，結果如表 5 所示。

表 3 螞蟻 a_2 在第一代的第一次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.001
AC	0.71377587	0.001
AD	0	0.001
BC	0.02855103	0.001
BE	0	0.001
CD	0.17844396	0.001
CE	0.07851543	0.001
DE	0	0.001

表 4 螞蟻 a_3 在第一代的第一次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.001
AC	0	0.001
AD	0	0.001
BC	0	0.001
BE	0.32233109	0.001
CD	0	0.001
CE	0.57246003	0.001
DE	0.10520887	0.001

而螞蟻 a_2 也跟著之前的方式選擇下一個節點，隨機產生一亂數值，假設為 0.47 故選擇路徑 AD，走完後並更新區域費洛蒙值，及陣列 $R_2[C,D,A]=40$，結果如表 6 所示。

表 5 螞蟻 a_1 在第一代的第二次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.001
AC	0	0.001
AD	0	0.001
BC	0.39024390	0.001
BE	0.60975609	0.001
CD	0	0.001
CE	0	0.001
DE	0	0.001

表 6 螞蟻 a_2 在第一代的第二次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.001
AC	0	0.001
AD	0.57633980	0.001
BC	0	0.001
BE	0	0.001
CD	0	0.001
CE	0	0.001
DE	0.42367601	0.001

而螞蟻 a_3 也跟著之前的方式選擇下一個節點，隨機產生一亂數值，假設為 0.16 故選擇路徑 BE，走完後並更新區域費洛蒙值，及陣列 $R_3[D,E,B]=55$，結果如表 7 所示。

表 7 螞蟻 a_3 在第一代的第二次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.001
AC	0	0.001
AD	0	0.001
BC	0	0.001
BE	0.36231884	0.001
CD	0	0.001
CE	0.64347826	0.001
DE	0	0.001

接著又回到螞蟻 a_1 的第三次狀態轉移及區域費洛蒙更新。隨機產生一亂數值，假設為 0.87 故選擇路徑 CE，走完後並更新區域費洛蒙值，及陣列 R_1[A,B,C,E]=80，結果如表 8 所示。

表 8 螞蟻 a_1 在第一代的第三次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.001
AC	0	0.001
AD	0	0.001
BC	0	0.001
BE	0	0.001
CD	0.69252077	0.001
CE	0.30747922	0.001
DE	0	0.001

接著為螞蟻 a_2 的第三次狀態轉移及區域費洛蒙更新。只能選擇路徑 AB，故機率值為 1，走完後並更新區域費洛蒙值，及陣列 R_2[C,D,A,B]=80，結果如表 9 所示。

表 9 螞蟻 a_2 在第一代的第三次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	1	0.001
AC	0	0.001
AD	0	0.001
BC	0	0.001
BE	0	0.001
CD	0	0.001
CE	0	0.001
DE	0	0.001

接著為螞蟻 a_3 的第三次狀態轉移及區域費洛蒙更新。隨機產生一亂數值，假設為 0.21 故選擇路徑 AB，走完後並更新區域費洛蒙值，及陣列 R_3[D,E,B,A]=95，結果如表 10 所示。

表 10 螞蟻 a_3 在第一代的第三次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0.28089887	0.001
AC	0	0.001
AD	0	0.001
BC	0.71910112	0.001
BE	0	0.001
CD	0	0.001
CE	0	0.001
DE	0	0.001

接下來螞蟻 a_1 在第四次的狀態轉移中只剩最後一個城市 D，故路徑 DE 的機率為 1，之後則從城市 D 走回原點 A，故路徑 AD 的機率為 1，之後更新區域費洛蒙及陣列 R_1[A,B,C,E,D,A]=145。相同的螞蟻 a_2 在第四次的狀態轉移中只剩最後一個城市 E，故路徑 BE 的機率為 1，之後則從城市 E 走回原點 C，故路徑 CE 的機率為 1，之後更新區域費洛蒙及陣列 R_2[C,D,A,B,E,C]=115。最後螞蟻 a_3 在第四次的狀態轉移中只剩最後一個城市 C，故路徑 AC 的機率為 1，之後則從城市 C 走回原點 D，故路徑 CD 的機率為 1，之後更新區域費洛蒙及陣列 R_3[D,E,B,A,C,D]=110。

當每隻螞蟻都從起點走回終點時，為一次迭代結束，此時則根據當代最佳螞蟻所經過的路徑(即當代最佳路徑)更新全域費洛蒙，更新結果如表 11。

表 11 第一次迭代結束之全域費洛蒙更新

路徑	費洛蒙
AB	0.00747272
AC	0.0002
AD	0.00747272
BC	0.00747272
BE	0.0002
CD	0.0002
CE	0.00747272
DE	0.00747272

第二次迭代開始，依造之前的放式開始建構路線，跟第一代不同的地方為，第二代的區域費洛蒙更新時，值會有所改變。假設第二代的第一隻螞蟻 a_1 起始點為 B，則 R_1[B]，接著求出可行走路徑之機率，並隨機產生一亂數值，假設為 0.89 故選擇路徑 BE，走完後並更新區域費洛蒙值，及陣列 R_1[B,E]=20，結果如表 12 所示。

表 12 螞蟻 a_1 在第二代的第一次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0.21650440	0.00747272
AC	0	0.0002
AD	0	0.00747272
BC	0.55169216	0.00747272
BE	0.23180343	0.00044
CD	0	0.0002
CE	0	0.00747272
DE	0	0.00747272

接著假設第二代的第二隻螞蟻 a_2 起始點為 D，則 R_2[D]，接著求出可行走路徑之機率，並隨機產生一亂數值，假設為 0.58 故選擇路徑 CD，走完後並更新區域費洛蒙值，及陣列 R_2[C,D]=10。結果如表 13 所示。

表 13 螞蟻 a_2 在第二代的第一次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00747272
AC	0	0.0002
AD	0.50609756	0.00747272
BC	0	0.00747272
BE	0	0.00044
CD	0.12195121	0.00044
CE	0	0.00747272
DE	0.37195121	0.00747272

接著假設第二代的第三隻螞蟻 a_3 起始點為 C，則 R_3[C]，接著求出可行走路徑之機率，並隨機產生一亂數值，假設為 0.04 故選擇路徑 AC，走完後並更新區域費洛蒙值，及陣列 R_3[A,C]=5。結果如表 14 所示。

表 14 螞蟻 a_3 在第二代的第一次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00747272
AC	0.13913043	0.00044
AD	0	0.00747272
BC	0.20695652	0.00747272
BE	0	0.00044
CD	0.07652173	0.00044
CE	0.57739130	0.00747272
DE	0	0.00747272

接著再回到螞蟻 a_1，求出可行走路徑之機率。隨機產生一亂數值，假設為 0.54 故選擇路徑 CE，走完後並更新區域費洛蒙值，及陣列 R_1[B,E,C]=35，結果如表 15 所示。

而螞蟻 a_2 也跟著之前的方式選擇下一個節點，隨機產生一亂數值，假設為 0.27 故選擇路徑 AC，走完後並更新區域費洛蒙值，及陣列 R_2[D,C,A]=15，結果如表 16 所示。

而螞蟻 a_3 也跟著之前的方式選擇下一個節點，隨機產生一亂數值，假設為 0.41 故選擇路徑 AD，走完後並更新區域費洛蒙值，及陣列 R_3[C,A,D]=35，結果如表 17。

接著又回到螞蟻 a_1 的第三次狀態轉移及區域費洛蒙更新。隨機產生一亂數值，假設為 0.92 故選擇路徑 CD，走完後並更新區域費洛蒙值，及陣列 R_1[B,E,C,D]=45，結果如表 18 所示。

接著為螞蟻 a2 的第三次狀態轉移及區域費洛蒙更新。只能選擇路徑 AB，故機率值為 1，走完後並更新區域費洛蒙值，及陣列 R2[D,C,A,B]=55，結果如表 19 所示。

接著為螞蟻 a3 的第三次狀態轉移及區域費洛蒙更新。只能選擇路徑 DE，故機率值為 1，走完後並更新區域費洛蒙值，及陣列 R3[C,A,D,E]=70，結果如表 20

表 15 螞蟻 a_1 在第二代的第二次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00747272
AC	0	0.00044
AD	0	0.00747272
BC	0	0.00747272
BE	0	0.00044
CD	0	0.00044
CE	0.84478371	0.00553090
DE	0.15521628	0.00747272

表 16 螞蟻 a_2 在第二代的第二次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00747272
AC	0.32592592	0.000608
AD	0	0.00747272
BC	0.22037037	0.00747272
BE	0	0.00044
CD	0	0.00044
CE	0.45370370	0.00553090
DE	0	0.00747272

表 17 螞蟻 a_3 在第二代的第二次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0.36006168	0.00747272
AC	0	0.000608
AD	0.63973831	0.00553090
BC	0	0.00747272
BE	0	0.00044
CD	0	0.00044
CE	0	0.00553090
DE	0	0.00747272

表 18 螞蟻 a_1 在第二代的第三次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00747272
AC	0.84668989	0.000608
AD	0	0.00553090
BC	0	0.00747272
BE	0	0.00044
CD	0.15331010	0.000608
CE	0	0.00553090
DE	0	0.00747272

接下來螞蟻 a_1 在第四次的狀態轉移中只剩最後一個城市 A，故路徑 AD 的機率為 1，則更新區域費洛蒙及陣列 R_1[B,E,C,D,A]=75，結果如表 21 所示。

接下來螞蟻 a_2 在第四次的狀態轉移中只剩最後一個城市 E，故路徑 BE 的機率為 1，則更新區域費洛蒙及陣列 R_2[D,C,A,B,E]=75，結果如表 22 所示。

表 19 螞蟻 a_2 在第二代的第三次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	1	0.00553090
AC	0	0.000608
AD	0	0.00553090
BC	0	0.00747272
BE	0	0.00044
CD	0	0.000608
CE	0	0.00553090
DE	0	0.00747272

表 20 螞蟻 a_3 在第二代的第三次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00553090
AC	0	0.000608
AD	0	0.00553090
BC	0	0.00747272
BE	0	0.00044
CD	0	0.000608
CE	0	0.00553090
DE	1	0.00553090

接下來螞蟻 a_3 在第四次的狀態轉移中只剩最後一個城市 B，故路徑 BE 的機率為 1，則更新區域費洛蒙及陣列 R_3[C,A,D,E,B]=90，結果如表 23 所示。

最後螞蟻 a_1 將走回原點 B，故路徑 AB 的機率為 1，則更新區域費洛蒙及陣列 R_1[B,E,C,D,A,B]=115，結果如表 24 所示。

表 21 螞蟻 a_1 在第二代的第四次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00553090
AC	0	0.000608
AD	1	0.00417163
BC	0	0.00747272
BE	0	0.00044
CD	0	0.000608
CE	0	0.00553090
DE	0	0.00553090

表 22 螞蟻 a_2 在第二代的第四次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00553090
AC	0	0.000608
AD	0	0.00417163
BC	0	0.00747272
BE	1	0.000608
CD	0	0.000608
CE	0	0.00553090
DE	0	0.00553090

最後螞蟻 a_2 將走回原點 D，故路徑 DE 的機率為 1，則更新區域費洛蒙及陣列 R_2[D,C,A,B,E,D]=110，結果如表 25 所示。

最後螞蟻 a_3 將走回原點 C 故路徑 BC 的機率為 1，則更新區域費洛蒙及陣列 R_3[C,A,D,E,B,C]=115，結果如表 26 所示。

表 23 螞蟻 a_3 在第二代的第四次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00553090
AC	0	0.000608
AD	0	0.00417163
BC	0	0.00747272
BE	1	0.0007256
CD	0	0.000608
CE	0	0.00553090
DE	0	0.00553090

表 24 螞蟻 a_1 在第二代的第五次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	1	0.00417163
AC	0	0.000608
AD	0	0.00417163
BC	0	0.00747272
BE	0	0.0007256
CD	0	0.000608
CE	0	0.00553090
DE	0	0.00553090

表 25 螞蟻 a_2 在第二代的第五次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00417163
AC	0	0.000608
AD	0	0.00417163
BC	0	0.00747272
BE	0	0.0007256
CD	0	0.000608
CE	0	0.00553090
DE	1	0.00417163

表 26 螞蟻 a_3 在第二代的第五次狀態轉移及區域費洛蒙更新

路徑	機率	費洛蒙
AB	0	0.00417163
AC	0	0.000608
AD	0	0.00417163
BC	1	0.00553090
BE	0	0.0007256
CD	0	0.000608
CE	0	0.00553090
DE	0	0.00417163

接著更新第二次迭代結束時的全域費洛蒙更新結果，顯示為表 27：

表 27 第二次迭代結束之全域費洛蒙更新

路徑	費洛蒙
AB	0.00810705
AC	0.00739432
AD	0.00083432
BC	0.00110618
BE	0.00741784
CD	0.00739432
CE	0.00110618
DE	0.00810705

之後如果需要更多的螞蟻數量，或是較多的城市，也是使用上面的方式建構路線，直到每隻螞蟻都走完各個城市，並回到起點，再更新費洛蒙量，以此達到最佳化螞蟻演算法。

2.2 求解連續函數的最佳化之螞蟻演算法

螞蟻演算法已在許多研究中被求證能有效的求解離散型的 NP 最佳化問題，旅行商問題(TSP)、背包問題(Knapsack problem)、圖著色問題(Graph coloring problem 簡稱 GCP)、二次分配問題(Quadratic assignment problem 簡稱 QSP)。而近期有許多學者在研究使用螞蟻演算法求解連續空間。

2.2.1 連續型螞蟻演算法介紹

陳燁在 2004 年指出求解連續型問題，大致可分為兩種方法：第一、將費洛蒙殘留離散的分佈到各條路徑上，再把空間劃分為幾個離散的區間，之後使用各個區間中最好的解來代表該區間的最佳化程度，最後再選出較好的區間，使用最佳化的演算法

來求解。第二、直接在連續解的空間上取值，因此必須要使用一定的函數來表示連續空間解上的費洛蒙殘留分佈[1]。

而陳燁在 2004 年的 paper[2]中使用的方法是先假一個自變量的精確到小數點後 d 位，所以自變量 x 就可以用 d 個 10 進的數字表示，因此如果精確度為 d 位，則會有 d+2 層的城市，每層將由 0~9 十個各位數表示，這也代表將會有 dx10+2 的城市，其中，加 2 代表城市的起始點與終點。而 d 層的城市，由左到右分別表示整數位、十分位、百分位…等，而只有 k-1 與 k 層間的城市會有路徑連接。即 10^1 只 10^0 連接、而 10^0 和 10^{-1} 有連接。支後只用和傳統的螞蟻演算法的移動規則以費洛蒙的更新則來對連續問題進行求解。

由圖 3 城市路徑示意圖可以看出陳燁 2004 年使用的方法，將會造成運算及記憶體的負擔，因為產生的路徑數，是 10 的 n 次方，這將是一個很大的數目。

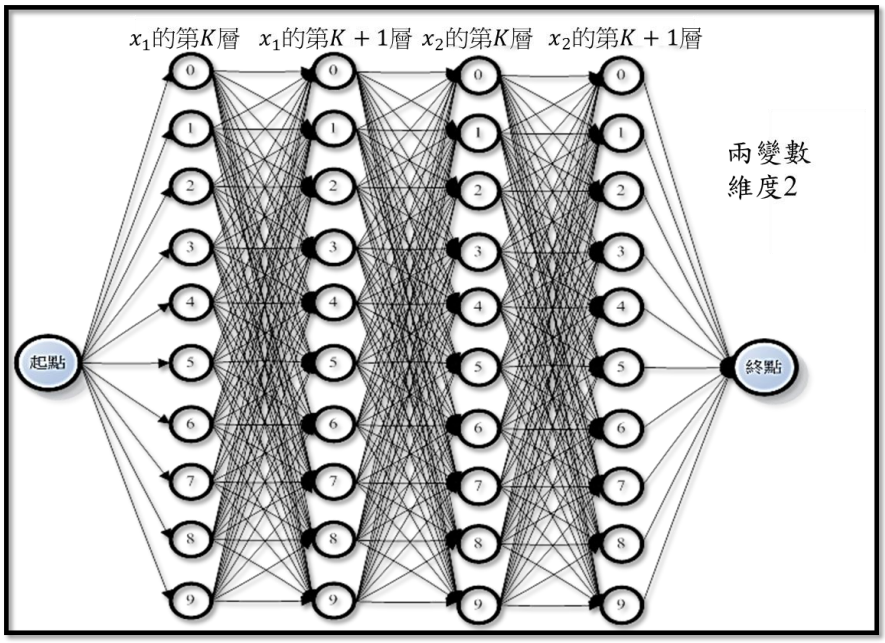


圖 3 城市路徑示意圖

在 2006 年 Toksari 對於連續型螞蟻演算法尋找全域最佳解，提出了演算法(ACO based algorithm，簡稱 ACO-BA)[11]，演算法步驟如下：

1. 初始設定，在搜尋空間隨機產生 m 組解，迭代次數，設定 dx 的範圍 $[-\alpha, \alpha]$，設定費洛蒙值 τ

2. 計算每組解的適合度值 f(x)

3. 判斷移動方向

$$\bar{x}_{initial}^{best} = x_{initial}^{best} + \left(x_{initial}^{best} \times 0.01\right)$$

如果 $f\left(\bar{x}_{initial}^{best}\right) \leq f\left(x_{initial}^{best}\right)$，則 $x_t = x_{t-1} + dx$

4. 更新費洛蒙

(1) $\tau_t = 0.1 \times t_{t-1}$

(2) $\tau_t = \tau_{t-1} + \left(0.01 \times f\left(x_{t-1}^{best}\right)\right)$

5. 找全域最佳解

如果 $f\left(x_t^{best}\right) \leq f\left(x_{t-1}^{best}\right)$則， $x^{global\min} = x_t^{best}$ ，否則不變

6. 更新[$-\alpha, \alpha$]的範圍

$$\alpha_k = 0.1 \times \alpha_{k-1}$$

7. 達終止條件

初始設定的迭代數

Toksari 在 2007 年又提出了(modified Ant Colony Optimization，簡稱 MACO)[12]，主要是改變了初始產生的解。

Baskan 等人在 2009 年，提出(ACO Reduced Search Space，簡稱 ACORSES)[13] 主要步驟如下：

1. 第一代在搜尋空間出始產生 m 組解，計算每組解的適合度值(舊的解)，找到最佳解 $f(x^{\text{best}})$，第二代開始搜尋空間改為 $x_{t-1}^{best} + \beta; x_{t-1}^{best} - \beta$

2. 蒸發費洛蒙

$$\tau_t = 0.1 \times t_{t-1}$$

3. 費洛蒙更新

$$\tau_t = \tau_{t-1} + \left(0.01 \times f\left(x_{t-1}^{best}\right)\right)$$

4. 判斷移動方向，對 m 組解產生新解

   $\bar{x}_{initial}^{best} = x_{initial}^{best} + \left(x_{initial}^{best} \times 0.01\right)$

   如果 $f\left(\bar{x}_{initial}^{best}\right) \le f\left(x_{initial}^{best}\right)$，則 $x_t = x_{t-1} + dx$

   dx 的空間為[$-\alpha, \alpha$]

5. 計算新解的適合度值，比較舊的解適合度與新的解適合度，適合度值較好的取代全域最佳解

6. 更新 α 、 β 值

   $\alpha_t = \alpha_{t-1} \times 0.99$

   $\beta_t = \beta_{t-1} \times 0.99$

而 Socha、Dorigo 等人在 2008[10]年也提出了使用螞蟻演算法求解連續型問題，而 ACO_R 演算法的主要步驟如下：

1. 初始隨機產生 k 組解(archive size)，每組解有 n 維度個值，如圖 4 所示。設定螞蟻數 m、迭代數 T、及其他參數初始值。

2. 計算每組解的適合度值(Fitness Value)，並由小到大排序每組解的適合度值，$f(x_1) \le f(x_2) \le \cdots \le f(x_l) \le \cdots \le f(x_k)$及算出每組解權重 ω_l，公式如下：

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}} \tag{6}$$

   其中，q 為學習率，其值介於 0~1 之間，k 為 archive size，l 為排序後的順序值；$\omega_1 \ge \omega_2 \ge \cdots \ge \omega_l \ge \cdots \ge \omega_k$ 。

3. 利用輪盤法(Roulette wheel)，計算每組解的權重 ω_l 在輪盤上的比例 P_l，公式如下：

$$P_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r} \tag{7}$$

而輪盤上的機率代表每組解被選重的機率，而被選重的解用來建構下一代的解，而權重值越大，代表著被選重的機率也較大。

4. 利用常態分配 $N(\mu_l^i, \sigma_l^i)$產生 m 組新的解：

$$x_l'^i = x_l^i + N_l^i\left(\mu_l^i, \sigma_l^i\right) \tag{8}$$

l 代表被選重的第 l 組解，μ_l^i 代表平均數，σ_l^i 代表標準差， i 代表第 l 組解的維度：

$$\mu^i = \{\mu_l^i, \ldots, \mu_k^i\} = \{s_l^i, \ldots, s_k^i\} \tag{9}$$

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{\left| s_e^i - s_l^i \right|}{k-1} \tag{10}$$

其中 ξ 代表費洛蒙的蒸發率，其值介於 0~1 之間。

5. 費洛蒙表的更新，將 k 組解的適合度值與新產生的 m 組解的適合度值做比較，如果新產生的解之適合度的值比原先 k 組解的適合度的值還小，則將原先解的值替換掉，如果沒有，則無需變動 k 組解的值，這將維持 archive size k。

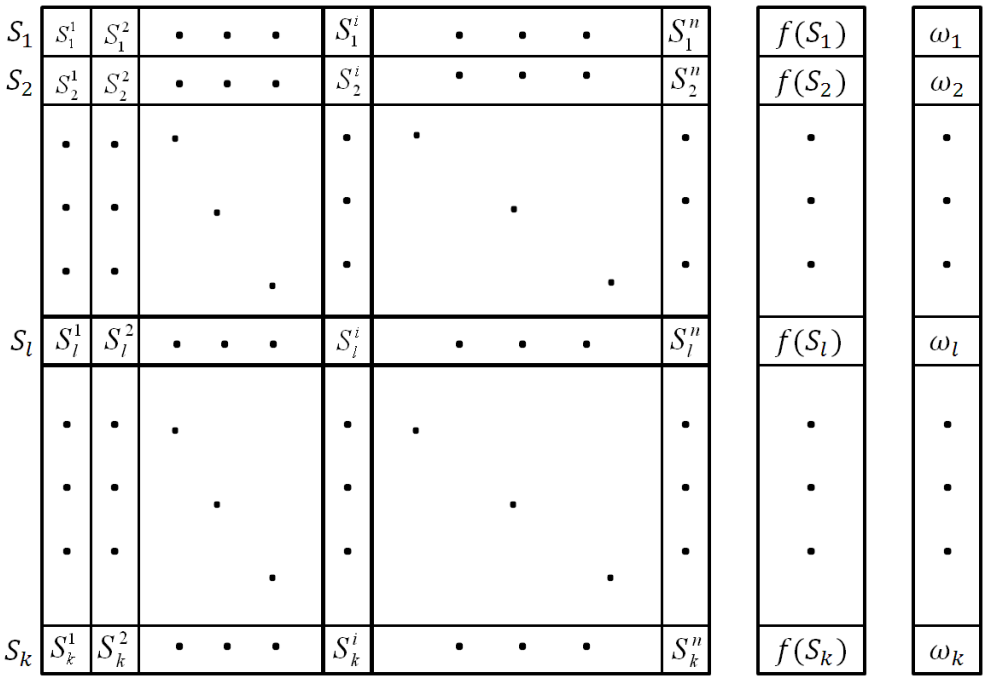


圖 4 ACO_R 的費洛蒙表

螞蟻演算法解連續型問題，流程圖如圖 5 所示：

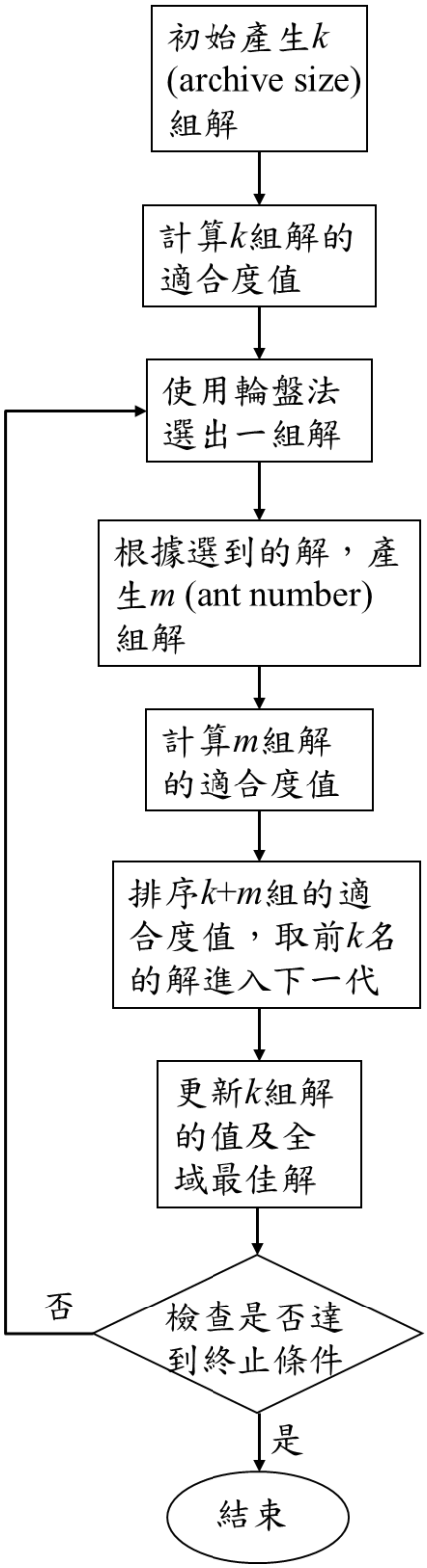


圖 5 續型螞蟻演 ACO_R 算法流程圖

而 Seçkiner,等人在 2013 年提出新型的更新費洛蒙公式[14]，步驟如下：

1. 找到出始最小的適合度值，計算每組解與最小適合度值的距離。

$$D_i = f_i - f_{\min} \tag{11}$$

2. 正規化，將各步驟一求出的各組值，帶入高斯分配，計算各組的正規化值，其中 t 為標準差。

$$\phi_i = e^{-\frac{D_i^2}{2t}} \tag{12}$$

3. 計算下一代產生螞蟻數量的比重，*m* 為螞蟻數

$$\tau_i = \frac{\phi_i}{\sum_{i=1}^{m} \phi_i} \tag{13}$$

4. 依照上一步驟的比重，產生新的解 *dx* 為初始[-α,α]中，隨機找尋的值

$$x_T^k = x_{T-1}^k \pm dx \tag{14}$$

5. 將 α 的值縮小 0.9 倍，T 為代數

$$\alpha_T = 0.9 \times \alpha_{T-1} \tag{15}$$

2.2.2 連續型螞蟻演算法 ACO_R 簡易範例

本節將使用 Socha、Dorigo 在 2008 年使用的 ACO_R 螞蟻演算法，模擬 *sphere* function 的實驗步驟，其中參數設定為 D(維度)=2，*k*(archive size)=5，*m*(螞蟻數)=2。其中 *sphere* function 的式子如下：

$$f_{sphere} = \sum_{i=1}^{2} x_1^2 + x_2^2 \tag{16}$$

步驟一：亂數產生五組解 S_1~S_5，每組解裡有兩個值 S^1、S^2。

	S^1	S^2
S_1	-99.74974	12.71706
S_2	-61.33915	61.74810
S_3	17.00186	-4.02539
S_4	-29.94170	79.19248
S_5	64.56801	49.32096

步驟二：帶入式子(16)，計算每組解的適合度值 $f(S_1 \sim S_5)$，並由小到大排序。

	S^1	S^2	$f(S_i)$
S_1	17.00186	-4.02539	305.26707
S_2	64.56801	49.32096	6601.58538
S_3	-29.94170	79.19248	7167.95490
S_4	-61.33915	61.74810	7575.31947
S_5	-99.74974	12.71706	10111.73595

步驟三：根據式子(7)，計算每組解的權重大小 $\omega_1 \sim \omega_5$，並由大到小排序。

ω_i
0.797884560802839090
0.107981933026372600
0.000267660451529762
0.000000012151765700
0.000000000000010105

步驟四：使用輪盤法挑出一組解，假設挑重第 1 組解，將選到的解，根據公式(9)、(10)求出平均數與標準差，帶入式子(8)產生 m 組解，並計算適合度值 $f(m_1,m_2)$。

	m^1	m^2	$f(m_i)$
m_1	16.95034	-116.07362	13760.39929
m_2	17.77212	-3.73430	329.7932458

步驟五：將 m 組解的適合度值，與 k 組的適合度值比較，把適合度值較差的替換掉，最後更新 k 組解。

	S^1	S^2	$f(S_i)$
S_1	-99.74974	12.71706	305.26707
S_2	17.77212	-3.73430	329.7932458
S_3	-61.33915	61.74810	6601.58538
S_4	17.00186	-4.02539	7167.95490
S_5	-29.94170	79.19248	7575.31947

2.3 螞蟻演算法的特性

螞蟻演算法有許多特性：

- 螞蟻獨立性：每隻螞蟻都是個獨立的搜尋單位，彼此平行運作。
- 經驗分享：之前螞蟻留下透過費洛蒙，提供下隻螞蟻選擇的依據。
- 回饋機制：正回饋方面，較短的路徑將會留下較多的費洛蒙，提供較好的經驗供往後搜尋，負回饋方面，藉由蒸發的機制，避免過早陷入區域最佳解。
- 簡易性：透過費洛蒙濃度的深淺，判斷最佳路徑，因此可以稍作修改後，便可求解不同問題。

第三章　改良的連續型螞蟻演算

以上在求解高維度的連續型問題時都有一些缺陷：

- 陳燁所使用的方法，會造成運算及記憶體的負擔，因為產生的路徑數，是 10 的 *n* 次方，這將是一個很大的數目。
- Dorigo 所使用的蒸發率，為一個固定值，且標準差只計算一次。

本研究將針對第二點作改良，提出更有效的螞蟻演算法，並求解高維度的問題。

3.1　蒸發率改良

針對費洛蒙的蒸發率 ξ，Socha、Dorigo 等人在 2008 的 paper 裡，提出蒸發率為一項常數係數，值為 0.85。而本研究的演算法將依照下式作改良。

$$\xi = \exp\left(\frac{\mathrm{SZ}}{\mathrm{SZ} - \left(\frac{\mathrm{SZ} - \mathrm{EZ}}{\mathrm{GN}}\right) * g}\right) \tag{17}$$

- SZ：(起始移動大小)=log((Ub-Lb)/pow(10.0, 1))
- EZ：(結束移動大小)=log((Ub-Lb)/pow(10.0, 15))
- Ub、Lb 為搜尋空間的上界與下界
- GN：為迭代總次數
- g 隨著迭代次數線性遞增，直到達最大迭代次數

由式子 17 可以看出，蒸發率的值在一開始很大，但隨著 g 的值遞增之後，蒸發率，迅速的遞減，如圖 6 所示，到最後一代時，值非常靠近 0(起始移動大小/結束移動大小)，使得在計算標準差時，在最後的階段能迅速的收斂，並加速找到全域最佳解。而 Socha、Dorigo 等人在 2008 的 paper 裡所解決的問題函式，維度都較低，因此我們將使用改良過後的蒸發率，來解決維度較高的問題，預期會達到較精準的最佳值。

3.2 標準差改良

Dorigo 等人在 2008 的 paper 中，提出計算標準差的方法為：使用輪盤法挑重一組解，計算那組解在每個維度上與其它組解的距離總和在除以(archive size-1)，最後再乘上蒸發率，做為每個維度的標準差。

而本研究將加入 ES(演化策略)的突變公式，將之前計算過的標準差乘上指數函數，使得標準差能經過多次計算，公式如下：

$$\sigma_l'^i = \sigma_l^i \exp\left(\tau' N(0,1) + \tau N_l^i(0,1)\right) \tag{18}$$

其中，τ'=5、τ=9，參數值為經過實驗後的最佳值。

之後再使用上式求出來的標準差，帶入常態分配產生新的解而，產生新值的公式如下：

$$x_l'^i = x_l^i + N_l^i\left(0, \sigma_l'^i\right) \tag{19}$$

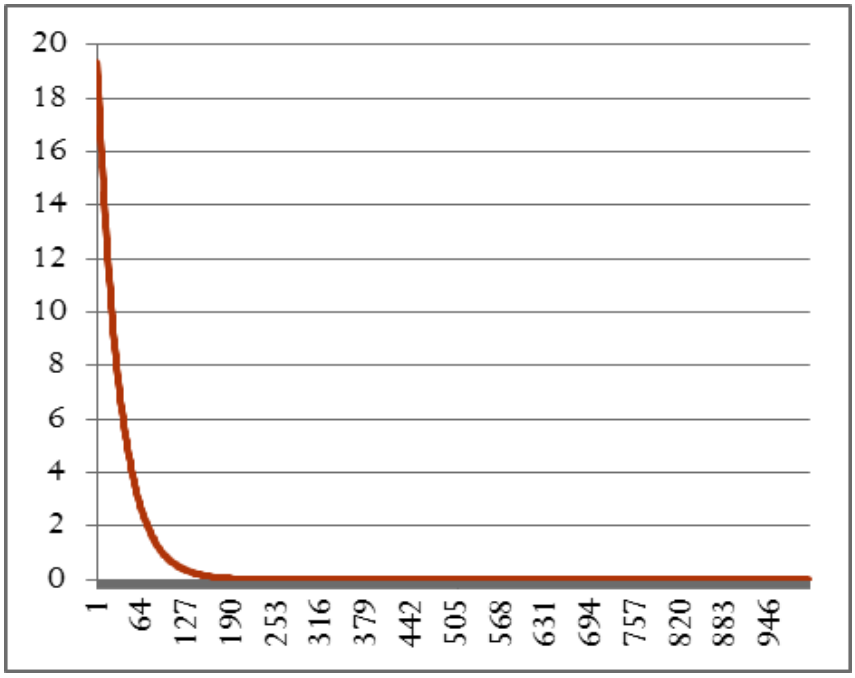


圖 6 蒸發率變化圖

第四章　實例分析

本研究之演算法因加入 ES 的突變機制，故稱為 ACORES，其中

ACO 代表 Ant Colony Optimization

R 代表 Real number

ES 代表 Evolution Strategies

4.1　演算法驗證

欲求解最佳化問題，有許多方法，像是只用套裝軟體，或是使用自行撰寫的程式進行求解，但是套裝軟體存在著許多的風險，因此列出了下列幾點，判斷演算法好壞的條件：

1. 有效性，必須是可執行、可編譯，如果連最基本的要求都無法達成，更不需討論求解了。
2. 有限性，必須在可接受的時間限制，或是可支援的硬體空間下，產生結果，如果一個程式碼須執行很久，那這樣的程式碼的效能將會大打折扣，因為現今有許多問題是有時間限制的。而如果超出許多硬體的支援，則程式碼能使用的範圍也將有所侷限。
3. 穩定性，現今有許多問題，非常的複雜，有許多區域最佳解，但是一個良好的演算法，應具備跳出區域最佳解的能力，並獨力執行多次後的結果應差距不大，如果每次的執行結果，誤差都很大，那演算法的可信度將會降低很多。
4. 準確性，當以上條件都符合時，結果的精確度就變得相當重要，一個良好的演算法，其結果，應該要有一定的精確度，否則和其他演算法比較起來，將沒有太突出的貢獻。
5. 完整性，一個良好的演算法，應該具備良好的適應能力，不該因為問題複雜程度提升，而無法對於問題求解，因為每個問題的複雜程度不盡相同，限制

也都不一樣，如果一個演算法，只能求解一個問題，那可用性就不大了。

6. 易用性，套裝軟體，因為有版權的關係，價格較貴，限制的條件又多，解程式碼無法進行更改。而自行撰寫的演算法可自行調整參數，使其改善執行時間，或是結果的精確度。

根據上列的條件，我們將使用許多不同的函式、不同維度大小的問題進行測試，並將測試結果分為四大部分，詳細結果將在下節敘述。

4.2 實驗結果

本研究的測試函數主要分為四大部分

1. Dorigo 在 2008 年提出連續型螞蟻演算法所測試的函式[10]

   是由 Socha, Krzysztof, and Marco Dorigo 在 2008 年提出的 paper ‘Ant colony optimization for continuous domains’.所測試的許多函式。

2. 低維度的 10 個 benchmark[14]

   是由 Seçkiner, S. U., 等人在 2013 年提出的 paper‘Ant colony optimization for continuous functions by using novel pheromone updating’.所使用的 10 個低維度測試函式。

3. 常見的 23 個 benchmark[15]

   是由 M. M., Yao, X., 在 2012 年提出的 paper ‘Diversity Guided Evolutionary Programming: A novel approach for continuous optimization’.所使用的 23 個測試函式。前 13 個函式維度皆為 30，後 10 個為低維度函式。

4. CEC2008 的 6 個 benchmark[22]

   是一個 2008 年在香港舉辦的智能計算的世界大會，所使用的測試函式，主要對於高維度問題進行求解。

4.2.1 ACO_R 連續型螞蟻演算法測試的函式－實驗結果(一)

這部分的實驗結果，分為三小部分

- Dorigo(2008)與 Kern(2004)比較的測試函式，如附表 5

表 28 實驗結果(一)－與 Dorigo、Kren 比較結果

Formulation	ACORES	ACO_R	CMA-ES
PL	1.5	**1.0 (175)**	6.3
DP	40	**1.0 (170)**	6.4
SP	**1.0 (991)**	1.5	1.8
EL	**1.0 (972)**	12	4.6
CG	1.2	1.4	**1.0 (3840)**
TB	1.3	**1.0 (2567)**	1.7

※CMA-ES－ES with Covariance Matrix Adaptation

- 與不同的螞蟻演算法測試函式，如附表 6、附表 7

表 29 實驗結果(一)－不同螞蟻演算法比較結果

Formulation	ACORES	ACO_R	CACO	CIAC
R_2	2.3	**1.0 (820)**	8.3	14
SM	**1.0 (277)**	2.8	79	180
Gr_{10}	1.2	**1.0 (1390)**	36	36
GP	**1.0 (166)**	2.3	32	141
MG	**1.0 (185)**	1.9	9.3	63
B2	**1.0 (184)**	2.9	－	65
$S_{4,5}$	**1.0 [45%] 252**	3.1 [57%]	－	156 [5%]

※CACO－Continuous ACO

CIAC－Continuous Interacting Ant Colony

➢ 與其他演算法比較的測試函式，如附表 6、附表 7

表 30 實驗結果(一)－其他演算法的比較結果

Formulation	ACORES	ACO_R	CGA	ECTS
RC	**1.0 (109)**	7.9	5.6	2.2
B2	**1.0 (184)**	3	2.3	―
ES	**1.0 (209)**	3.7	7	―
GP	**1.0 (166)**	2.4	2.5	1.4
R_2	4	1.7	2	**1.0 (480)**
Z_2	**1.0 (105)**	2.8	5.9	1.8
DJ	**1.0 (166)**	2.4	4.5	―
$H_{3,4}$	**1.0 (151)**	2.3	3.9	3.6
$S_{4,5}$	**1.0 [45%] 252**	3.1 [57%]	2.4 [76%]	3.4 [75%]
$S_{4,7}$	**1.0 [45%] 178**	4.2 [79%]	3.8 [83%]	4.9 [80%]
$S_{4,10}$	**1.0 [55%] 264**	2.7 [81%]	2.4 [83%]	3.4 [80%]
Z_5	**1.0 (415)**	1.7	3.3	5.4
$H_{6,4}$	**1.0 (229)**	3.1	4.1	6.6

※CGA－Continuous Genetic Algorithm

ECTS－Enhanced Continuous Tabu Search

本部分的實驗的結果，是依據每個演算法測試函式時，獨立執行 n 次，計算達到最佳解(終止條件)時的最大函式計算次數(FEs)，並計算 n 次 FEs 之平均，將平均值最低的演算法用 1.0 表示，括號代表該演算法的平均 FEs 值，其餘演算法，則除以最佳計算次數表示之。－代表沒有資料。由於 Shekel 函式，容易陷入區域最佳解，因此多計算了成功率。

表 28 的終止條件為，$\left|f-f^*\right|<\varepsilon_{\min}$，其中，$f$ 代表目前的最佳解，f*代表問題的全域最佳值，而 $\varepsilon_{\min}=10^{-10}$。

表 29、表 30 的終止條件為，$\left|f-f^*\right|<\varepsilon_1 f+\varepsilon_2$，其中，$f$ 代表目前的最佳解，f*代表問題的全域最佳值，而 $\varepsilon_1=\varepsilon_2=10^{-4}$。

4.2.2 低維度的 10 個 benchmark－實驗結果(二)

ACO_NPU 為 paper 所使用的演算法[14]，ACORES 為本研究演算法。本研究將設定與 paper 相同的螞蟻數、archive size 、迭代數，如 Seçkiner(2013)測試的 benchmark 附表 1。測試在低維度時，ACORES 是否能穩定收斂至最佳值。

表 31 實驗結果(二)

Formulation	ACO_NPU			ACORES		
	Best *f*(x)	Best x	Best y	**Best *f*(x)**	Best x	Best y
f_1	**-3**	3		**-3**	3	
f_2	**4.59E-98**	2.64E-21		**4.17E-92**	-1.43E-23	
f_3	**0**	3	3	**0**	3	3
f_4	**0**	1	1	**0**	1	1
f_5	**-10**	-10	-1.82E-17	**-10**	-10	4.70E-17
f_6	**0**	-9.52E-12	-9.52E-12	**0**	4.27E-11	3.04E-10
f_7	**-837.9658**	420.9687	420.9687	**-837.9658**	420.9687	420.9703
f_8	**-2**	1.06E-05	1.60E-05	**-1.99**	8.91E-03	6.6E-04
f_9	**1**	3	4	**1**	3	4
f_{10}	**-1.0316**	-0.0898	0.7125	**-1.0316**	-0.0898	0.7127

由結果可以看出，ACORES 在求解低維度的問題時，能穩定收斂至最佳解。

4.2.3 常見的 23 個 benchmark－實驗結果(三)

這個部份的實驗結果分為兩小部份

- 前 13 個 benchmark，如附表 2，維度都為 30，FEs 為 150,000
- 後 10 個 benchmark，如附表 3，是低維度，FEs 為 10,000

※ FEs：函式的計算次數。主要是讓不同演算法能在一個公平的機制下作比較。

每個 benchmark 獨立執行 30 次，並計算平均數與標準差。

表 32 實驗結果(三)－前 13 個 benchmark

Minimum value						
	CEP		DGEP		ACORES	
	Mean	Dev	Mean	Dev	Mean	Dev
f_1	2.2E-04	5.9E-04	5.42E-08	1.48E-08	**4.80E-23**	2.26E-23
f_2	2.6E-03	1.7E-04	6.64E-12	3.61E-12	**1.48E-12**	2.78E-13
f_3	5.0E-02	6.6E-02	4.13E-08	9.25E-09	**1.56E-23**	4.04E-24
f_4	2.0	1.2	1.06	0.32	**9.22E-06**	8.73E-06
f_5	6.17	13.61	**1.28**	0.76	39.14	34.56937
f_6	577.76	1125.76	**0**	0	**0**	0
f_7	1.8E-02	6.4E-03	1.94E-12	8.23E-13	**3.59E-30**	4.54E-30
f_8	-7917.1	634.5	-12567.4	0.18	**-12569.5**	5.457E-12
f_9	89.0	23.1	1.56E-12	5.12E-9	**0**	0
f_{10}	9.2	2.8	**2.47E-16**	6.38E-17	1.65E-12	2.8E-13
f_{11}	8.6E-02	0.12	7.52E-14	2.54E-14	**0**	0
f_{12}	1.76	2.4	3.32E-12	1.94E-13	**9.53E-26**	4.79E-26
f_{13}	1.4	3.7	1.74E-04	2.65E-05	**1.40E-24**	6.08E-25

※粗體字為三個演算法比較過後的最佳解。Mean 為平均數，Dev 為標準差。

表 33 實驗結果(三)－後 10 個 benchmark

Minimum value						
	CEP		DGEP		ACORES	
	Mean	Dev	Mean	Dev	Mean	Dev
f_{14}	1.66	1.19	1.02	0.04	**0.998**	2.22E-16
f_{15}	4.7E-04	3.0E-04	**2.17E-04**	8.2E-05	7.72E-04	6.65E-05
f_{16}	**-1.031**	4.9E-07	**-1.031**	0.00	**-1.031**	8.14E-08
f_{17}	**0.398**	1.5E-07	**0.398**	2.4E-07	**0.398**	1.67E-16
f_{18}	**3.0**	0	**3.000**	0.000	**3.0**	6E-15
f_{19}	**-3.86**	1.4E-02	**-3.86**	9.4E-03	**-3.86**	1.78E-15
f_{20}	-3.28	5.8E-02	**-3.32**	2.5E-04	**-3.32**	0
f_{21}	-6.86	2.67	**-9.87**	0.54	-5.4582	2.56172
f_{22}	-8.27	2.95	**-10.47**	0.08	-2.92768	2.585283
f_{23}	-9.10	2.92	**-10.51**	3.8E-02	-2.49723	1.732236

※粗體字為三個演算法比較過後的最佳解。Mean 為平均數，Dev 為標準差。

CEP－Classical Evolutionary Programming

DGEP－Diversity Guided Evolutionary Programming

4.2.4 CEC2008 的 6 個 benchmark－實驗結果(四)

使用的 6 個測試函數，如附表 4，其中 $z=(x-o), x=[x_1, x_2, ...x_D], o=[o_1, o_2, ..., o_D]$。測試的維度為 1000 維，FEs=5000000，k=m=50，每個問題獨立執行 25 次。

表 34 實驗結果(四)

FEs \ Prob		1	2	3	4	5	6
5.00e+4	1st	1.04E+06	1.58E+02	4.51E+11	1.27E+06	2.01E+04	2.16E+01
	7th	1.11E+06	1.64E+02	5.13E+11	1.32E+06	2.02E+04	2.16E+01
	13th	1.14E+06	1.65E+02	5.39E+11	1.34E+06	2.02E+04	2.16E+01
	19th	1.17E+06	1.69E+02	5.58E+11	1.36E+06	2.03E+04	2.16E+01
	25th	1.28E+06	1.71E+02	6.45E+11	1.40E+06	2.04E+04	2.16E+01
	Mean	**1.14E+06**	**1.66E+02**	**5.37E+11**	**1.34E+06**	**2.02E+04**	**2.16E+01**
	Std	**5.17E+04**	**3.29E+00**	**4.23E+10**	**3.18E+04**	**7.28E+01**	**6.88E-03**
5.00e+5	1st	3.60E+03	1.38E+02	1.34E+07	4.27E+04	1.79E+04	2.16E+01
	7th	3.98E+03	1.40E+02	1.55E+07	4.56E+04	1.79E+04	2.16E+01
	13th	4.16E+03	1.43E+02	1.61E+07	4.65E+04	1.79E+04	2.16E+01
	19th	4.27E+03	1.46E+02	1.70E+07	4.75E+04	1.79E+04	2.16E+01
	25th	4.46E+03	1.52E+02	1.90E+07	5.21E+04	1.80E+04	2.16E+01
	Mean	**4.12E+03**	**1.44E+02**	**1.63E+07**	**4.66E+04**	**1.79E+04**	**2.16E+01**
	Std	**2.21E+02**	**3.67E+00**	**1.38E+06**	**1.88E+03**	**3.29E+01**	**4.96E-02**
5.00e+6	1st	2.05E-12	1.29E+02	1.09E+03	3.00E+04	1.77E+04	2.00E+01
	7th	2.16E-12	1.32E+02	1.14E+03	3.28E+04	1.77E+04	2.00E+01
	13th	2.22E-12	1.35E+02	1.24E+03	3.35E+04	1.77E+04	2.00E+01
	19th	2.33E-12	1.38E+02	1.28E+03	3.45E+04	1.78E+04	2.00E+01
	25th	2.44E-12	1.42E+02	1.37E+03	3.94E+04	1.78E+04	2.00E+01
	Mean	**2.25E-12**	**1.36E+02**	**1.23E+03**	**3.38E+04**	**1.77E+04**	**2.00E+01**
	Std	**9.66E-14**	**3.56E+00**	**8.43E+01**	**1.84E+03**	**1.67E+01**	**7.86E-06**

※1st 代表最好的解，25th 代表最差的解。Mean 為平均數，Std 為標準差。

實驗結果數據為 $f(x)-f(x^*)$ 的誤差表，$f(x)$ 代表目前最佳解，$f(x^*)$ 代表問題的全域最佳解。並排序每個問題在 FEs 為 50000(第 999 代)、500000(第 9999 代)、5000000(第 99999 代)時的最佳解，並記錄 25 次由最好到最差的解。

第五章　結論與建議

5.1　結論與研究貢獻

本研究，針對之前學者提出的螞蟻演算法之部分缺點，進行改良，並對於連續型問題，進行求解。結果與貢獻如下：

1. 改良固定蒸發率，傳統演算法在初始設定蒸發率到結束時，蒸發率為一定值，這將使得在初期，可以需要花較多的時間，才能找到較好的解，而在後期，卻可能陷入區域最佳解，因此改良後，讓蒸發率能動態的由大漸漸變小，使得在找尋最小值時，能快速的收斂至最佳解。
2. 改良標準差的公式，本研究將使用 Dorigo 在 2008 年[10]使用的標準差，並將算出的結果，結合傳統 ES 的標準差算法，讓標準差計算多次，最後結果能找到更精準的解。
3. 使用改良式螞蟻演算法求解高維度的連續型問題，結果達到預期的效果，讓近期無人測試大維度連續型問題，有了較多的資訊。

5.2　未來研究方向

未來研究，可以由下列幾點進行：

1. 有許多參數是固定的，如：學習率 q、τ 等，往後可以嘗試始用動態的方法，或測試不同的值。
2. 本方法使用輪盤法，挑種一組解來產生 m 組螞蟻數，這可能會導致多樣性不足，可以嘗試值選擇多組解來產生 m 組解，提升解的多樣性，避免過早收斂。
3. 針對 CEC2008 的問題，持續測試個參數的設定值，找到最佳的解。

5.3 加入 GPU

由於現今的製程技術不斷的提升，而 CPU 的尺寸也越做越小，但是有許多問題也不斷的出現，例如：散熱的問題、記憶體的平寬等。還有一個問題是，雖然 CPU 的時脈週期(clock cycle)仍然持續的在增加，但是增加的速度已大不如前，漸漸的趨於平緩。由於以上的眾多問題，使得 GPU 的應用變得相當重要，雖然在現今的電腦架構中，大部分的工作還是交給 CPU 執行，如：檔案的讀寫、程式的執行、I/O，這使得 CPU 花在運算的時間相對減少許多，而 GPU 在處理平行運算的能力優於 CPU，且 GPU 的價格並不會太過於昂貴，漸漸的成為計算的主流。因此往後研究可以使用 CUDA 的技術，結合 CPU 與 GPU 的運算能力來對問題進行求解，以減少運算時間找尋更精確的最佳解。

而近幾年來在平行運算上有突出表現的研究為 Graphic Processing Unit 簡稱：GPU，它是由 NVIDIA 在 1999 年所堆出的一項技術，起初是使用在圖像處理，負責繪圖運算，但由於 GPU 有高密度的運算能力，能同時間執行多組運算試，因此 NVIDIA 又在 2007 年推出 CUDA 的技術，讓程式設計師可使用 C/C++語言來進行撰寫程式碼，再透過 NVCC 編譯器編譯後，交給 GPU 進行平行運算。因為有這項技術的產生，使得電腦運算時間能有更大的突破，同時也能減少許多資源的消耗，讓最佳化的問題得以更快速的求出最佳解。

5.3.1 GPU 的運算

GPU 主要功能是用來處理電腦上的影像運算，由 NVIDIA 自 1999[21]所推出的，最初的 GPU 只被使用於 3D 圖像的描繪，可用性並不大，直到 CUDA 的技術出現，使得 GPU 的應用領域大幅的提升許多，讓許多複雜的計算能夠平行化。

現今電腦中的處理器多為中央處理器(Central Processing Unit，CPU)和 GPU 所組成，而 GPU 只負責繪圖，大部分的工作還是交給 CPU 執行，但是 GPU 在處理平行

運算的能力優於 CPU，且 GPU 的價格並不會太過於昂貴。隨著 CPU 在硬體方便可支援的能力已慢慢達到極限，且現今有許多複雜且重複性高的問題產生，平行運算的重要性已經是現在熱門的研究議題。

GPU 是由一組 SIMT (Single Instruction Multiple Threads)所組成，而 SIMT 裡面包含許多的 Streaming Multiprocessors (SM)，每個 SM 則是由 Streaming Processors (SP)所組成，如圖 9 所示。

表 35 GPU 發展

年代	說明	年代	說明
1993	黃仁勛、Curtis Priem、Chris Malachowsky 共同創立 NVIDIA 公司	2004	推出 GeForce 6800
1995	發表首款產品 NV-1	2006	推出 GeForce 8，GPU 開始支援 CUDA
1997	發布 NV-3，即 RIVA-128 晶片	2007	正式發布 CUDA 架構
1999	發布全球第一顆 GPU－GeForce-256	2008	發布 GeForce9 系列及 GT200，行動處理器：Tegra 系列
2001	推出第一款可編譯的 GPU－GeForce 3	2009	CUDA GPU 的新一代架構－“Fermi”
2002	推出 GeForce 4	2010	Fermi 世代的 GPU－GeForce GTX 480、GTX470，Tesla 系列 GPU－C2050、M2050、S2050

5.3.2 CPU+GPU 的異質平行運算

異質計算(Heterogeneous Computing)，簡單的定義為：使用多個不同架構的指令集的計算單元，組合起來的運算。為何會有這種架構的運算出現，原因就在於傳統的

運算方式屬於循序式，即一個指令完成後才執行下一個指令，但是此種運算方式在遇到運算量較大且複雜的問題時，會消耗非常龐大的資源及時間。如果可以在同時間執行多個指令，達到平行運算的功能，將會大幅減少運算時間，而每個指令的特性與複雜度不盡相同，因此需要能有效的同時執行不同的指令達到快速的平行化效果，這將是異質平行化計算的最大好處。

而 CPU 和 GPU 的就是現今很常見的異質運算。雖然 CPU 和 GPU 在電腦上都具有的處理運算的能力，但是兩者的特性與使用的地方卻有很大的差異。CPU 負責處理邏輯性強、流程控制較複雜及擁有大量迴圈的指令，GPU 則是負責處理需要有高密度計算的圖形描繪。由於 CPU 需要滿足各種運算來達到通用性，又要兼具平行性，因此平行的能力當然就無法很突出。CPU 主要是透過大量的快取記憶體來提高執行效率，希望用很低的延遲獲得指令與資料。而 GPU 需應付 2D、3D 的複雜運算與顯示，所以 GPU 需要較高的記憶體頻寬，和許多的執行單元，如圖 7 所示。

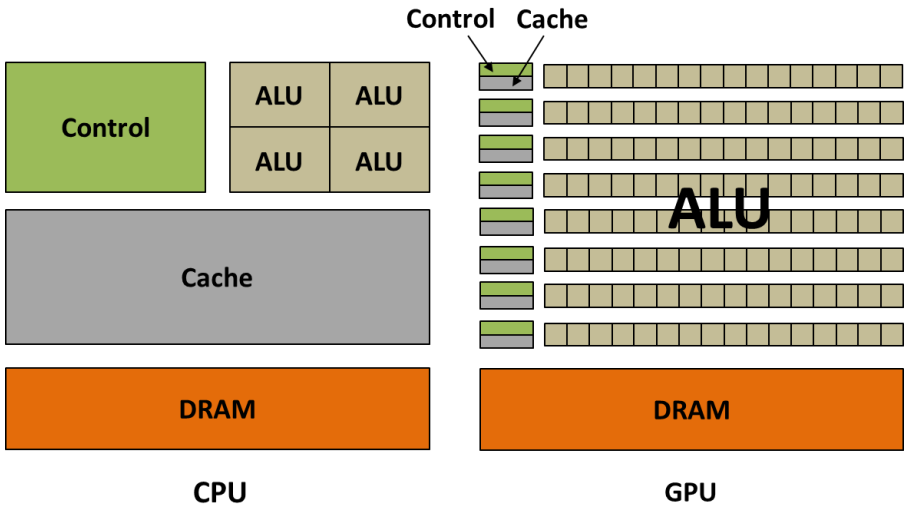


圖 7 CPU 與 GPU 的硬體架構

5.3.3 CUDA 技術

CUDA 的全名為：Compute Unified Device Architecture，統一運算裝置架構，是 NVIDIA 由 2007 年所提出的一種整合應用開發技術，程式設計師可使用 C 語言來進行撰寫，再透過 NVCC 編譯器編譯後，交給 GPU 進行平行運算，如圖 8 所示。

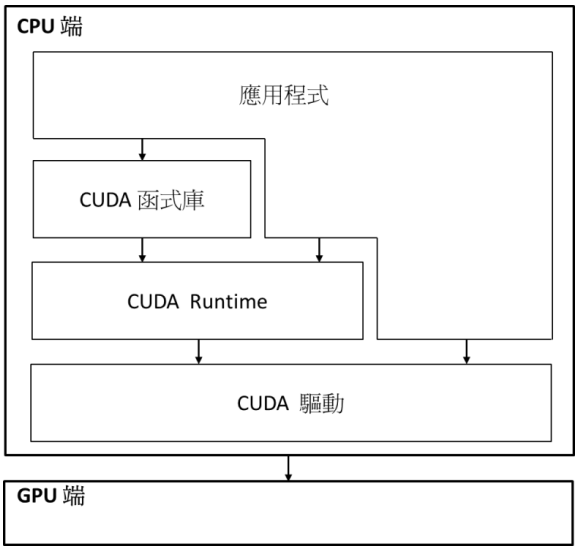


圖 8 CUDA 程式架構

架構

CUDA 的程式將分成兩部份：

(1) 主機端(Host)，為 CPU 端，主要是處理邏輯問題，以及串行計算

(2) 裝置端(Device)，為 GPU 端，主要在執行緒的平行運算

CPU 及 GPU 都有各自的記憶體空間及位址，互相獨立，在此 CPU 之記憶體稱為主記憶體，而 GPU 的記憶體稱為顯示記憶體。欲使用顯示記憶體時，則需要呼叫 CUDA API 的記憶體管理函式。

執行在 GPU 上的平行化程式稱為 kernel(核心函式)，而 kernel 的以網格(Grid)的形式所組成，每個網格裡又有若干個執行緒的區塊(block)，每個 block 是由若干個執行緒(thread)所組成，假設有 4 個區塊，每個區塊裡有 64 個執行緒，則結構如圖 9 所示。

而 kernel 就是在平行化各個 block，每個 block 各自獨立，沒有執行的先後順序，且無法相互溝通，GPU 的平性計算就是靠著同時間多個 block 執行運算來檢哨運算時間達到平行化的效果。

記憶體

在 GPU 上，CUDA 也使用了許多不同的記憶體模式，表 36 為六種不同記憶體的比較及簡單的功能說明，架構如圖 10 所示。

表 36 各記憶體的說明與比較

記憶體	位置	有無快取	存取模式	功能說明
shared memory	on block	N/A	可讀/寫	快速、供同一個區的執行續存取
register	on thread	N/A	可讀/寫	快速、延遲性低
local memory	on thread	No	可讀/寫	存取慢、存放大型的陣列及結構
constant memory	thread、host	Yes	裝置端：可讀 主機端：可讀/寫	快速、空間小、存放使用率高的唯讀參數
texture memory	thread、host	Yes	裝置端：可讀 主機端：可讀/寫	以陣列形式存放、適用於大量資料的隨機存取
global memory	thread、host	No	裝置端：可讀/寫 主機端：可讀/寫	延遲性高、CPU 和 GPU 都可讀寫

註：N/A 表示 GPU 上的高速記憶體

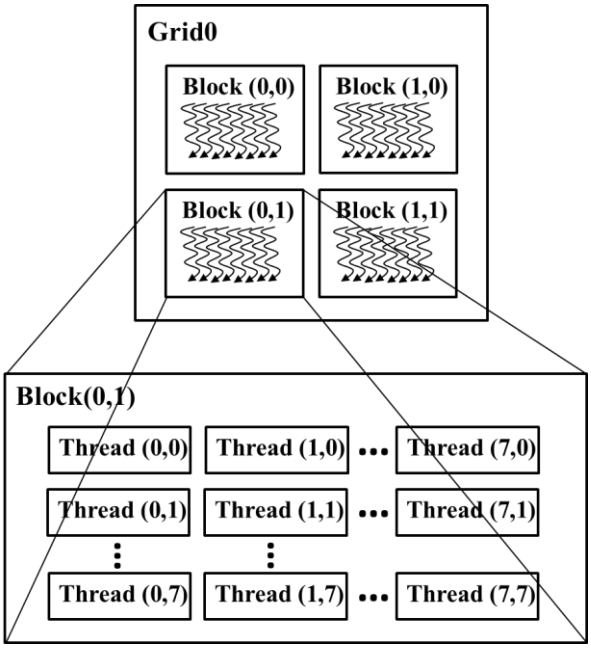


圖 9 執行緒結構

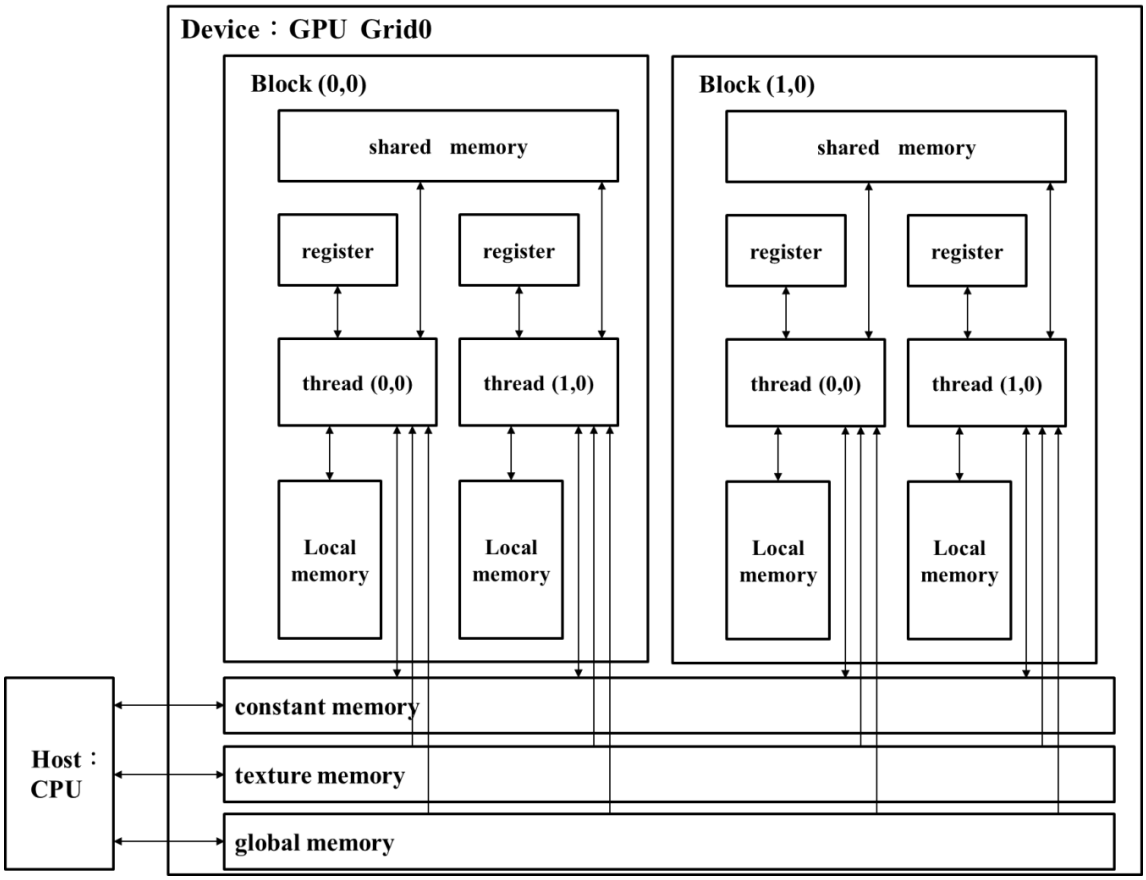


圖 10 記憶體架構

參考文獻

中文文獻

[1] 汪镭、吴启迪 ，「蚁群算法在连续空间寻优问题求解中的应用」，控制與應用，2003，18 卷 1 期，P45~48。

[2] 陈烨，用于连续函数优化的蚁群算法，「四川大学学报(工程科学版)，2004 年 11 月」，第 36 卷第 6 期。

[3] 鄭克文，「應用高效率螞蟻演算法於多期動態資產配置最佳化的研究，暨南大學資訊管理研究所」，碩士論文，2007 年。

英文文獻

[4] Dorigo, M., Maniezzo, V., and Coloni, A., “The ant system: an autocatalytic optimizing process,” *Technical Report* no. 91-016, Politecnico di Milano, Italy 1991.

[5] Marco Dorigo, “Ant System: Optimization by a Colony of Cooperating Agents”, *IEEE Transactions on Systems, Man, and Cybernetics--Part B* , vol. 26, No. 2, pp. 29—41, 1996.

[6] M. Dorigo and L.M. Gambardella, “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem”, *IEEE Transactions on Evolutionary Computation*, 1(1):53-- 66, 1997

[7] Dorigo, M., Caro G. D. and Gambardella, L. M., “Ant System for Discrete Optimization”, Artificial Live, Vol.15, No.3, 137-172, 1999.

[8] Stutzle, Thomas, and Holger Hoos. “MAX-MIN ant system and local search for the traveling salesman problem.” Evolutionary Computation, 1997., IEEE International Conference on. IEEE, 1997.

[9] Kern, Stefan, et al. “Learning probability distributions in continuous evolutionary algorithms–a comparative review.” Natural Computing 3.1 (2004): 77-112.

[10] Socha, Krzysztof, and Marco Dorigo. “Ant colony optimization for continuous domains.” European Journal of Operational Research 185.3 (2008): 1155-1173.

[11] Toksari, M. Duran. “Ant colony optimization for finding the global minimum.” Applied Mathematics and Computation 176.1 (2006): 308-316.

[12] Duran Toksarı, M. “A heuristic approach to find the global optimum of function.” Journal of Computational and Applied Mathematics 209.2 (2007): 160-166.

[13] Baskan, Ozgur, et al. “A new solution algorithm for improving performance of ant colony optimization.” Applied Mathematics and Computation 211.1 (2009): 75-84.

[14] Seçkiner, Serap Ulusam, et al. “Ant colony optimization for continuous functions by using novel pheromone updating.” Applied Mathematics and Computation 219.9 (2013): 4163-4175.

[15] Alam, Mohammad Shafiul, et al. “Diversity Guided Evolutionary Programming: A novel approach for continuous optimization.” Applied soft computing 12.6 (2012): 1693-1707.

[16] CUDA : Computer Unified Device Architecture Programming Guide 3.1 (2010).

[17] CUDA : Computer Unified Device Architecture Programming Guide PG-02829-001_v5.0 (2012).

參考網站

[18] http://iridia.ulb.ac.be/~mdorigo/ACO/index.html

[19] http://newsroom.intel.com/docs/DOC-2383

[20] http://www.nvidia.com.tw/object/what-is-gpu-computing-tw.html

[21] http://www.nvidia.cn/object/corporate-timeline-cn.html

[22] http://sci2s.ugr.es/EAMHCO/

[23] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html

附錄

附表 1 Seçkiner(2013)測試的 benchmark

Formulation	Range	min	*m,k,I*
$f_1(x)=\begin{cases}x^2 & if\ x\le 1\\ (x-3)^2-3 & if\ x>1\end{cases}$	[0,10]	-3 x = 3	4,10,100
$f_2(x)=\left[x\times\sin\left(\frac{1}{x}\right)\right]^4+\left[x\times\cos\left(\frac{1}{x}\right)\right]^4$	[-1,1]	0 x = 0	3,10,500
$f_3(x,y)=\frac{(x-3)^8}{1+(x-3)^8}+\frac{(y-3)^4}{1+(y-3)^4}$	[-10,10]	0 x = y = 3	3,10,500
$f_4(x,y)=\left(100\times\left(x-y^2\right)^2\right)+(1-x)^2$	[-5,5]	0 x = y = 1	40,40,500
$f_5(x,y)=\frac{x}{1+\lvert y\rvert}$	[-10,10]	-10 x = -10 y = 0	5,100,1125
$f_6(x,y)=x^2+2y^2-0.3\cos(3\pi x)$ $-0.4\cos(4\pi y)+0.7$	[-10,10]	0 x = y = 0	2,10,500
$f_7(x)=\sum_{i=1}^{n}-x_i\times\sin\sqrt{\lvert x_i\rvert}$	[-500,500]	-837.9858 x=420.9687	5,10,150
$f_8(x,y)=x^2+y^2-\cos(18x)-\cos(18y)$	[0,10]	-2 x = y = 0	3,100,75
$f_9(x,y)=\exp\left(\frac{1}{2}\left(x^2+y^2-25\right)^2\right)+$ $\sin^4(4x-3y)+\frac{1}{2}(2x+y-10)^2$	[0,4]	1 x = 3 y = 4	5,150,500
$f_{10}(x,y)=\left[4-2.1x^2+\frac{x^4}{3}\right]x^2+xy+\left[-4+4y^2\right]y^2$	x[-3,3] y[-2,2]	-1.0316 x = -0.0898 y = 0.7125	5,10,100

附表 2 Yao(2012)使用的 benchmark(f_1-f_{13})

Formulation	Range	D	f_{min}
$f_1(x)=\sum_{i=1}^{n}x_i^2$	[-100,100]	30	0
$f_2=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	[-10,10]	30	0
$f_3=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_i\right)^2$	[-100,100]	30	0
$f_4(x)=\max_i\{\lvert x_i\rvert,1\le i<n\}$	[-100,100]	30	0
$f_5(x)=\sum_{i=1}^{n-1}\left(100\left(x_i^2-x_{i+1}\right)^2+\left(x_i-1\right)^2\right)$	[-30,30]	30	0
$f_6(x)=\sum_{i=1}^{n}\left(\lfloor x_i+0.5\rfloor\right)^2$	[-100,100]	30	0
$f_7(x)=\sum_{i=1}^{n}ix_i^4+random[0,1)$	[-1.28,1.28]	30	0
$f_8(x)=\sum_{i=1}^{n-1}-x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	[-500,500]	30	-1256 9.5
$f_9(x)=\sum_{i=1}^{n}\left(x_i^2-10\cos(2\pi x_i)+10\right)$	[-5.12,5.12]	30	0
$f_{10}(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+e$	[-32,32]	30	0
$f_{11}(x)=\frac{1}{4000}\left(\sum_{i=1}^{n}x_i^2\right)-\left(\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right)+1$	[-600,600]	30	0
$f_{12}(x)=\frac{\pi}{n}\left(10\sin^2(\pi y_1)+\left(y_n-1\right)^2\right)+\sum_{i=1}^{n}u(x_i,10,100,4)+\frac{\pi}{n}\left[\sum_{i=1}^{n-1}\left(y_i-1\right)^2\left(1+10\sin^2(\pi y_{i+1})\right)\right]$	[-50,50]	30	0
$f_{13}(x)=0.1\left(\sin^2(\pi x_1)+\left(x_n-1\right)^2\left(1+\sin^2(2\pi x_n)\right)\right)+\sum_{i=1}^{n}u(x_i,5,100,4)+0.1\left[\sum_{i=1}^{n-1}\left(x_i-1\right)^2\left(1+\sin^2(3\pi x_{i+1})\right)\right]$	[-50,50]	30	0

附表 3 Yao(2012)使用的 benchmark(f_{14}-f_{23})

Formulation	Range	D	f_{min}
$f_{14}(x)=\left[\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}\right]^{-1}$	[-65.536, 65.536]	2	1
$f_{15}(x)=\sum_{i=1}^{11}\left(a_i-\frac{x_1(b_i^2+b_ix_2)}{b_i^2+b_ix_3x_4}\right)^2$	[-5,5]	4	0.0003075
$f_{16}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	[-5,5]	2	-1.0316285
$f_{17}(x)=\left(x_2-\frac{5.1}{4\pi^2}+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x_1)+10$	[-5, 15]	2	0.398
$f_{18}(x)=\left[1+(x_1+x_2+1)^2\times(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right]\times\left[30+(2x_1-3x_2)^2\times18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right]$	[-2,2]	2	3
$f_{19}(x)=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2\right]$	[0,1]	3	-3.86
$f_{20}(x)=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{6}a_{ij}(x_j-p_{ij})^2\right]$	[0,1]	6	-3.32
$f_{21}(x)=-\sum_{i=1}^{5}\left[(x-a_i)(x-a_i)^T+c_i\right]^{-1}$	[0,10]	4	-10.1532
$f_{22}(x)=-\sum_{i=1}^{7}\left[(x-a_i)(x-a_i)^T+c_i\right]^{-1}$	[0,10]	4	-10.4029
$f_{23}(x)=-\sum_{i=1}^{10}\left[(x-a_i)(x-a_i)^T+c_i\right]^{-1}$	[0,10]	4	-10.5364

附表 4 CEC'2008 的 6 個 benchmark

Formulation	Range	f_bias	f_{min}
$F_1(x)=\sum_{i=1}^{D} z_i^2 + f\_bias_1$	$[-100,100]$	-450	-450
$F_2(x)=\max_i\{\lvert z_i\rvert, 1\le i\le D\}+ f\_bias_2$	$[-100,100]$	-450	-450
$F_3(x)=\sum_{i=1}^{D-1}\left(100\left(z_i^2 - z_{i+1}\right)^2 + \left(z_i - 1\right)^2\right)+ f\_bias_3$	$[-100,100]$	390	390
$F_4(x)=\sum_{i=1}^{D}\left(z_i^2 - 10\cos(2\pi z_i)+10\right)+ f\_bias_4$	$[-5,5]$	-330	-330
$F_5(x)=\sum_{i=1}^{D}\frac{z_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{z_i}{\sqrt{i}}\right)+1+ f\_bias_5$	$[-600,600]$	-180	-180
$F_6(x)=-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right)-$ $\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right)+20+e+ f\_bias_6$	$[-32,32]$	-140	-140

附表 5 Kern_2004 比較的函式

Function	Range	Dimensions	Formulation
Plane (PL)	[0.5,1.5]	10	$F_{PL}(x)=x_1$
(DP)	[0.5,1.5]	10	$F_{DP}(x)=\frac{1}{n}\sum_{i=1}^{n}x_i$
(SP)	[-3,7]	10	$F_{SP}(x)=\sum_{i=1}^{n}x_i^2$
Ellipsoid (EL)	[-3,7]	10	$F_{EL}(x)=\sum_{i=1}^{n}\left(100^{\frac{i-1}{n-1}}x_i\right)^2$
Cigar (CG)	[-3,7]	10	$F_{CG}(x)=x_1^2+10^4\sum_{i=2}^{n}x_i^2$
(TB)	[-3,7]	10	$F_{TB}(x)=10^4x_1^2+\sum_{i=2}^{n}x_i^2$

PL－Plane

DP－Diagonal plane

SP－Sphere

EL－Ellipsoid

CG－Cigar

附表 6 相關螞蟻演算法使用函式(一)

Formulation	Range	D	f_{min}
$F_{RC}(x)=\left(x_2-\frac{5}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_i+10$	[-5,15]	2	0.398
$F_{B2}(x)=x_1^2+2x_2^2-\frac{3}{10}\cos(3\pi x_1)-\frac{2}{5}\cos(4\pi x_2)+\frac{7}{10}$	[-100,100]	2	0
$F_{ES}(x)=-\cos(x_1)\cos(x_2)e^{-((-x_1-\pi)^2+(x_2-\pi)^2)}$	[-100,100]	2	-1
$F_{GP}(x)=\left[1+(x_1+x_2+1)^2\times(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right]\times\left[30+(2x_1-3x_2)^2\times18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right]$	[-2,2]	2	3
$F_{MG}(x)=(x_1-x_2)^2+\left(\frac{x_1+x_2-10}{3}\right)^2$	[-20,20]	2	0
$F_{Rn}(x)=\sum_{i=1}^{n-1}100(x_i^2-x_{x+1})^2+(x_i-1)^2$	[-5,10]	2	0
$F_{Zn}(x)=\left(\sum_{j=1}^{n}x_j^2\right)+\left(\sum_{j=1}^{n}\frac{jx_j}{2}\right)^2+\left(\sum_{j=1}^{n}\frac{jx_j}{2}\right)^4$	[-5,10]	2,5	0
$F_{DJ}(x)=x_1^2+x_2^2+x_3^2$	[-5.12,5.12]	3	0
$F_{GR_n}(x)=\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	[-5.12,5.12]	10	0
$F_{SM}(x)=\sum_{i=1}^{n}x_i^2$	[-5.12,5.12]	6	0

RC－Branin RCOS

ES－Easom

GP－Goldstein and Price

MG－Martin and Gaddy

Rn－Rosenbrock

Zn－Zakharov

DJ－De Jong

GRn－Griewangk

SM－Sphere model

附表 7 相關螞蟻演算法使用函式(二)

Formulation	Range	D	f_{min}
$f_{H3,4}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}\left(x_j - p_{ij}\right)^2\right]$	[0,1]	3	-3.86
$f_{H6,4}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right]$	[0,1]	6	-3.32
$f_{S4,5}(x) = -\sum_{i=1}^{5} \left[(x-a_i)(x-a_i)^T + c_i\right]^{-1}$	[0,10]	4	-10.1532
$f_{S4,7}(x) = -\sum_{i=1}^{7} \left[(x-a_i)(x-a_i)^T + c_i\right]^{-1}$	[0,10]	4	-10.4029
$f_{S4,10}(x) = -\sum_{i=1}^{10} \left[(x-a_i)(x-a_i)^T + c_i\right]^{-1}$	[0,10]	4	-10.5364

H－Hartmann

S－Shekel

附表 8 各實驗參數設定

	Yao_2008	ACO_NPU	Yao_2012	CEC2008
螞蟻數	2	每個問題與論文設定相同	前 13：30 後 10：10	50
archiev_size	50		前 13：30 後 10：10	50
迭代次數	視問題而定		前 13：4999 後 10：999	99999
學習率	0.0001	0.01	0.01	0.01

下載與安裝

步驟 1： 在 https://developer.nvidia.com/cuda-downloads 下載 CUDA Toolkit，選擇適用的作業系統及型號與類型，主要分成 Windows、Linux、Mac OS X。

步驟 2： 最新的 CUDA Toolkit 版本為：CUDA Toolkit 5.0，裡面包含了 CUDA SDK 以及許多的範例檔和函示庫。而要編譯過範例檔需要安裝具有 C/C++的編譯器，而在 windows 下，目前支援的有 VS 2003、VS 2005、VS 2008、VS 2010 等開發環境。

步驟 3： 接下來在 Visual Studio 中使用 CUDA，以 VS 2008 為例：

一、 建立一個 Win32 Console 的空專案，在專案上按右鍵→自訂建置規則，點選...\NvCudaRuntimeApi.v5.0.rules，按下確定。

二、 在工具列上點選工具→選項→專案和方案→VC++目錄

(1)在[可執行檔]拉到最下面，右邊新增(toolkit 的 bin)

(2)在[Include 檔案]拉到最下面，右邊新增(toolkit 的 include)

(3)在[程式庫檔]拉到最下面，右邊新增(toolkit 的 lib 裡面的 Win32)

三、 新增一個.cpp 檔案 檔名 EX:hello.cu

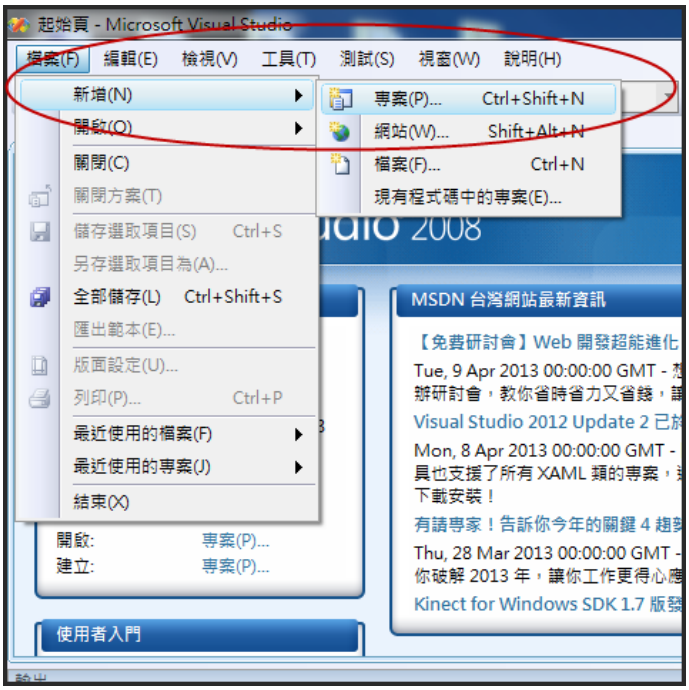
四、 在專案上按右鍵→屬性→組態屬性→連結器

(1) 輸入，[其他相依性]加入 cuda.lib cudart.lib

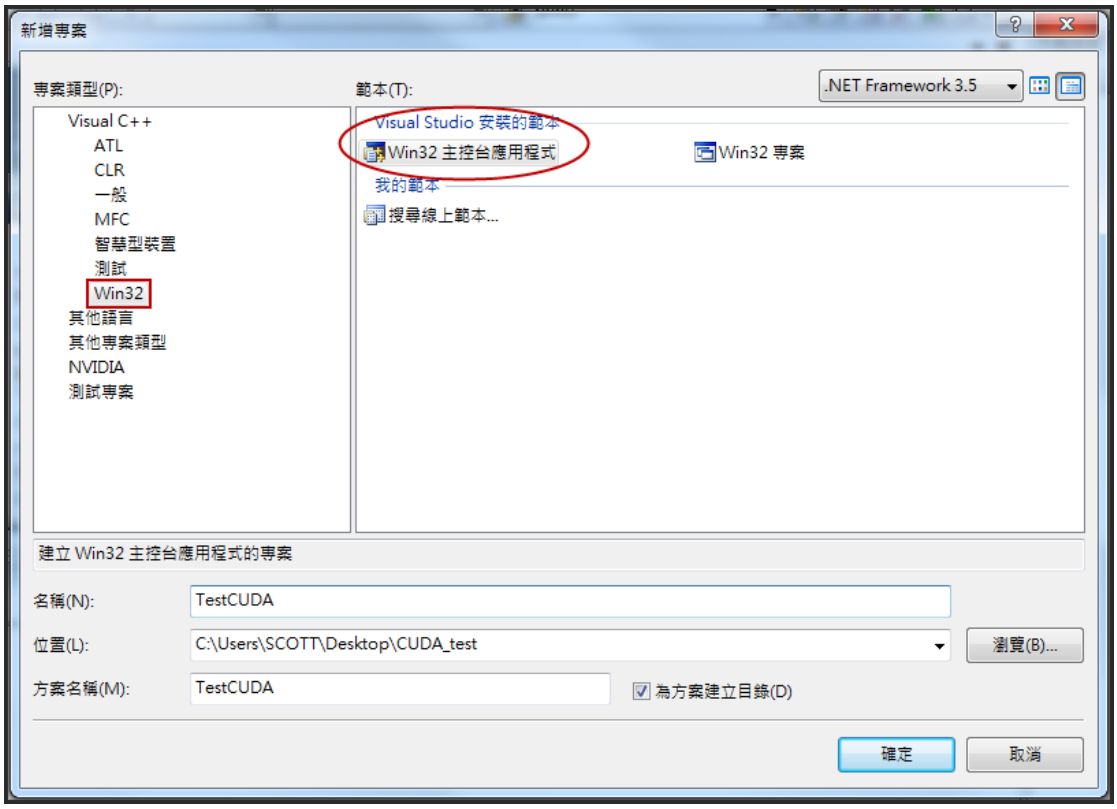
(2) 一般，[其他程式庫目錄]，右邊新增，再右邊新增(tookit 的 lib 裡面的 Win32)

(3) 確定

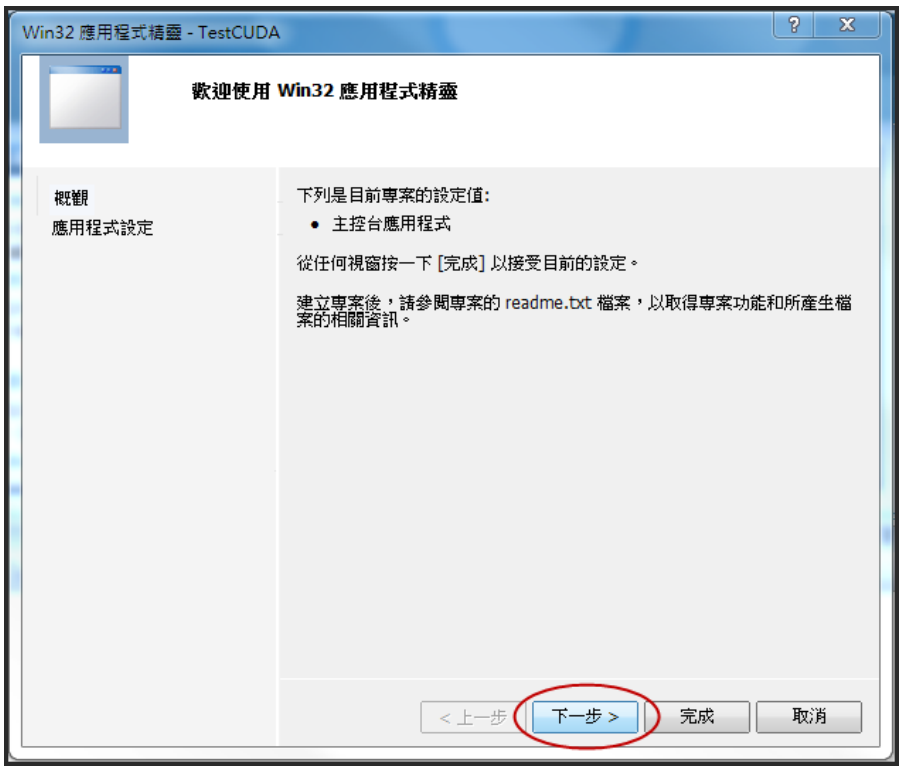
步驟 4： 在 Visual Studio 的開發環境輸入 CUDA 的程式碼，之後直接編譯執行即可。



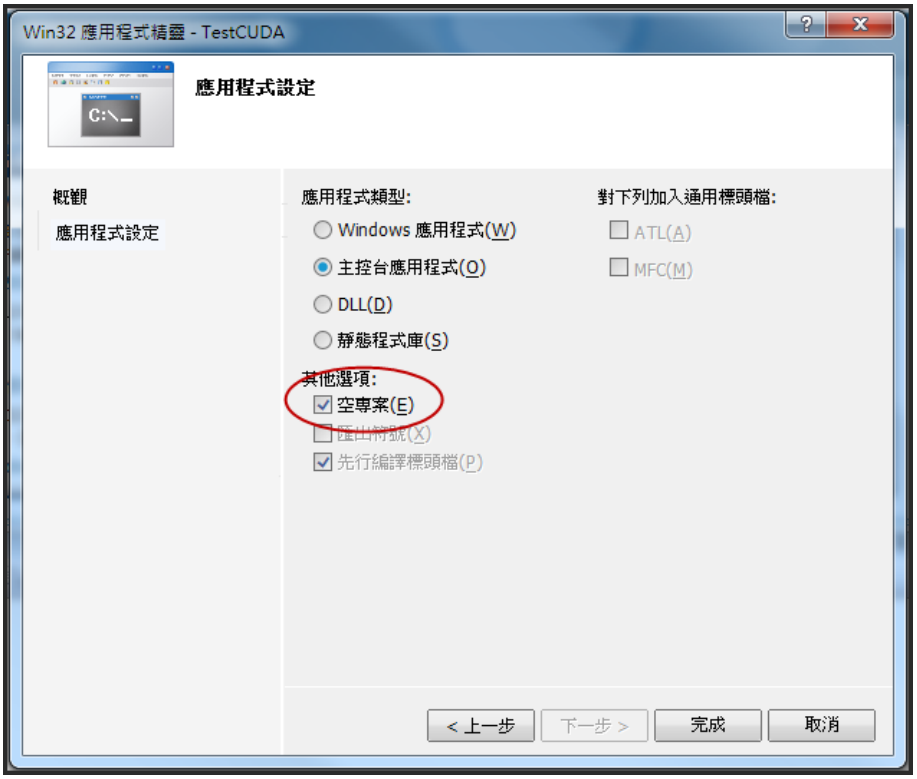
附圖 1 新增專案



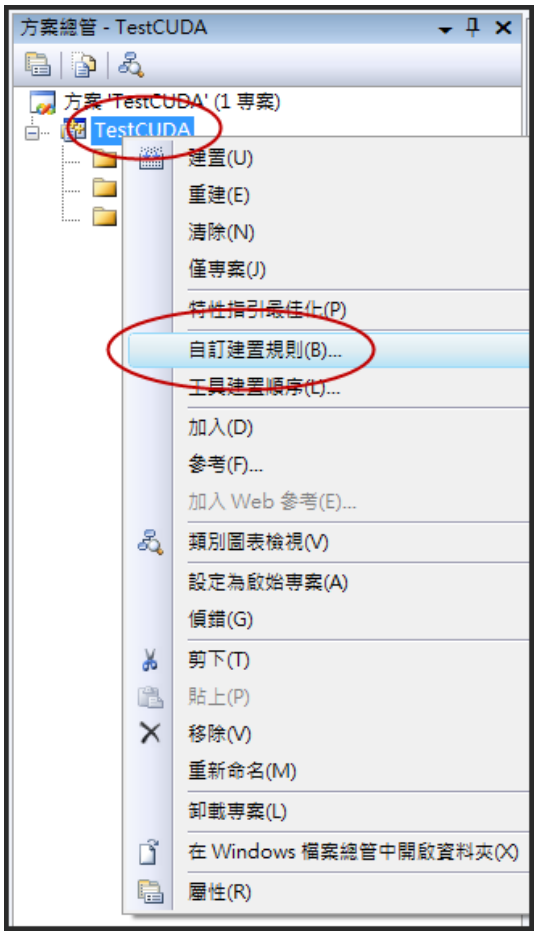
附圖 2 新增 Win32 主控台應用程式



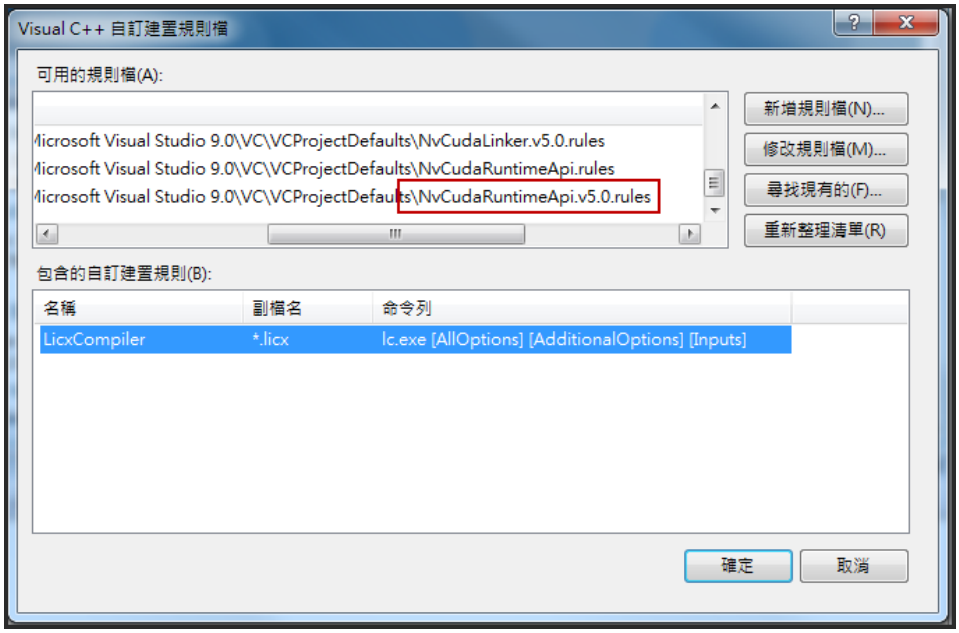
附圖 3 使用程式精靈



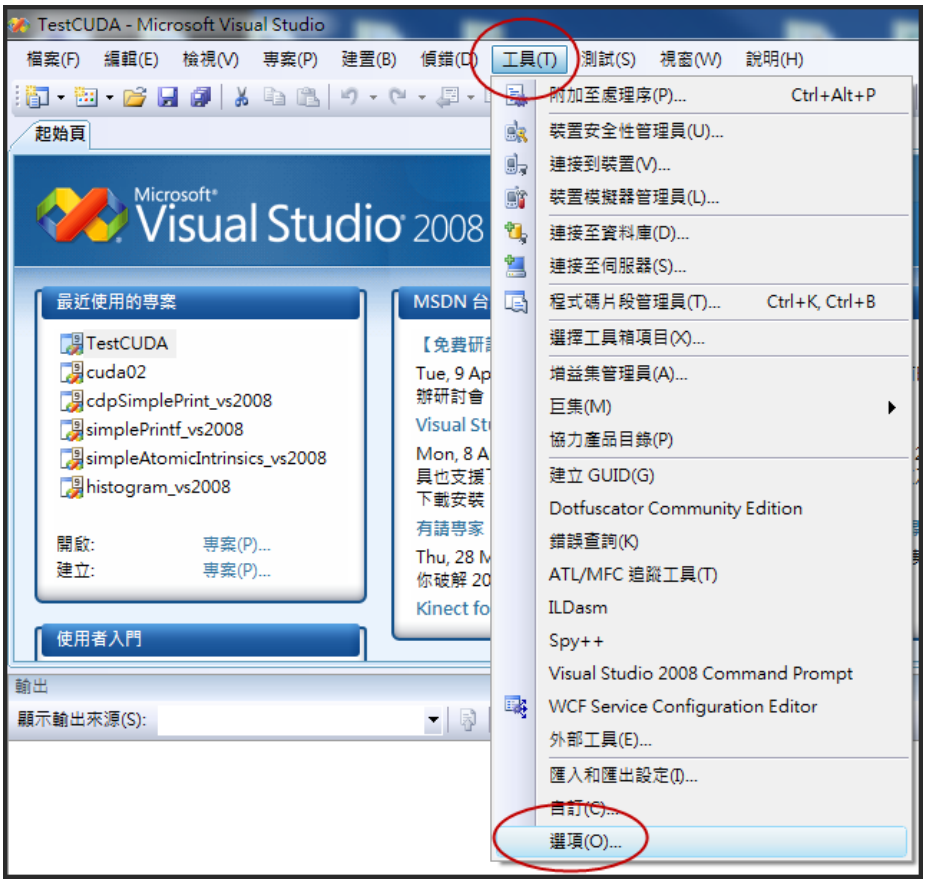
附圖 4 新增空專案



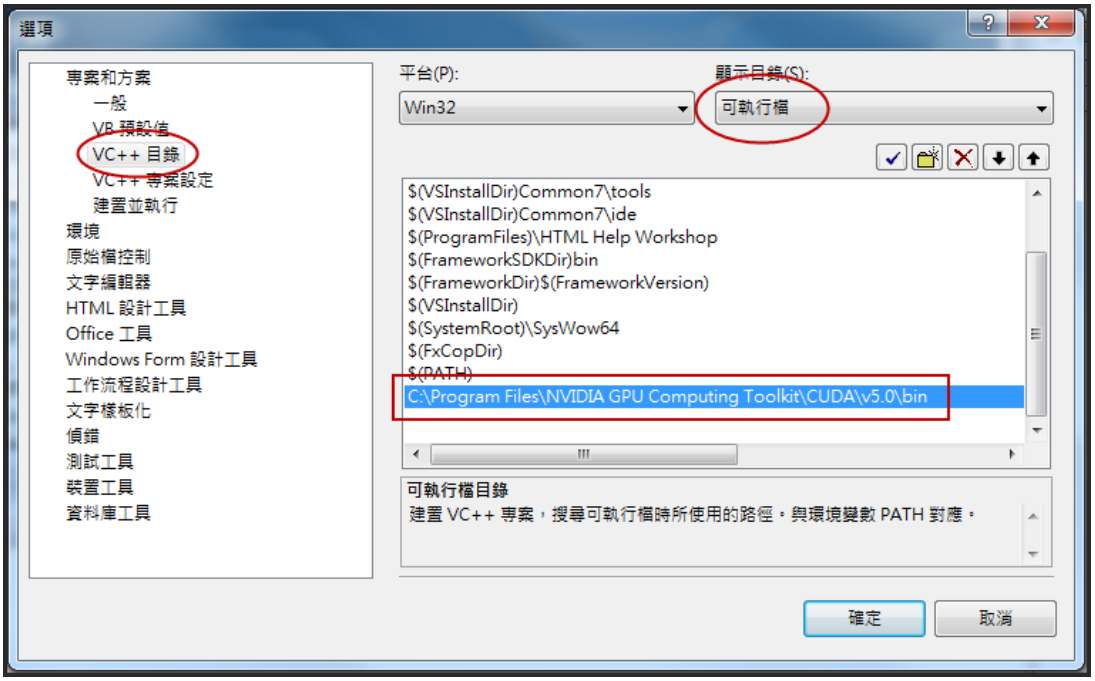
附圖 5 修改專案建置規則



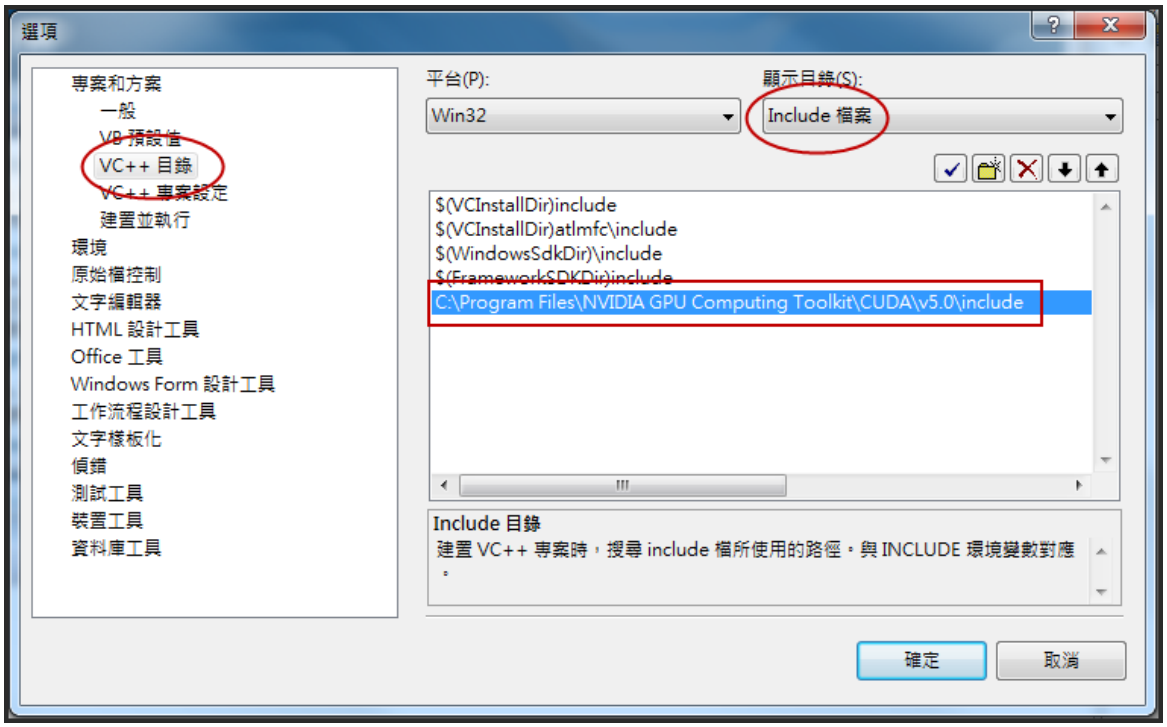
附圖 6 加入規則檔.rules



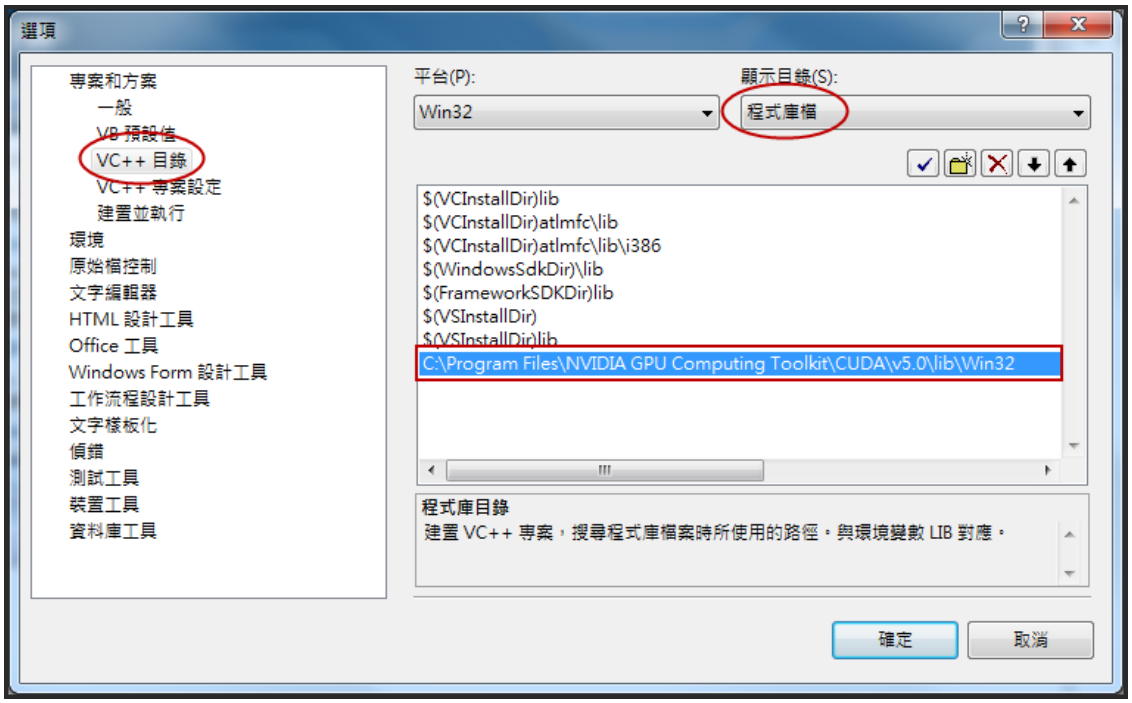
附圖 7 修改專案程式庫



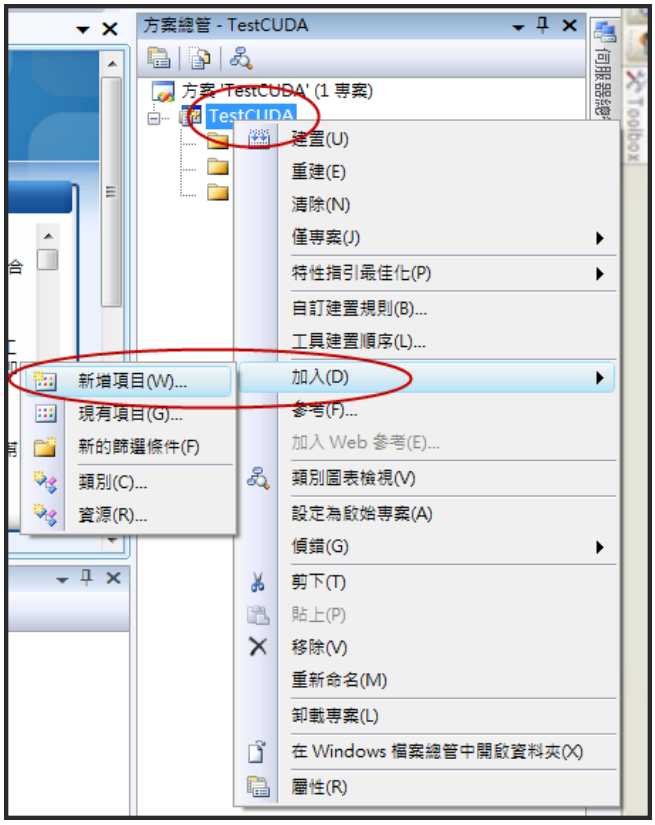
附圖 8 加入[可執行檔]



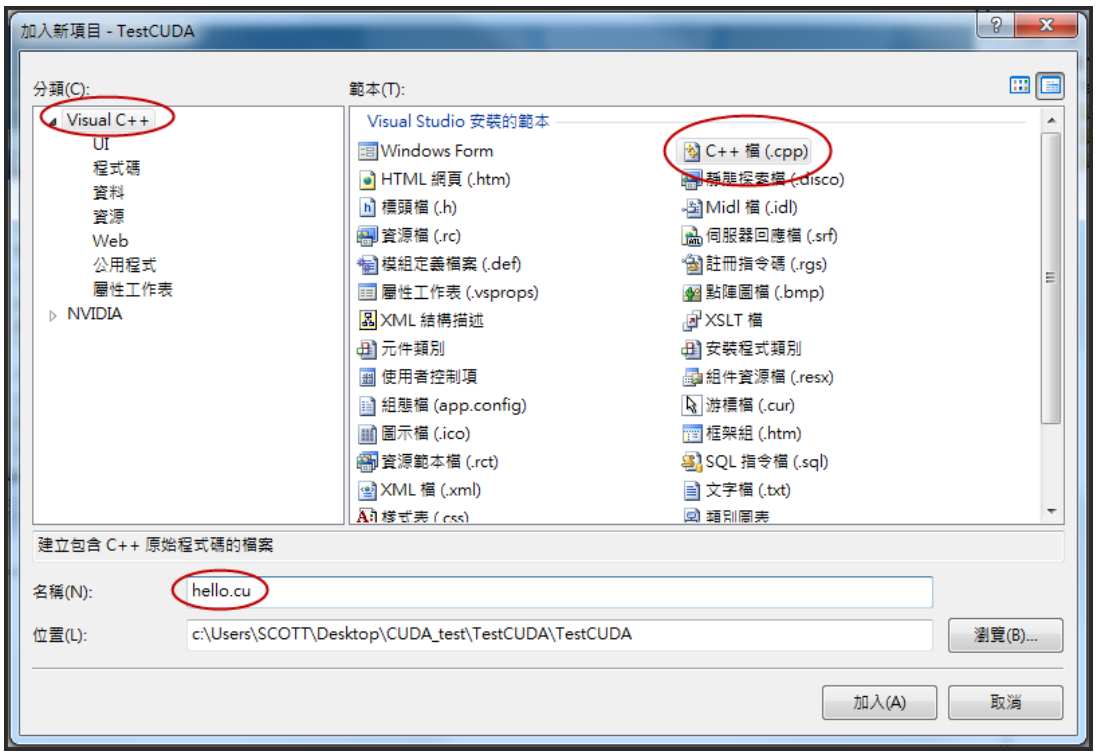
附圖 9 加入[Include 檔案]



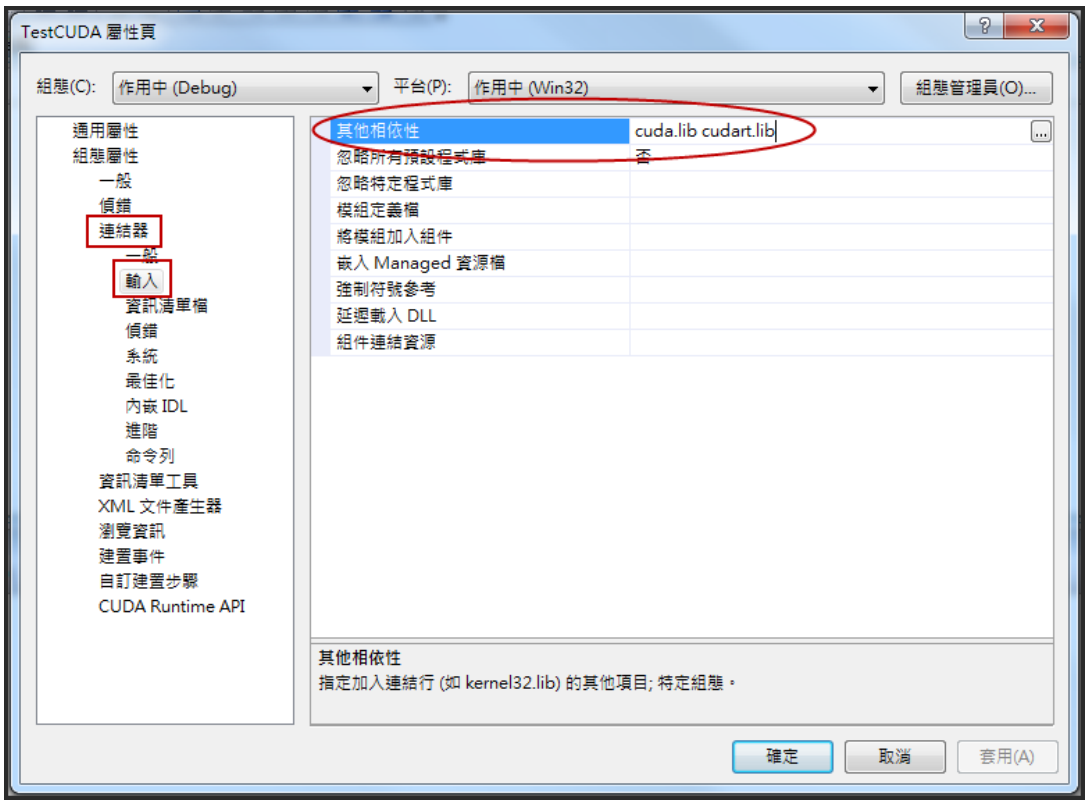
附圖 10 加入[程式庫檔]



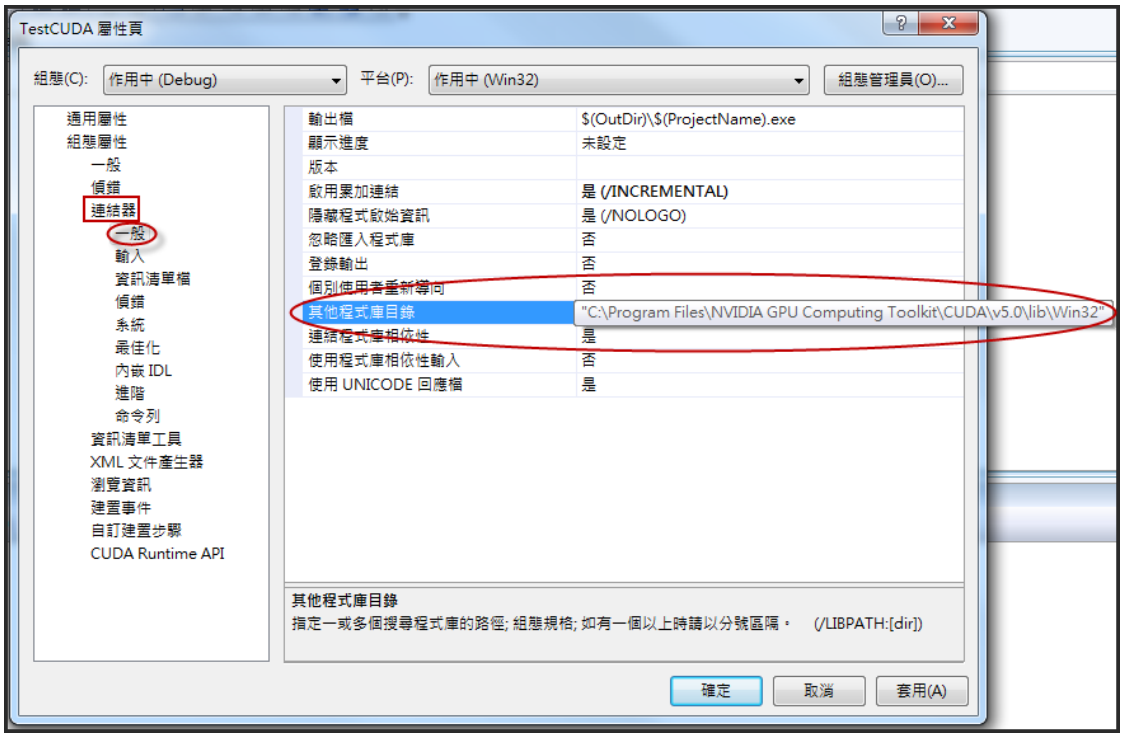
附圖 11 新增檔案



附圖 12 新增.cpp 檔



附圖 13 加入函式庫



附圖 14 加入函式庫