使用SARSA(λ)決定PSO和CPSO-S最佳的切換時機達成函數值最佳化

研究生：劉鎮榮　　指導教授：林昇甫　博士

國立交通大學電控工程研究所

摘　要

本篇論文提出一個演算法，使用 SARSA(λ)來決定基本粒子群最佳化及協同式粒子群最佳化的使用時機。經測試函數的極值搜尋結果得知，由於本論文所提出演算法結合了基本粒子群與協同粒子群最佳化，在所挑選的五個測試函數中維度間不論是否關聯以及測試函數具有單峰極值或是多峰極值的情況，在執行相同的時間之下的最佳值搜尋結果比起基本的粒子群最佳化、以及將原本的粒子群分割成維度更小的子群的協同式粒子群最佳化、基本粒子群最佳化和協同粒子群最佳化交替作用的混合式粒子群最佳化等這三種演算法的結果都能夠有最佳或是次佳的表現。即代表本論文在演算法處理各種函數最佳值搜尋的問題上比起基本粒子群最佳化和協同式粒子群最佳化僅僅處理特定函數問題上要來的好，而在能夠處理不同函數的問題上又能夠比混合式粒子群最佳化需要更少的搜尋時間此為本論文的貢獻。

Using SARSA(λ) to Find the Best Switching Policy Between PSO and CPSO-S to Achieve Optimal Function Values

Student: Zhen-Rong Liu Advisor: Dr. Sheng-Fuu Lin

Institute of Electrical Control Engineering

National Chiao Tung University

Abstract

This paper proposed an algorithm, using reinforcement learning mechanism to determine the best method of particle swarm optimization and cooperative particle swarm optimization. The test function extremum search showed that, the proposed algorithm in this paper combining with particle swarm optimization and cooperative particle swarm optimization can deal with a function with or without dimensional correlation and unimodal or multimodal, in the same execution time proposed algorithm in this paper compared to the basic particle swarm optimization, cooperative particle swarm optimization, hybrid particle swarm optimization. The results show that the proposed algorithm is able to have the best or second performance of other three algorithms

誌謝

這篇論文能夠完成實在要感謝很多人，若沒有這些貴人的支持與幫助相信是沒有這一篇論文的產生。

這篇論文能夠完成首先要感謝我的指導教授-林昇甫老師，感謝老師的指導讓學生學到了很多做事做學問的態度，也感謝老師在最後的口試以及論文修訂上的費心幫忙，也讓我體會弟子事師，敬同于父，習其道也，學其言語一日為師，終生為父。

家人也是我要感謝的人，我要十分謝謝我的媽媽我讀研究所的日子裡不但要做我的經濟上的支柱，還要為我的學業操心，沒有媽媽的支持我一定無法完成碩士的學業，另外我也要感謝我的妹妹，多謝她的幫忙。

碩士論文的完善需要口試委員的監督與指導，感謝我的口試委員-蘇建焜教授與徐永吉，感謝您們撥空遠道而來，也感謝您們指導學生口試，有了您們的指導學生的碩士論文才能更臻完備。

研究的路上我還要感謝研究室的學長、學弟們，感謝你們的陪伴與幫忙，尤其是博班學長逸章，感謝他的指點讓我不在毫無頭緒，感謝家昌學弟三更半夜還出外替實驗室的大家買消夜，還有俊良及裕筆，感謝你們的協助與幫忙。

在此，僅將此論文獻給我最愛的媽媽、妹妹、師長、學長、學弟妹，願與大家分享這成果。

劉鎮榮

一百零一年　三月　九日

目錄

圖目錄

表目錄

第一章

緒論

在真實世界中許多問題皆與最佳化有關，通常皆希望能透過對這些問題進行分析並用函數來描述,再對其求取最佳解來處理。一般說來這些問題常常是複雜的，非線性的，非凸的(非凹的)，不連續的，甚至難以用函數來描述使得傳統的最快下降法或是牛頓法難以有效甚至無法去求取最佳解。因此有許多最佳化理論被提出來例如數學規劃(mathematical programming)、最佳化理論(theory of optimization)這些學門都在探討如何找尋最佳解，還有一些啟發於生物上特性所發展出的方法如:基因演算法、粒子群演算法，都被作為求取最佳解問題的工具。本論文將在第一章的 1.1 節介紹相關研究，在 1.2 節中介紹研究動機，最後再 1.3 節介紹本論文的架構。

1.1 相關研究

近來,由於強大的計算機平台使得在 1950 年代由 Bremermann、Friedberg 等人所提出的演化式演算法得以實現，演化式演算法主要有三類:(1)EP(演化式規劃)，(2)ES(演化策略)，(3)GA(基因演算法)這些方法接根據達爾文進化論的觀念來運作即物競天擇，適者生存不適者淘汰。表 1.1 將這三類方法的異同整理如下。

表 1.1 演化式演算法的種類

	個體表示法	挑選機制	突變	重組
演化策略	實數向量	隨機挑選	重要的遺傳運算	次要的運算
演化規劃	實數向量	個體都會被挑選到	最重要的遺傳運算	沒有
基因演算法	位元實數向量	使用輪盤式選擇	次要的運算使用機率很低	主要的運算，為交配，有單點雙點、多點，等作法

由物競天擇的這個想法有了 Particle Swarm Optimization(PSO)、Cooperative Particle Swarm Optimization(CPSO)、Cooperative Genetic Algorithm(CGA)、Covariance Matrix Adaption Evolution Strategy(CMA-ES)等研究的產生，其中 J. Kennedy， R. C. Eberhart (1995)所提出的[1]PSO 在解空間搜尋最佳解有不錯的性質。但是當問題本身的變數過於龐大之時(curse of dimensionality)PSO 找到全域最佳解的機會將會以變數維度增加量指數性的降低，故有了協同演化(CPSO)研究的產生。CPSO 是由 Frans van den Bergh 和 A. P. Engelbrecht 在 2004 年所提出的[2]，提出這個想法主要是由於 Potter 提出的[3][4] CCGA(cooperative co-evolutionary genetic algorithm)發現當最佳化問題的維度過高時使用維度切割的方法將問題維度降下來後再進行演化可以使得效果大大的提高，而被引用至基本粒子群最佳化演算法中而形成。他的優點是由於引入了子群體(sub-swarm)的想法，使得粒子群中每個不同的粒子之間屬於相同維度的部分形成子群體，藉由子群體間的互相協調，共同演化，彌補了基本粒子群最佳化中由於缺乏個體間訊息交流所出現的問題。

機器學習主要為三大種類:監督式學習(supervise learning)，非監督式學習(unsupervised learning)，增強式學習(reinforcement learning)當中的一種，主要是透過代理人(agent)所做出的決策，並與環境(enviroment)互動取得激勵(reward)的訊號，代理人藉由找尋最佳的決策來使得激勵(reward)最大化，處理增強式學習的方法中TD(temporal difference)是最常見的一種方法[5]-[7]，他的好處是每得到依次激勵訊號，代理人便可以藉由此方法立即做學習，而不必等到整個訓練過程的結束，對一些訓練過程非常長的問題因為能夠即時的更新，而Q-learning，SARSA(state action reward state action)[8]即根據TD這種想法來學習不同狀態下不同決策的好壞。

1.2 研究動機

由於協同式粒子群最佳化會在處理一些欺騙函數的問題時出現陷入假的極小值(pseudominimizer)因此必須結合基本的粒子群最佳化演算法來解決這個問題。卻又因為無法得知何時陷入了這種情形使得必須利用協同式粒子群最佳化與基本粒子群最佳化交互作用來解決這個問題。另一方面因為協同式粒子群最佳化是將維度分割來增加粒子族群的多樣性，這樣一來演算法在判斷適應值優劣時會隨著問題維度的上昇以問題維度的倍數提高函數值的評估次數，這將嚴重的減緩演算法的效率，因此本論文希望能夠透過增強式學習法(reinforcement learning)中的 SARSA(λ)來學習協同式粒子群演算法和基本粒子群演算法切換的時機，省去不必要的協同式粒子群演算法的執行，藉此提高演算法效率，又能夠避免掉協同式粒子群演算法以及基本粒子群演算法各自會產生的缺點。表 1.2 整理了協同式粒子群最佳化以及基本粒子群最佳化的優點與缺點。

表 1.2 基本粒子群最佳化與協同式粒子群最佳化之優缺點

	優點	缺點
基本粒子群最佳化	1. 函數適應值評估次數較少 2. 沒有落入假的最佳解問題	1.粒子群種類較少，且個體間不能傳遞訊息 2.有兩步向前，一步退後的問題
協同式粒子群最佳化	1. 粒子群種類較多且個體間能夠交換訊息 2. 沒有兩步向前，一步退後的問題	1. 函數適應值評估次數較多 2. 有落入假的最佳解問題

1.3 論文架構

本篇論文剩餘章節的安排如下：論文使用到的一些相關知識與理論會在第二章作簡要的描述，我們所提出的演算法會在第三章提出並且作詳細的說明，實驗結果及分析會詳列在第四章，最後第五章會介紹結論與未來發展。

第二章
相關知識及理論

本章將對本論文提出的演算法所牽涉到的背景知識做簡單扼要的說明與介紹，本論文所提出的演算法是利用 SARSA(λ)來找出決定基本的粒子群演算法與協同式粒子群演算法切換的最佳時機。

本章將在 2.1 節介紹粒子群最佳化;在 2.2 節中將介紹協同式粒子群演算法;最後在 2.3 節將介紹增強式學習法。

2.1 粒子族群最佳化

粒子族群最佳化(particle swarm optimization)是一種基於群體的演化算法，他是由 R. C. Eberhart 和 J. Kennedy [1]於 1995 所發表的一種方法，這種方法主要是受啟發於觀察及研究鳥群覓食過程中的行為得來。鳥群在找尋食物的過程中，每一個個體雖然不一定知道食物在何處，但是透過身邊同伴飛行方向的牽引便大大提高了搜尋食物的效率跟機會，這種透過群體合作表現出的行為被稱之為群體智能。在求解最佳化問題，神經網路訓練，函數極值，模糊控制系統方面都被廣泛的應用，並獲得許多研究成果。

2.1.1 基本的粒子族群最佳化

在基本的粒子群最佳化演算法中，粒子群由粒子所組成每個粒子的位置即代表著最佳化問題的潛在的可能解，而粒子群依據自身當前位置與個體最佳解及整

個群體中最佳解這三項資訊來做為更新狀態之依據，如式(2.1), (2.2)所示:

$$v_{ij}(t+1) = v_{ij}(t) + v_{cognition} + v_{social} \tag{2.1}$$

$$v_{cognition} = c_1 rand\ [p_{ij}(t) - x_{ij}(t)] \tag{2.2}$$

$$v_{social} = c_2 rand\ [g_j(t) - x_{ij}(t)] \tag{2.3}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{2.4}$$

其中 $x_{ij}(t)$代表第 i 個粒子在第 j 個維度在時刻 t 的位置， $p_{ij}(t)$代表第 i 個粒子在第 j 個維度在時刻 t 之前最優的位置，g_j 代表整個粒子群在第 j 個維度在時刻 t 之前最優的位置。上面式(2.1)的含義為將每個個體視為 n 維空間中一個沒有體積的粒子(點),在搜索空間中以一定的速度飛行，而此速度根據他本身的飛行經驗以及同伴的飛行經驗進行動態調整而分為當前速度，個體的認知，以及社會影響三部分，其中個體的認知即對應了式(2.2)即每一個個體會對他在搜索過程中所經歷過一個最好的結果 p_{ij} 加以記錄下來並以這個認知做為其下一階段行動之依據，當中的 c_1 為個體的認知部分的權重， $rand$ 則代表在 $[0,1]$ 上均勻分布的模型所產生的隨機數，而式(2.3)則表示了社會影響即整個粒子群中每一個個體均會受到這個粒子群中表現最優良的個體的行為 g_j 所影響而增加這種行為發生的機會，當中的 c_2 為社會影響部分的權重， $rand$ 則代表在 $[0,1]$ 上均勻分布的模型所產生的隨機數，這種學習群體中不同個體的行為也被稱為社會行為。式(2.4)就是下一階段每個粒子群所在的位置。如果式(2.2)中的 c_1 為零則代表速度更新式(2.1)中的個體

認知部分沒有作用而只剩下社會影響，以及當前速度兩個部分在這種模型下粒子之間雖然可以相互作用但容易陷入局部極值點，而若式(2.3) 中的c_2為零則代表速度更新式(2.1)中的社會影響的部分沒有作用而只剩下個體認知，以及當前速度兩個部分在這種模型下粒子之間無法相互作用，相當於單獨的粒子進行重複整個粒子群個數的最佳位置的搜索，這使得找到最佳解的機率大大的減少，由上面分析知道，式(2.2)、式(2.3)即個體認知部分以及社會影響對粒子群最佳化演算法的重要性。

以下為粒子群最佳化演算法的流程:

Step 1:初始化粒子群的隨機位置(x_{ij})和速度(v_{ij})。

Step 2:評值粒子的適應值(fitness value)，適應值根據最佳化問題所給的函數而定。

Step 3:對每個粒子，將其當前適應值與其個體最佳位置p_{ij}比較，若當前適應值較佳則以當前粒子的位置更新p_{ij}。

Step 4:對每個粒子，將其當前適應值與整個粒子群中最佳位置g_j比較若當前適應值較佳則以當前粒子的位置更新粒子群最佳位置g_j。

Step 5:根據式(2.1)和式(2.4) 更新粒子的速度(v_{ij})以及位置(x_{ij})。

Step 6:檢查是否滿足停止條件,如未滿足則返回 step 2。

通常以是否達到最大疊代次數或是最佳適應值的改變量是否小於給定的閥值做為是否終止演算法之依據。

圖 2.1 為基本的粒子群演算法流程圖:

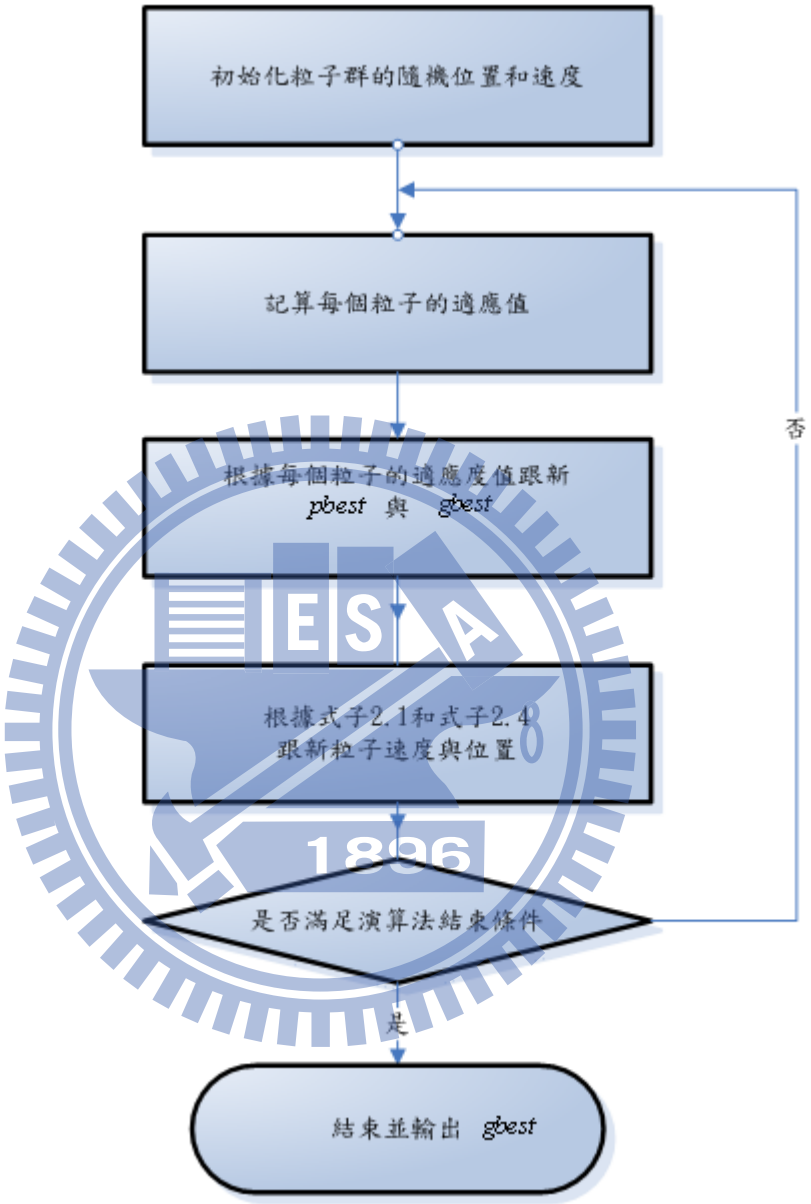


圖 2.1　基本粒子群演算法流程。

2.1.2 改良的粒子族群最佳化

改良的粒子群演算法是由 Y. Shi 和 R. C. Eberhart[9]所提出的，對式(2.1)中第

一部分即當前速度加入了慣性項 w,這一項的引進著眼於假設 w=0 則下一階段的速度僅取決於粒子當前位置 x_{ij} 、 p_{ij} 、 g_j 這三個變數，使得速度本身沒有了記憶性，假設有一個粒子位於 $gbest=[g_1 \quad g_2 \quad \cdots \quad g_n]$，則他將保持靜止,而其他粒子則飛向他本身最佳位置 $pbest=[p_1 \quad p_2 \quad \cdots \quad p_n]$ 與 $gbest=[g_1 \quad g_2 \quad \cdots \quad g_n]$ 的加權中心。在這種條件下，粒子僅能探索有限的區域，更像一種局部的算法，因此 w 慣性項的作用在賦予粒子擴展搜索全域的能力，即透過調整 w 的大小可以起到調整粒子群最佳化的全局跟局部搜索能力的作用，修正後的速度更新方式如式(2.5)所示:

$$v_{ij}(t+1)=wv_{ij}(t)+v_{cognition}+v_{social} \tag{2.5}$$

對於搜索空間較大的問題，為了在搜索速度以及搜索精度上達到較好的平衡，通常的做法是，使算法在前期有較大的全域搜尋能力以得到較佳的 *pbest* 、 *gbest* ，而在後期有較高的局部搜索能力以提高收斂的速度。為此可以使 w 隨著時間線性的減小，以上是對式(2.1)第一部分做的改良。之後 M. Clerc 和 J. Kenndy 又為粒子群最佳化同時考慮了慣性項、c_1、c_2 及簡化後的速度更新公式對收斂性的影響引入了收縮因子(constriction factor)，對演算法收斂性的影響來確保簡化後粒子群算法的收斂性[10][11]。改良過後的式(2.6)、(2.7)、(2.8):

$$v_{ij}(t+1)=\chi(wv_{ij}(t)+v_{cognition}+v_{social}) \tag{2.6}$$

$$x_{ij}(t+1)=x_{ij}(t)+v_{ij}(t+1) \tag{2.7}$$

$$\chi=\frac{2}{\left|2-\phi-\sqrt{\phi^2-4\phi}\right|} \quad ，其中 \quad \phi=c_1+c_2 \tag{2.8}$$

由上述研究可以知道粒子群最佳化的演算法的速度更新公式是此方法表現好壞的最核心環節，從速度更新公式 (2.1)、(2.5)、(2.6)來看，他們都摹擬了生物習慣的作用(慣性項)、個體認知能力(*pbest*)、社會行為(*gbest*)。以這三項的變化來引導粒子(個體)最佳化的過程，但是明顯的缺少了粒子之間的訊息共享、競爭與合作關係的摹擬。在一個簡單的例子當中可以容易的發現缺少了粒子之間訊息共享、競爭與合作會有什麼不良的影響考慮一個三維最佳化問題 $\min f(x) = (x-a)^2$ 其中 a 為(20, 20, 20)。明顯最佳解出現在 $x = a$ =(20, 20, 20)，另當前時間為 t，$y(t)$ 為當前最佳解 *gbest*，$x(t)$ 為粒子群中的一個粒子，在 $y(t+1)$ 與 $y(t)$ 相同的情況下 $x(t)$ 會朝著 *gbest* (即 $y(t)$)的方向移動。因此式(2.9)在這種情況下成立:

$$\|x(t+1) - y(t+1)\| < \|x(t) - y(t)\| \tag{2.9}$$

假設 $x(t)$ =(5, 20, 5)， $y(t)$ =(17, 2, 17)， $x(t+1)$ =(15, 5, 15)，因為 $f(x(t))$ =450，$f(y(t))$ =342， $f(x(t+1))$ =275，所以 $x(t+1)$ 的適應值比 $y(t)$ 還要優良，由粒子群最佳化規則知道 *gbest* 必然不再是 $y(t)$ ，但是觀察到 $x(t+1)$ 的第 2 個維度可以發現由 t 時間的 20 變為 $t+1$ 時間的 5，即使在適應值大幅的進步的情況下可以很容易的發現第 2 個維度卻由原本最佳的位置移動到了不好的位置，這種缺陷被 Frans van den Bergh 和 A. P. Engelbrecht[2]提出來，被稱為二步向前一步向後(two step forward，one step back)，而為了解決這種問題進而對基本粒子群最佳化演算法提出新的模型。這方法也就是協同式粒子群最佳化(cooperative particle swarm optimization)。

2.2 協同式粒子族群最佳化

在上一節提到了原始的粒子群最佳化(PSO)由於少了個體之間的合作競爭的機制導致了二步向前一步向後(two step forward one step back)的問題所以 Frans van den Bergh 和 A. P. Engelbrecht 提出了個體能夠互相競爭合作的方法，方法敘述如下:原始的粒子群按維劃分出子群，每個子群中個粒子都有自己當前個體的最佳位置 x_{ij} (i 是粒子在子群中的編號， j 則是子群的編號同時也是維度編號)，各子群又各擁有各自的當前全局最優位置 g_j ，所有子群中的最優位置之組合便成為當前整體最優位置 *gbest* 。個子群內的粒子的適應值計算是利用 context vector 將該粒子與其他子群內的 g_j 組合後帶入適應值函數 f 計算出來。由上述可知由於子群的劃分使得每一個維度(即每一個子群)內的每一個個體都能透過 g_j 來互相傳遞訊息與產生競爭也可以防止原始 PSO 所發生的二步向前一步向後的問題，不過卻要以未劃分子群前適應值計算複雜度的 n (原始粒子群維度)倍作為代價。如下所示即為最佳化問題粒子數為四，維度為四 CPSO-S 的做法:將每一個維度都視為依個子群更新速度及位置都以子群為單位，更新公式與式(2.6)，式(2.7)相同。

定義:*ccontext vector* $\equiv b(j,z) = (g_1, g_2, \cdots, g_{j-1}, z, g_{j+1}, \cdots, g_n)$
產生並初始化n個一維的子群每個子群$\bar{x}_j$均有s個粒子, $j \in [1..n], \bar{x}_j \in R^s$
重複以下
　對每個子群$j \in [1..n]$:
　　對每個子群中的粒子$i \in [1..s]$:
　　若$f(b(j,x_{ij})) < f(b(j,p_{ij}))$
　　則$p_{ij} = x_{ij}$
　　若$f(b(j,p_{ij})) < f(b(j,g_j))$
　　則$g_j = p_{ij}$
　　結束
　　對每個子群進行粒子群最佳化中位置及速度的更新
　結束
直到滿足終止條件

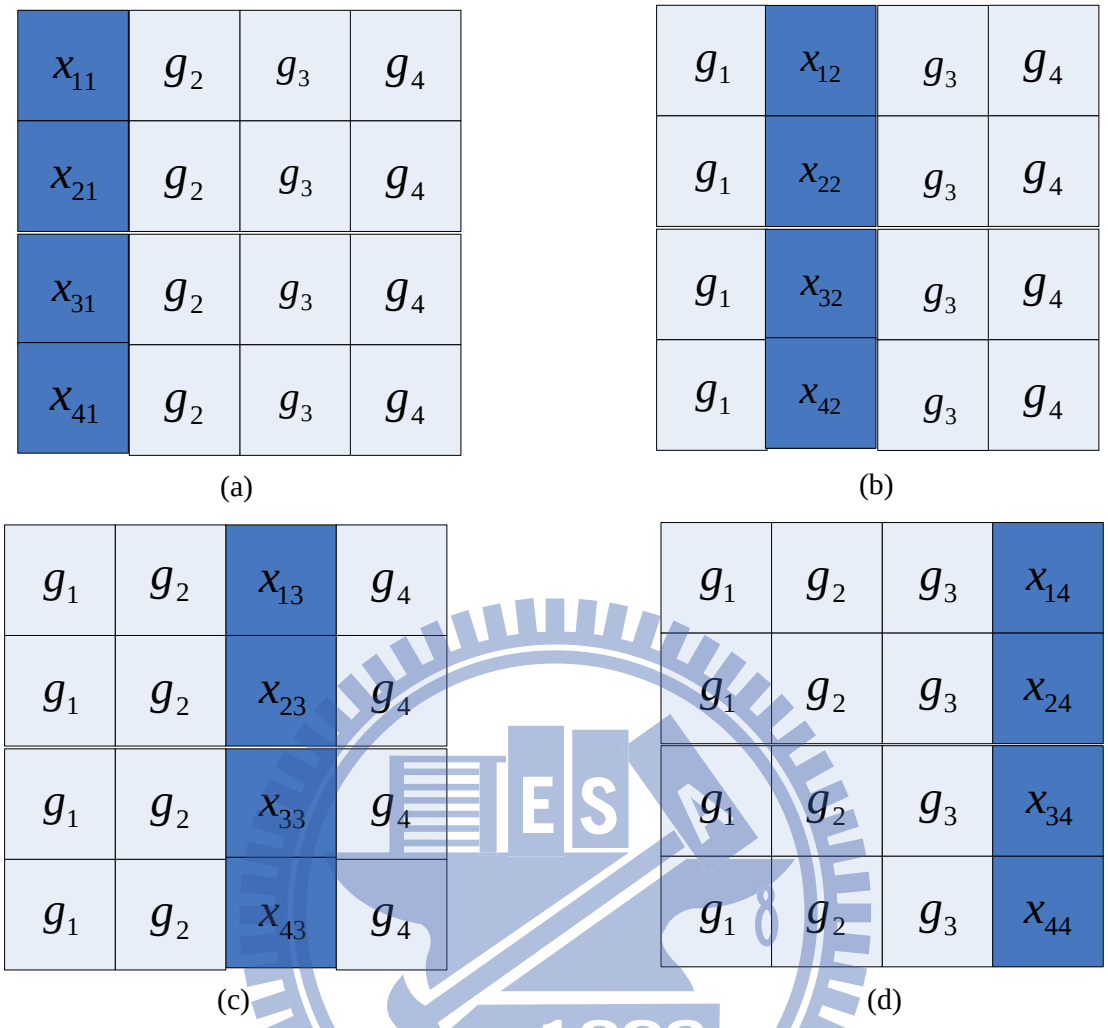


圖 2.2 CPSO-S 演算法示意圖: (a)表示第一個子群(維度)之間資訊的交換；(b)表示第二個子群(維度)之間資訊的交換；(c)表示第三個子群(維度)之間資訊的交換；(d) 表示第四個子群(維度)之間資訊的交換。

除了計算的複雜度大為提高之外，還有另外二個問題，其一是子群劃分即隱含了將各個維度對適應值函數的影響切割來看，對於有些維度間互相相關的問題這樣做卻會造成破壞性的效果使得他的表現甚至比起 PSO 還要來的差。其二是 CPSO-S 會落入假的最小值 (pseudominimizer)而使得 CPSO-S 連收斂於局部最優解都無法被保證。其原因即為 CPSO-S 將維度分割後的結果，然而 PSO 卻沒有這一方面的問題，因此 CPSO-H 這個以 CPSO-S 與 PSO 交替作用的方法希望結合 CPSO-S 當中競爭合作的機制與 PSO 可以避免落入假的最小值

(pseudominimizer)之特性，這裡就自然產生了一個問題，是否能夠找出一個方法來決定何時使用 CPSO-S，何時又使用 PSO 較佳，本論文即在以解決這個問題為動機來做研究。

2.3 增強式學習(reinforcement learning)

增強式學習(reinforcement learning)，是先定義每一種情況(state)的好或壞，也就是給機器一套價值觀，之後讓機器自行去學習。機器觀察很多資料以後，作出不同的反應即行動(action)，但是不同的反應可能會有不同的結果，結果的好壞就成為獎勵(reward signal) 的資料，讓機器學會下次遇到類似的情形(state)如何做出正確的行動(action)。 下圖 2.3 為增強式學習(reinforcement learning) 的架構。

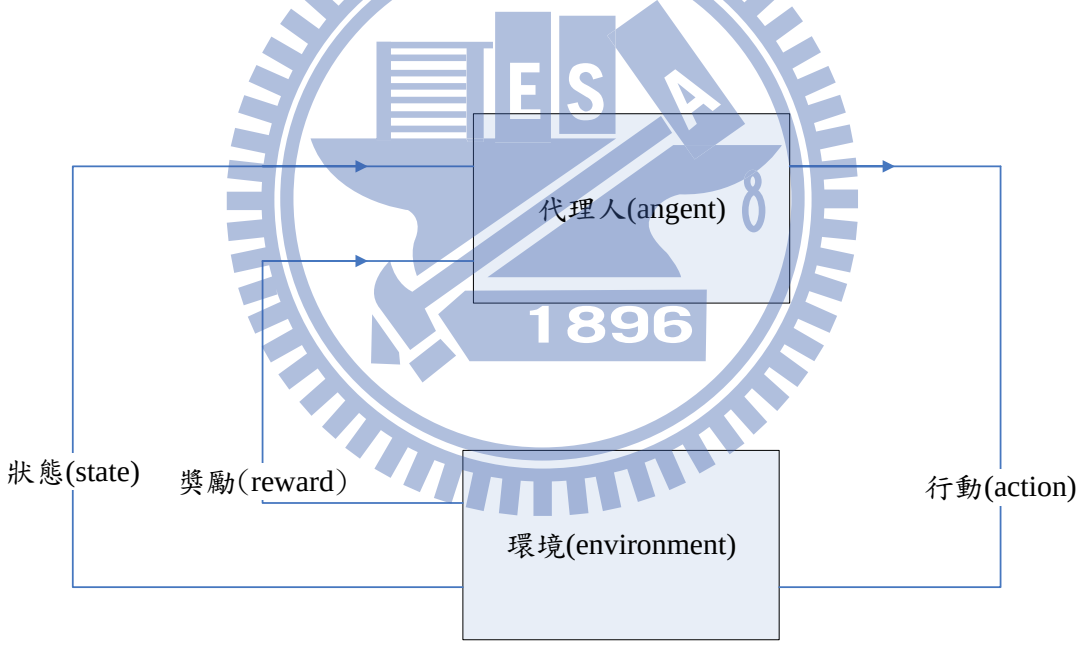


圖 2.3 增強式學習架構。

以下對增強式學習的一些基本元素做介紹:

1.狀態(states)：$\{s_1, s_2, \cdots, s_{n_s}\}$其中 n_s 為狀態的個數

2.代理人所做的的行動(actions) $\{a_1, a_2, \cdots, a_{n_a}\}$其中 n_a 為行動的個數

3環境:環境會對當前的狀態以及代理人所做的反應改變下一刻的狀態並給出獎

勵(reward)做為代理人行為好壞的評判依據。

因此代理人(agent)必須學習在每個不同狀態下所需做出的反應做選擇找出能夠獲得最大獎勵的行為依此想法就有了下列式:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{2.10}$$

上面 t 表示reinforcement learning task由 t 時間開始經過了T個學習步驟(step)為一個訓練過程(episode)而 γ 則為折扣率(discount factor)。增強式學習就是希望透過訓練能夠讓(2.10)能夠達到最大化。增強式學習若滿足下列(2.11)，(2.12)二式則稱馬可夫決定過程(Markov decision process):

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{2.11}$$

$$R_{ss'}^{a} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.12}$$

有了這個性質再加上動態規劃的方法(dynamic programming)則對應式(2.13)的狀態價值函數(stae-value function)表示了在代理人採取了策略 $\pi(s,a)$ 之下在 t 時刻狀態為 s 之情形下獎勵的期望值:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s\} \tag{2.13}$$

以及式(2.14)行動價值函數(action-value function)表示在代理人採取了策略 $\pi(s,a)$ 之下在 t 時刻狀態為 s 行為是 a 之情形下獎勵的期望值:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\} \tag{2.14}$$

最後以Bellman Equation 分別表示為式(2.15)和(2.16):

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\
&= E_{\pi}\{\sum_{k=0}^{T}\gamma^k r_{t+k+1} \mid s_t = s\} \\
&= E_{\pi}\{r_{t+1} + \gamma\sum_{k=0}^{T}\gamma^k r_{t+k+2} \mid s_t = s\} \\
&= \sum_a \pi(s,a)\sum_{s'} P_{ss'}^a[R_{ss'}^a + \gamma E_{\pi}\{\sum_{k=0}^{T}\gamma^k r_{t+k+2} \mid s_{t+1} = s'\}] \\
&= \sum_a \pi(s,a)\sum_{s'} P_{ss'}^a[R_{ss'}^a + \gamma V^{\pi}(s')\}]
\end{aligned}
\tag{2.15}
$$

$$
\begin{aligned}
Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} &= E_{\pi}\{\sum_{k=0}^{T}\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\} \\
&= \sum_{s'} P_{ss'}^a[R_{ss'}^a + \gamma V^{\pi}(s')\}]
\end{aligned}
\tag{2.16}
$$

式(2.15)和(2.16) 中的 $P_{ss'}^a$ 即為轉移機率(transition probability)說明了在 $t+1$ 時刻的狀態 s' 之機率僅由上一個時刻的狀態及行動所決定。 圖2.4(a)、2.4(b)分別為(2.15)、(2.16)式的倒傳遞，圖2.4中狀態 s 都由空心圓表示，含有行動 a 的都由實心圓表示。圖2.4(a)中 s 與 a 間的連線代表了狀態為 s 下產生行動 a 的機率 $\pi(s,a)$ ，而 a 與 s' 間的線段代表了在狀態 s 下挑選了行動 a 所產生狀態 s' 的轉移及獲得的獎勵 r。圖2.4(b)中 (s,a) 與 s' 間的線段代表了在狀態 s 下挑選了行動 a 所產生狀態 s' 的轉移及獲得的獎勵 r ，而 s' 與 a' 間的線段與圖2.4(a) s 與 a 間的連線代表的意義相同。

有了(2.15)式若我們可以知道增強式學習中每個狀態，每個行動對應的狀態轉移機率(transition probability) 即(2.11)式，則對應任意策略(policy)都可以透過寫出(2.15)式對每個狀態的狀態價值函數(state-value function)並求解聯立方程就能得到正確的解。不過通常的情況我們卻無法得知一個未知環境(系統)的狀態

轉移機率，為了處理這問題就有了蒙地卡羅法(Monte Carlo method)及差分學習(temporal-difference)被提出。

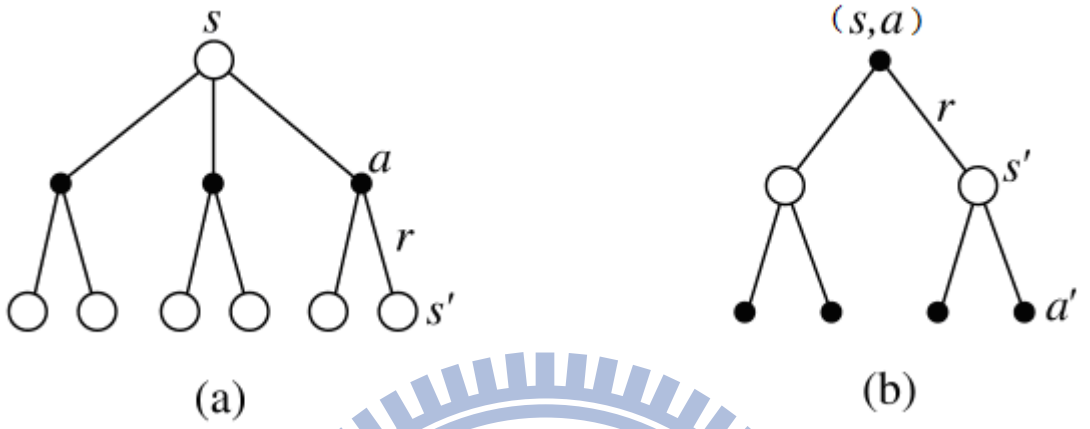


圖2.4 倒傳遞圖:(a)代表在狀態為 s 下 $V^{\pi}(s)$ 計算方式；(b)代表在狀態為 s 行動為 a 的情況下 $Q^{\pi}(s,a)$ 計算方式。

2.3.1 價值函數和策略的疊代(value iteration and policy iteration)

策略的疊代(policy iteration)主要是透過疊代的方式不斷改善策略下面序列可以表達此方法之機制:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^{\pi^*}$$

透過 E (逼近給定策略 π 的價值函數)與 I (改善現有的策略 π)交替的作用即策略的疊代(policy iteration)演算法的做法，下面是演算法的描述:

E 表示對不同策略(π)下價值函數(value function) V^{π} 的值利用疊代的方式去逼近，做法如下:

Step 1:$l=0$，對所有狀態s初始化$V^{\pi_l}(s)=0$。

Step 2:$k=0$，$V_k(s)=V^{\pi_l}(s)$，$l=l+1$，$\pi(s)=\pi_l(s)$。

Step 3:對所有狀態s，$V_{k+1}(s)=\sum_{s'}P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)}+\gamma V_k(s')]$，$\Delta=\max_s(\left|V_{k+1}(s)-V_k(s)\right|)$。

Step 4: $k=k+1$，若Δ<自訂的閥值，則輸出$V^{\pi_l}(s)=V_k(s)$，否則回到 Step 3。

I表示在取得策略π_l所對應的價值函數$V^{\pi}(s)$後對策略進行改善其做法如下:

Step 5:對所有狀態s，$\pi_{l+1}(s)=\arg\max_a\sum_{s'}P_{ss'}^{a}[R_{ss'}^{a}+\gamma V^{\pi_l}(s')]$。

S**tep 6**:，若$\pi_{l+1}(s)=\pi_l(s)$則停止，否則返回 Step 2。

最後由於策略的疊代(policy iteration)是由E (policy evaluation)與I (policy improvement)組成而I (policy improvement)必須要在E (policy evaluation)收斂到一定程度時($\Delta=\max_s(\left|V_{k+1}(s)-V_k(s)\right|)$<閥值)才能夠進行，即在$E$部分對所有狀態掃過數次，為了增進策略函數的疊代的效率有了價值函數疊代的想法，即在E部分並不等到$V_k(s)$收斂才進行I部分而是僅在E部分中執行 step 3 一次之後直接進入I部分，流程如下:

Step 1:$m=0$，對所有狀態s初始化價值函數$V_m(s)=0$。

Step 2:對所有狀態s，$V_{m+1}(s)=\max_a\sum_{s'}P_{ss'}^{a}[R_{ss'}^{a}+\gamma V_m(s')]$，$\Delta=\max_s(\left|V_{k+1}(s)-V_k(s)\right|)$。

Step 3:m=m+1，若Δ<自訂的閥值，則輸出$V_{m+1}(s)=\arg\max_a\sum_{s'}P_{ss'}^{a}[R_{ss'}^{a}+\gamma V_m(s')]$否則回到 Step 2。

價值函數的疊代(value iteration) 主要是用疊代的方式來找出$\pi^*(s,a)$即代理人處理增強式學習的目標，最佳的行動反應策略(policy)它能夠使得下面(2.17)式

成立:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.17}$$

也就是達到最大化期望的獎勵(maximum expected reward)，做法如圖2.5。

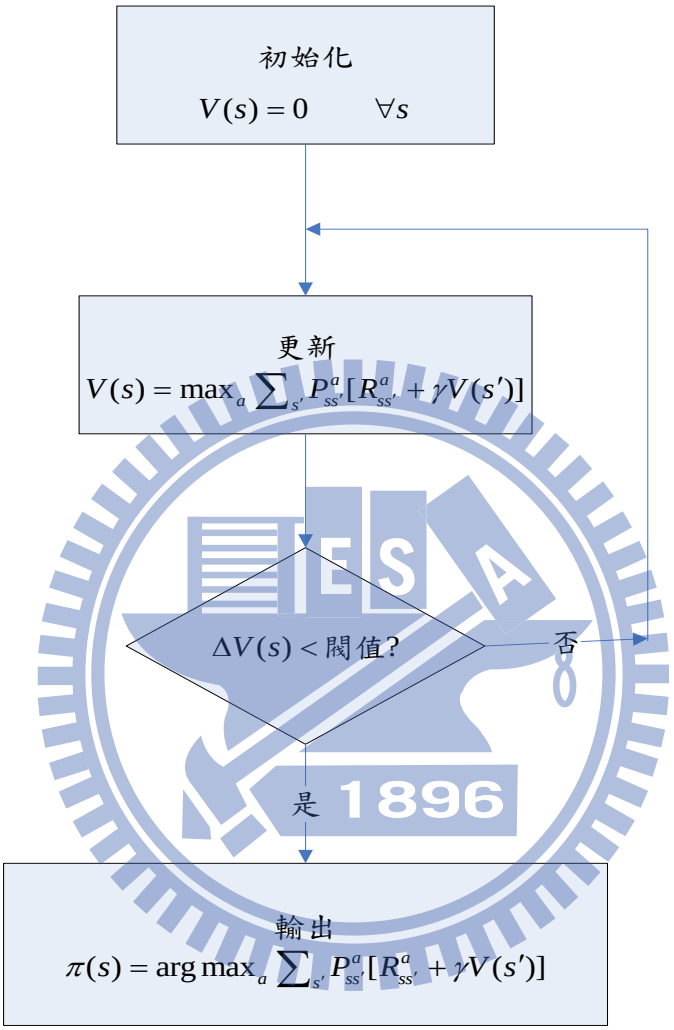


圖 2.5 價值函數疊代流程。

2.3.2 蒙地卡羅(Monte Carlo)法

上節的演算法給出了最佳化的策略(policy)如何在轉移機率(transition probability)已知的情況下利用疊代法加以求得，但如同之前提到的轉移機率實際上常常是未知的，此時就可以使用Monte Carlo來處理。Monte Carlo 法主要是將

模擬代理人與環境互動所得到的狀態，行動，獎勵這些資訊收集後利用平均的方法來求得對應於任意狀態、行動，期望的獎勵式(2.13)為何。由大數法則知道當能夠獲得的資訊越大那麼就越逼近式(2.13)真實的值，這種方法非常直觀也容易了解，但是由於他更新價值函數(value-function)的值必須等到一個學習過程(episode)完成之後若是學習過程中的學習步驟(step)數量過多那麼它更新每個狀態的速度過慢導致了求取最佳策略的疊代過程變得十分耗時，即學習的速度變得很緩慢。蒙地卡羅對給定的策略(π)求取的價值函數流程:

Step 1: $t=0$，對所有的狀態 s，初始化 $V_t(s)$。

Step 2:利用策略 π 產生一個訓練過程(episode)。

Step 3:對所有狀態 s，找出在這個訓練過程中(episode) 狀態 s 第一次出現之後的回傳獎勵(reward)，$R_t(s)$，$V_{t+1}^{\pi}(s)=V_t^{\pi}(s)+R_t(s)$。

Step 4: $V^{\pi}(s)=\frac{V_{t+1}^{\pi}(s)}{t+1}$，$t=t+1$，回到 Step 2。

圖 2.6 是蒙地卡羅一個訓練過程的倒傳遞圖它可以表示在 Step 3 中狀態 s 出現之後的回傳獎勵(reward)，$R_t(s)$，至於有多少次的行動選擇與狀態轉移必須看一個訓練過程的長短或是系統狀態是否到達終止狀態

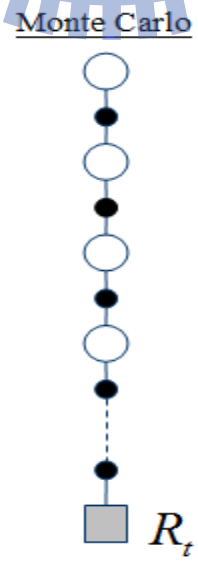


圖 2.6 蒙地卡羅一個訓練過程的倒傳遞圖。

2.3.3 差分學習(temporal difference learning)

差分學習方法是由 Sutton[5]所提出，這種方法結合了蒙地卡羅(Monte Carlo)以及動態規劃(dynamic programming)的優點他的更新方法如下列式(2.18):

$$V_{t+1}(s_t) = V_t(s_t) + \alpha [r_t + \gamma V_{t+1}(s_{t+1}) - V_t(s_t)] \quad (2.18)$$

觀察上式可以知道 $r_{t+1} + \gamma V_t(s_{t+1})$ 即利用動態規劃的想法僅由一個 step 後的獎勵 r_{t+1} 加上 s_{t+1} 狀態的價值函數(value-function)值做為經驗回傳(experience return)而不同於 Monte Carlo 將所有學習過程(episode)內的學習步驟(step)跑完後取得的經驗回傳(experience return)因此大大的提高了學習的效率，因為僅僅在一次與環境的互動之後我們馬上就能夠取得狀態 s_t 的 update，此外式(2.18)的疊代方法隱含了蒙地卡羅法對經驗回傳(experience return)取平均的效果因此他確實將動態規劃及蒙地卡羅法的優點融和在一起。流程如下:

Step 1:對給定的策略 π 初始化 $V^{\pi}(s)$，$t = 1$。

Step 2:任意選擇狀態 s_t 做為訓練過程的起始狀態。

Step 3:由 $\pi(s_t)$ 決定行動(action)，觀察獎勵 r_{t+1} 和下一個狀態 s_{t+1}，更新

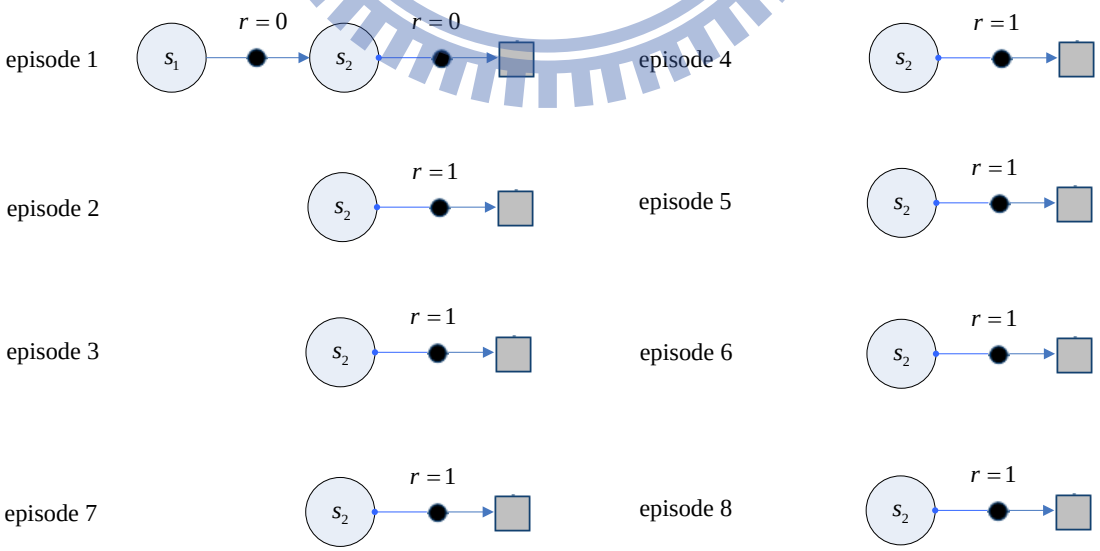
$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha(r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t))$。

Step 4: $t = t + 1$，若 $t < T$ （T 為一個訓練過程的總步數）則回到 Step 3。

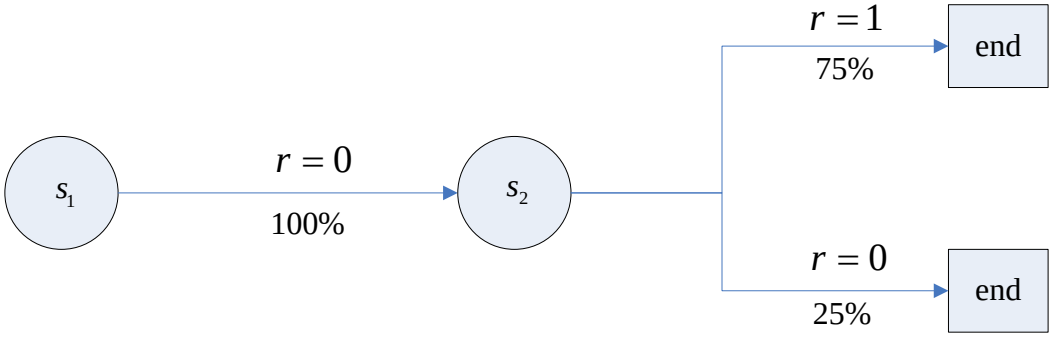
圖 2.7 是差分學習一個訓練過程的倒傳遞圖它代表了在 Step 3 中狀態 s_t 經過一次的行動選擇以及狀態轉移即可對 $V^{\pi}(s_t)$ 做更新的動作。

圖 2.7 差分學習一個訓練過程的倒傳遞圖。

下面舉一個例子可以說明差分學習的優點。假設有 8 個訓練過程第一個訓練過程起始狀態為 s_1 經過一次的行動選擇獲得獎勵 $r=0$ 系統狀態轉換到了 s_2 然後再經過一次的行動選擇獲得獎勵 $r=0$ 系統進入終止狀態(以方塊表示)，而第二到第八個的訓練過程都是起始狀態為 s_2 經過一次的行動選擇獲得獎勵 $r=1$ 系統進入終止狀態(以方塊表示)如圖 2.8(a)，若由蒙地卡羅法來計算 s_1 及 s_2 之價值函數則 $V(s_1)=0$ ，$V(s_2)=\frac{3}{4}$，而若透過差分學習風法來計算則 $V(s_1)=0$ ，$V(s_2)=\frac{3}{4}$ ，由蒙地卡羅來算 $V(s_1)$ 可以得到最小平方誤差(minimum square error)，但是若具有馬可夫過程特性的系統即圖 2.8(b)那麼差分學習這種方法可以透過式(2.18)反映出這種性質。



(a)



(b)

圖 2.8 (a)為具有二狀態(s_1、s_2)的 8 個訓練過程第一個訓練過程起始狀態為s_1經過一次的行動選擇獲得獎勵$r=0$系統狀態轉換到了s_2然後再經過一次的行動選擇獲得獎勵$r=0$系統進入終止狀態(以方塊表示)，而第二到第八個的訓練過程都是起始狀態為s_2經過一次的行動選擇獲得獎勵$r=1$系統進入終止狀態(以方塊表示)；(b)考慮系統具有馬可夫過程特性下的狀態轉移圖。

2.3.4 廣義的差分學習 TD(λ)

TD(λ)定義了λ-return 如式(2.19)，式(2.20)為在時刻t之n-step return:

$$R_t^{\lambda}=(1-\lambda)\sum_{n=1}^{T-t}\lambda^{n-1}R_t^{(n)}+\lambda^{T-t-1}R_t \tag{2.19}$$

$$R_t^{(n)}=r_{t+1}+\gamma r_{t+2}+\gamma^2 r_{t+3}+\cdots+\gamma^{n-1}r_{t+n}+\gamma^n V_t(s_{t+n}) \tag{2.20}$$

圖 2.9 左邊第一個表示了時刻t之n-step return $R_t^{(n)}$，$n=1$即狀態僅經過一次的行動選擇和一次的狀態轉換所得到的獎勵回傳$R_t^{(1)}$，左邊第二個則表示了$n=2$即狀態僅經過二次的行動選擇和二次的狀態轉換所得到的獎勵回傳$R_t^{(2)}$依此類推，而圖 2.9 最右邊的表示R_t包含了系統進入終止狀態前所有的獎勵回傳。

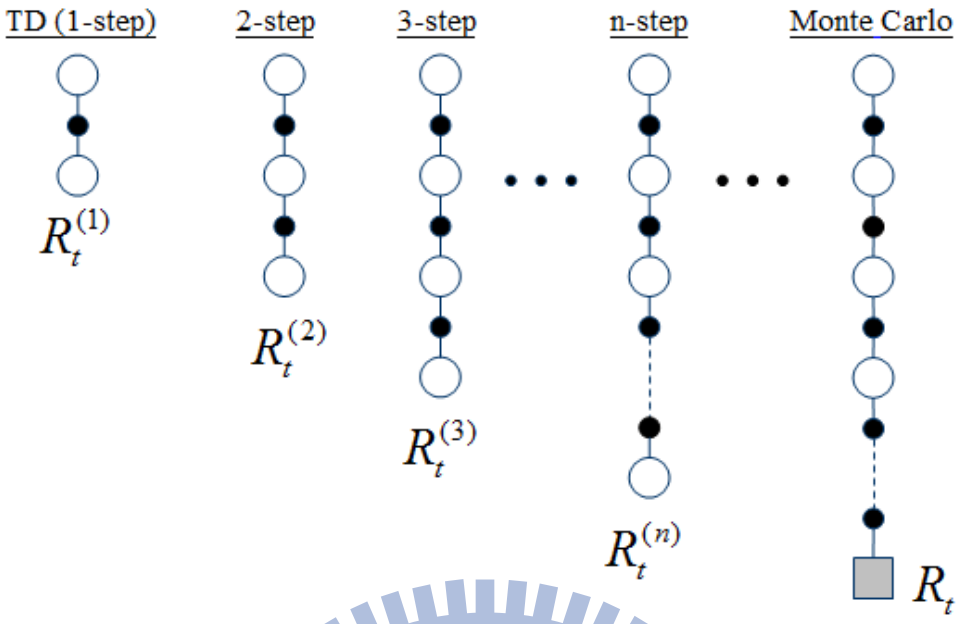


圖 2.9 廣義差分學習的倒傳遞圖。

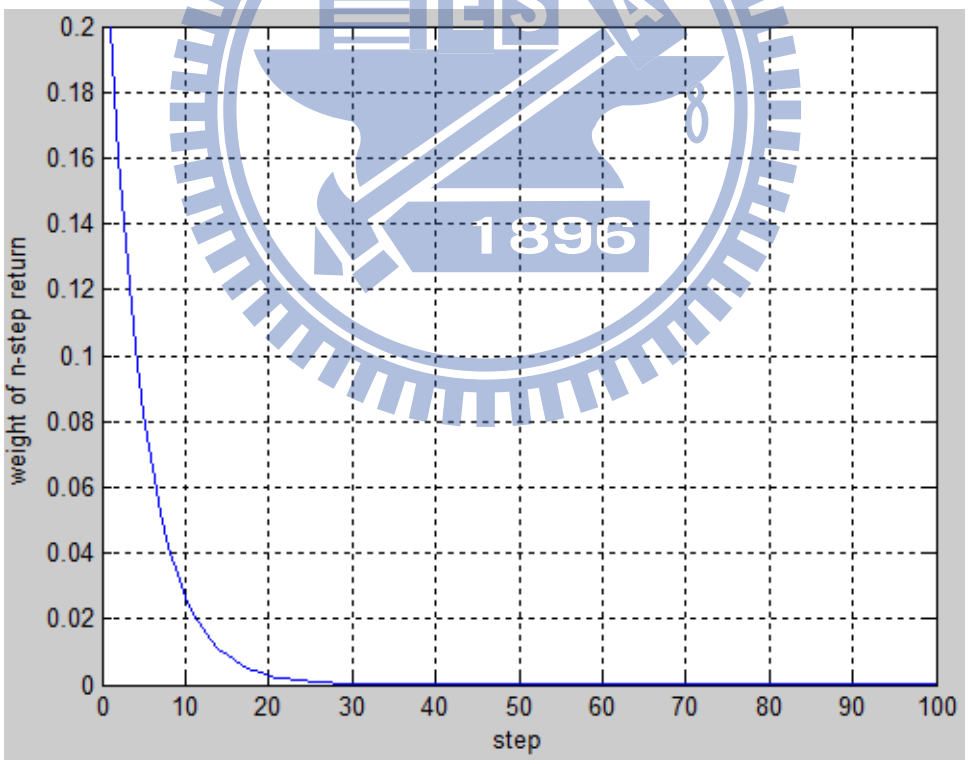


圖 2.10 $R_t^{(n)}$ 的權重分配。

圖 2.10 表示了 λ=0.8，t=1 時每一個 $R_t^{(n)}$ (n-step return)的權重大小 $(1-\lambda)\lambda^{n-1}$ ，其中橫軸表示 $R_t^{(n)}$ 當中的 n 值，縱軸為 $R_t^{(n)}$ 所對應的權重 $(1-\lambda)\lambda^{n-1}$ 。由式(2.19)可知當 λ=1 時式(2.19)就成為了式(2.10) 而式(2.10)即為在蒙地卡羅法中所使用的回傳，當 λ=0 時式(2.19)就成了 $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$ 這即為 TD 法中所使用的回傳。由上述可知 TD(λ)是一種介於蒙地卡羅法和差分學習之間的一種學習方法。

2.3.5 基於廣義差分學習的行動價值函數(SARSA(λ))

由於若我們僅知 $V^{\pi}(s)$ 即僅有每個狀態的價值函數則在沒有系統狀態的轉移機率 $P_{ss'}^{a}$ 的情況下無法藉由 $V^{\pi}(s) = \sum_{a}\pi(s,a)\sum_{s'}P_{ss'}^{a}[R_{ss'}^{a} + \gamma V^{\pi}(s')\}]$ 找出適當的 $\pi(s,a)$ 來最佳化 $V^{\pi}(s)$ ，而若求取 $Q^{\pi}(s,a) = \sum_{s'}P_{ss'}^{a}[R_{ss'}^{a} + \gamma V^{\pi}(s')\}]$ 則可以避開 $P_{ss'}^{a}$ 未知的問題。SARSA(λ)為 $Q^{\pi}(s,a)$ 其中的一種演算法，借由 $Q^{\pi}(s,a)$ 即可對所有狀態 s 找出使 $V^{\pi}(s)$ 最大化之行動 a 。

SARSA(λ)是一個行動價值函數(action-value)它提供了增強式演算法在控制方面的應用圖 2.11 可表示一個控制的過程他更新行動價值函數的方式即利用了 TD(λ)、資格追蹤(eligibility trace)來執行。

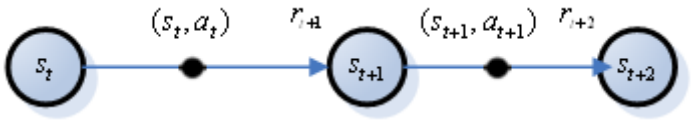


圖 2.11 SARSA 用於控制的簡單示意圖。

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \qquad \text{對所有 } (s,a) \tag{2.21}$$

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{2.22}$$

$$e_t(s,a)=\begin{cases}\gamma\lambda e_{t-1}(s,a)+1 & \text{若} s=s_t, a=a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{否則}\end{cases} \tag{2.23}$$

SARSA(λ)演算法流程:

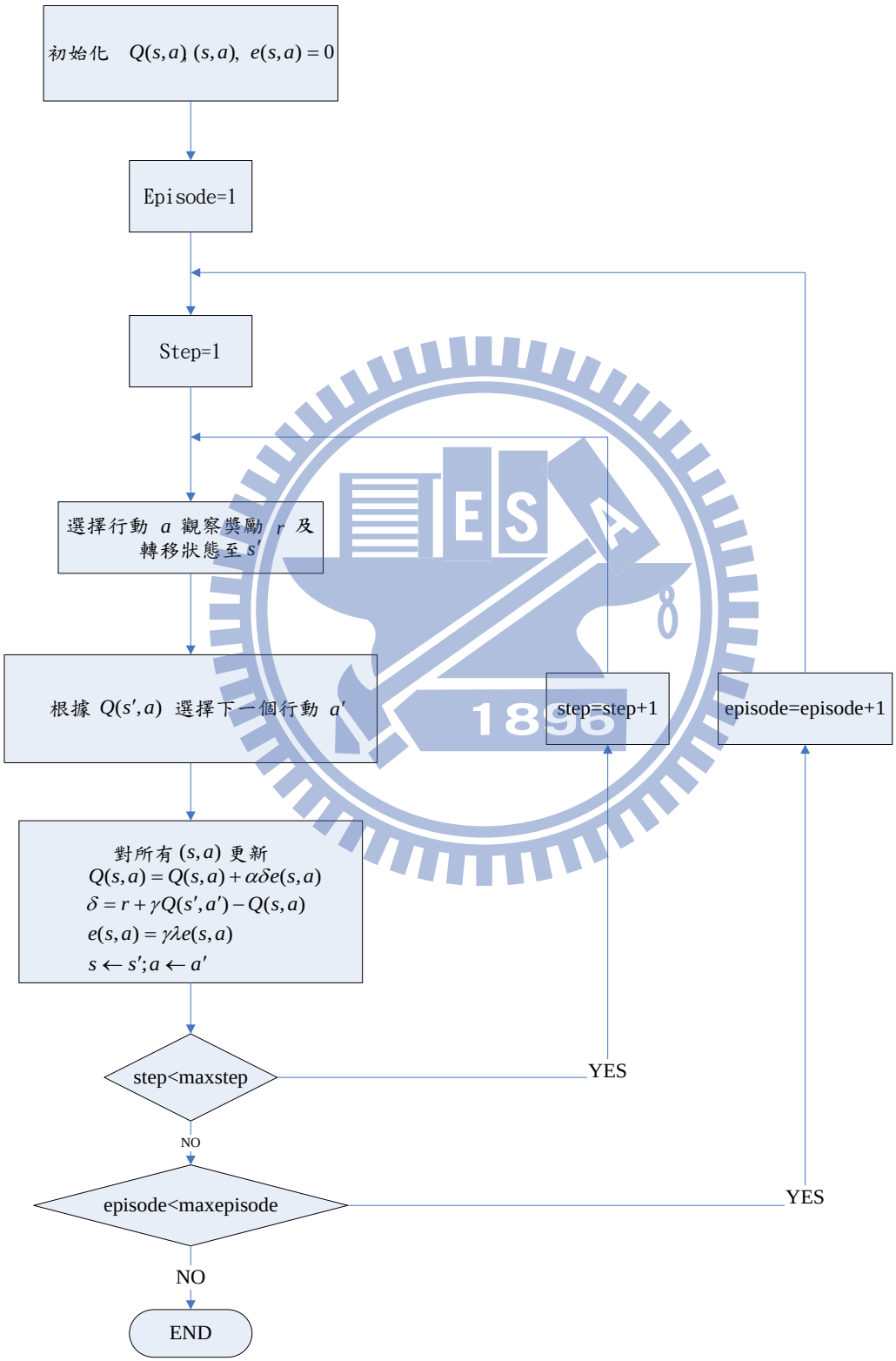


圖 2.12 SARSA 演算法流程。

第三章
使用 SARSA(λ)決定基本粒子群及協同式粒子群切換時機

在上一章提到了由於 PSO 演算法存在著兩步向前一步退後的問題，並且 CPSO-S 存在著落入假的極小值(pseudominimizer)的問題，變成這兩種方法必須交替的使用來避免上述問題的發生，但是要是我們能透過 SARSA(λ)找出在哪些狀態下，使用 PSO 較有利或是使用 CPSO-S 較有利，那麼就能夠避免掉執行不必要的 CPSO-S 所需要的大量時間，或是避免了 PSO 出現兩步向前一步向後的問題，加快最佳解的找尋。本章 3.1 節說明本論文提出的演算法的想法，所遭遇的問題在 3.2 節說明解決的方法，3.3 節則是對演算法流程及細節做說明。

3.1 演算法簡述

由於基本粒子群最佳化以及協同粒子群最佳化在尋找函數極值上各有優點與缺點，為了彌補這兩個演算法的缺點，以及使用這兩個演算法的優點，所以有混合式粒子群最佳化(CPSO-H)方法被提出，不過由於這個方法是使用基本粒子群最佳化以及協同式粒子群最佳化兩法交替作用，這產生了一個問題就是有些函數極值搜尋的問題里各個維度間是彼此相關的，這時候我們若還是將協同粒子群演算法與基本粒子群演算法交互作用非但無法讓協同式粒子群最佳化當中粒子群的多樣性對尋優過程產生幫助，反而因為粒子群的多樣性造成評估粒子群優

劣所必須花的時間大幅增加。

本論文想要利用 SARSA(λ)演算法來做 PSO 與 CPSO-S 在不同狀態下的切換時機並且將這種方法用於函數最佳值的找尋，但是將會面臨到一個困難即為考慮到搜尋空間的維度在非常高的時候若是直接對每個不同維度的搜索區間直接做離散化將會遭遇到維度災難(curse of dimension)，這將會使得在使用 SARSA(λ)時出現狀態數隨著處理函數最佳值搜尋問題中函數的維度呈指數型的增加，因此不能夠直接以不同維度的搜索區間直接做離散化來作為 SARSA(λ)當中的狀態。

圖 3.1 SARSA(λ)用於切換時機的決定。

透過 CMA-ES 中斜方差矩陣(covariance matrix)的啟發，本論文將利用這個方法找到粒子群之中每個粒子所在的區域的最佳解分佈。利用斜方差矩陣

(covariance matrix) 求出主成分向量(principal component vector)，以這個向量和每個粒子與當前群體最優 gbest 向量之夾角做為 SARSA(λ)狀態的一個部分，另一個部分即利用每個粒子與當前群體最優解的歐氏距離做為另一個部分，至此狀態部分便處理完畢。行動分為兩個部分其一為原始的 PSO，其二則為 CPSO-S。此外為了讓代理人能夠學習必須要定義獎勵(reward)，經過一些試驗發現到在求解函數極值這個問題上利用粒子群在上一時刻 t 的函數值與下一時刻 $t+1$ 的函數值之間的差做為獎勵(reward)能獲得最好的訓練效果。

3.2 粒子群由搜索空間轉換至狀態空間

本節將說明如何透過斜方差矩陣以及主成分分析的方法將粒子群最佳化裏面粒子所在的搜尋空間轉換至 SARSA(λ)當中所需要的狀態空間。並說明粒子群最優位置更動時的問題及處理方法。

3.2.1 粒子群個體的斜方差矩陣(covariance matrix)計算

由 N. Hansen 和 A. Ostermeier [12]的(基於自適應斜方差矩陣的演化策略)CMA-ES 中所提到的斜方差矩陣(covariance matrix)，本論文利用這個想法來求取粒子群中每個粒子所在區域中最佳解的分佈情形。我們假設最佳解分佈為以 m 為中心，以 C_μ 為斜方差矩陣的常態分佈，其中 C_μ 求法是在每個粒子群中以粒子所在位置 x 為中心以 σ 為標準差的常態分佈產生 λ 個點如式(3.1)所示:

$$x' \sim x + N(0, \sigma I) \tag{3.1}$$

並以這些點的適應值優劣來排序取出最好的前 μ 個點，再利用式(3.2)、(3.3)求出斜方差矩陣(covariance matrix) C_μ。由圖 3.2 可以很清楚的發現利用 C_μ 能夠

獲得粒子群當中的個體在此區域函數梯度變化最快的方向。圖 3.2 即代表了當求函數最佳化問題中維度為二時利用斜方差矩陣所得到的結果。

$$m = \frac{1}{\mu}\sum_{i=1}^{\mu} x'_{i:\lambda} \tag{3.2}$$

$$C_{\mu} = \sum_{i=1}^{\mu} w_i (x'_{i:\lambda} - x)(x'_{i:\lambda} - x)^T \tag{3.3}$$

這裡 $x_{i:\lambda}$ 中 $i:\lambda$ 代表 λ 個取樣點當中適應值排在第 i 個優良的點。

(a) (b) (c)

圖 3.2 [12] (a) 對 x 為均值(mean)，σ 為方差的常態分佈取樣 λ 個取樣點；(b)由 λ 個取樣點中找出適應值最好的前 μ 個點；(c) 對 x 為均值(mean) μ 個點形成的斜方差矩陣(實線橢圓)。

3.2.2 主成分分析(principal component analysis)

有了斜方差矩陣 C_{μ} 後並利用主成分分析(PCA)找出在所有表現較好的前 μ 個點分佈下投影在這個向量上能得到最大方差(variance)的向量，假設此向量為 u 那麼即是找尋使得下式有最大值的 u ，且 $\|u\|_2 = 1$:

$$\max\{\frac{1}{\mu}\sum_{i=1}^{\mu}((x'_{i:\lambda}-x)^T u)^2\} = \max\{\frac{1}{\mu}\sum_{i=1}^{\mu}u^T(x'_{i:\lambda}-x)(x'_{i:\lambda}-x)^T u\}$$

$$= \max\{u^T(\frac{1}{\mu}\sum_{i=1}^{\mu}(x'_{i:\lambda}-x)(x'_{i:\lambda}-x)^T)u\}$$

$$= \max\{u^T C_\mu u\}$$

所以u即為對應C_μ有最大特徵值之特徵向量。

圖 3.3 表示了將二維的資料(佈在二維平面均值(mean)為$(0,0)$)透過主成份分析的方式最適當的以一維的方法來表示。圖 3.3(a)可以看到有 5 個資料點若它們的斜方差矩陣為C則斜方差矩陣C所對應的最大與最小單位特徵向量分別與圖 3.3(b)、3.3(c)的直線向量相同，參看圖 3.3(b)、3.3(c)可以觀察到 5 個資料點投影在圖 3.3(b)通過座標中心$(0,0)$有最大特徵向量的直線上的 5 個點與資料點的均值(mean) $(0,0)$差的平方和，比起 5 個資料點投影在圖 3.3(c)通過座標中心$(0,0)$有最小特徵向量直線上的 5 個點與資料點的均值(mean) $(0,0)$差的平方和相比下，可以很容易由圖看出來投影在有最大特徵向量的直線上也就是圖 3.3(b)比起投影在有最小特徵向量的直線上也就是圖 3.3(c) ，圖 3.3(b)有較大的值，在考慮只由一維向量表示二維資料的情況下，我們選擇圖 3.3(b)中的直線確實能夠將資料點的斜方差矩陣C的資訊最大限度的保留下來。

最後由於粒子群的當前最優解並非是不會更動位置的，考慮到粒子群最優解若是在t時刻與$t+1$時刻出現移動，則會產生t時刻搜索空間任意一個粒子與當時最佳解的向量與$t+1$時刻同一個特定點與當時最佳解向量產生了不同，這會造成t時刻與$t+1$時刻每個搜尋空間的點都對應到了與先前不同之狀態，所以我們利用讓每個粒子在進行搜索空間與狀態空間的轉換之前先加上$t+1$時刻與t時刻最佳解位置的差作為補償即可解決此問題。

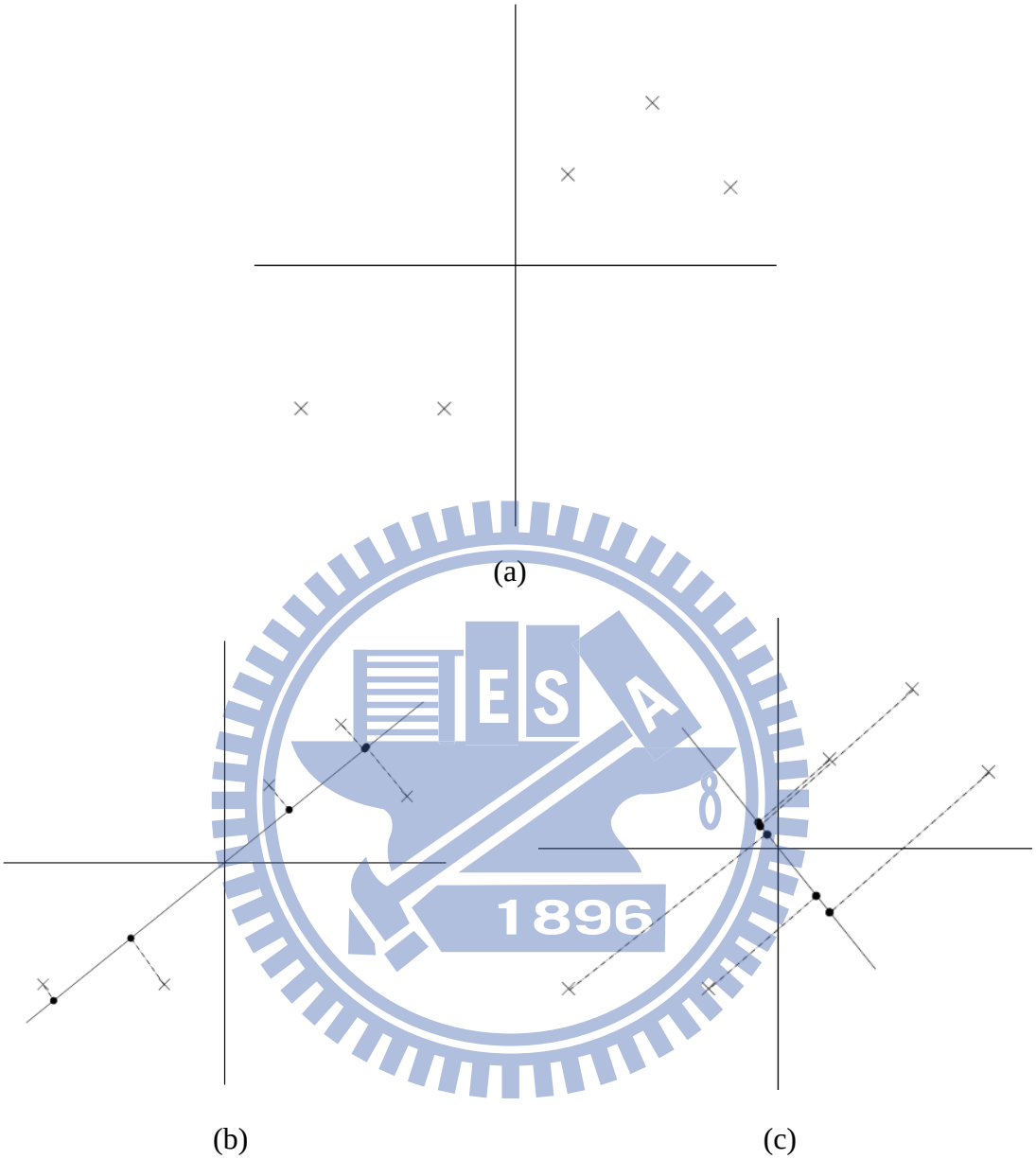


圖 3.3 主成份分析說明

(a)在二維平面的資料；(b)通過(0,0)及對應資料的斜方差矩陣最大特徵向量的一維直線以及資料投影在其上的點；(c) 通過(0,0)及對應資料的斜方差矩陣最小特徵向量的一維直線以及資料投影在其上的點。

3.3 演算法流程

上一節處理了狀態、行動、獎勵這三方面的問題之後，就可以對所有的狀態

及行動建立一個行動價值函數(action-value function)使用在第二章當中所介紹用於決策方面(decision making)的 SARSA(λ)來計算出行動價值函數，並以此做為在不同狀態下選擇 CPSO-S 與 PSO 行動的依據。由於粒子群當中每個粒子所經過的區域並不相同得到的獎勵(reward)也不同，所以每個粒子群中的粒子在一整個訓練過程中(episode)的每一個訓練步驟(step)都擁有自己的行動價值函數 $Q_t(s,a)$，其中 t 代表了第 t 個步驟，在每一個訓練步驟中粒子群都先透過 3.2 節當中所介紹的方法將粒子群由搜索空間轉換至狀態空間，並藉由 $Q_t(s,a)$ 來得到具有最佳值的行動 a ，由於每個粒子群對應 $Q_t(s,a)$ 值所得到的最佳行動並不相同，而不論基本粒子群最佳化或是協同粒子群最佳化均必須整個群體採取同一個演算法，故本論文將透過計算整個粒子群當中執行協同式粒子群最佳化這個行動的粒子個數若超過整個粒子群個數的一半，即整個粒子群當中採取基本粒子群最佳化的粒子個數不到整個粒子群個數的一半時，採取協同式粒子群最佳化，反之則採取基本粒子群最佳化。決定了採取的演算法之後即讓粒子群透過所採取的演算法進行在搜索空間的移動完成後透過粒子在搜索空間移動前與移動後之間的函數值之差作為這一個訓練步驟的獎勵(reward)，再經由 SARSA(λ)的方法(式(3.4)，(3.5)，(3.6))更新 $Q_{t+1}(s,a)$ 即第 $t+1$ 個 step 的行動價值函數(action-value function)，如此便完成一個訓練的步驟。

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \quad \text{對所有}(s,a) \tag{3.4}$$

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t) \tag{3.5}$$

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{若} s = s_t, a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{否則} \end{cases} \tag{3.6}$$

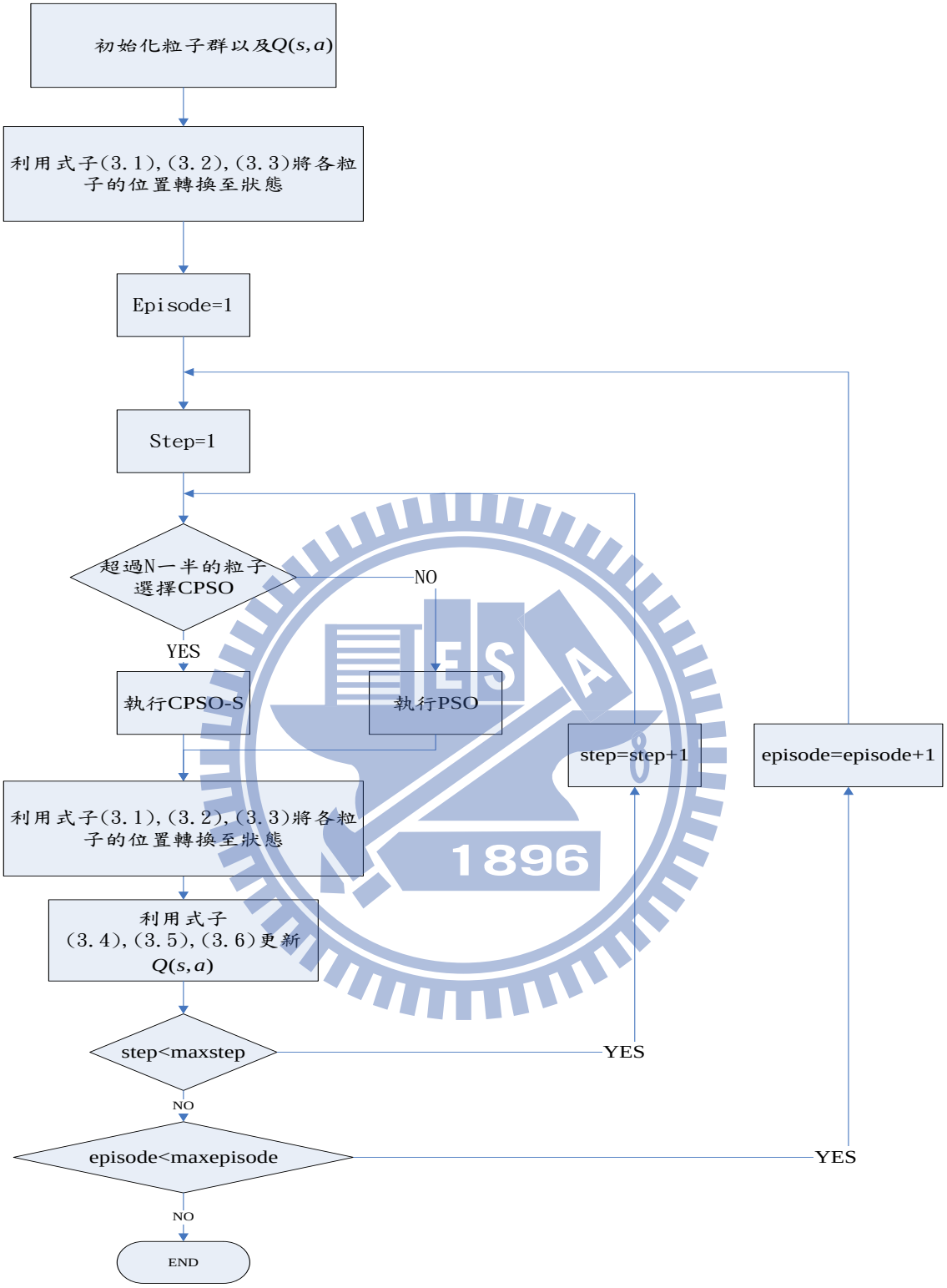


圖 3.4 本論文演算法流程。

第四章
實驗結果與分析

本章將利用第三章所提出的使用 SARSA(λ)決定基本粒子群及協同式粒子群切換時機，並與 PSO、CPSO-S、CPSO-H 演算法做比較。

本實驗研究設備為 Intel Core 2 Duo 2GHz 中央處理器，1GB 記憶體的筆記型電腦，使用的軟體為 Matlab R2008b 做為撰寫程式以及實驗模擬軟體。 4.1 節介紹測試函數以及實驗的數據，4.2 節為結果的分析。

4.1 函數極值找尋

這一節將使用測試函數 Ackley 、Griewangk 、Schwefel 與 Rastrigin，做為演算法 PSO、CPSO-S、CPSO-H，以及本論文所提出的方法測試它們在相同的訓練步數下找尋函數極值的表現，並將比較所得到的結果。

4.1.1 測試函數的選取

本文中用於函數極值演算法的測試函數有 4 種，其中包含 Ackley 函數、Griewangk 函數、Schwefel 函數、Rastrigin 函數等 4 種，以 f_1、f_2、f_3、f_4 表示，各方程式如式 (4.1)、(4.2)、(4.3)、(4.4) 所示

$$f_1 = -20e^{-b\sqrt{\frac{\sum_{i=1}^{n} x_i}{n}}} - e^{\frac{\sum_{i=1}^{n}\cos(cx_i)}{n}} + 20 + e \qquad (4.1)$$

$$f_2 = 10n + \sum_{i=1}^{n-1}(x_i^2 - 10\cos(2\pi x_i)) \quad (4.2)$$

$$f_3 = \sum_{i=1}^{n} -x_i \sin\sqrt{|x_i|} \quad (4.3)$$

$$f_4 = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2 \quad (4.4)$$

測試函數以 30 維來做測試即式(4.1)、(4.2)、(4.3)、(4.4)中的 n 均為 30。

4.1.2 參數設定

本論文實驗所使用的基本粒子群最佳化、協同式粒子群最佳化、混合式粒子群最佳化的演化公式如式(2.2)、(2.3)、(2.6)所示，參數選擇則依據 R.C. Eberhart and Y. Shi[9]。SARSA(λ)部分的行動價值函數更新公式如式(2.21)，(2.22)，(2.23)所示。

1.PSO: c_1=1.05，c_2=3.05，w=0.7298

2.CPSO-S: c_1=1.05，c_2=3.05，w=0.7298

3.CPSO-H: c_1=1.05，c_2=3.05，w=0.7298

4.本論文: c_1=1.05，c_2=3.05，w=0.7298，

SARSA(λ)的參數的部分利用De Jong function($f = \sum_{i=1}^{n} x_i^2$) 對學習效率、資格追蹤、隨機挑選行動的機率，在γ = 0.9下找出表現最佳的參數組合，表4.1列出了參數實驗的範圍。

表 4.1 學習參數的測試範圍

學習參數	測試範圍
α	[0, 0.2, 0.4, 0.6, 0.8, 1]
λ	[0.2, 0.4, 0.6, 0.8, 0.9, 0.95,1]
ε	[0.01, 0.05, 0.1]

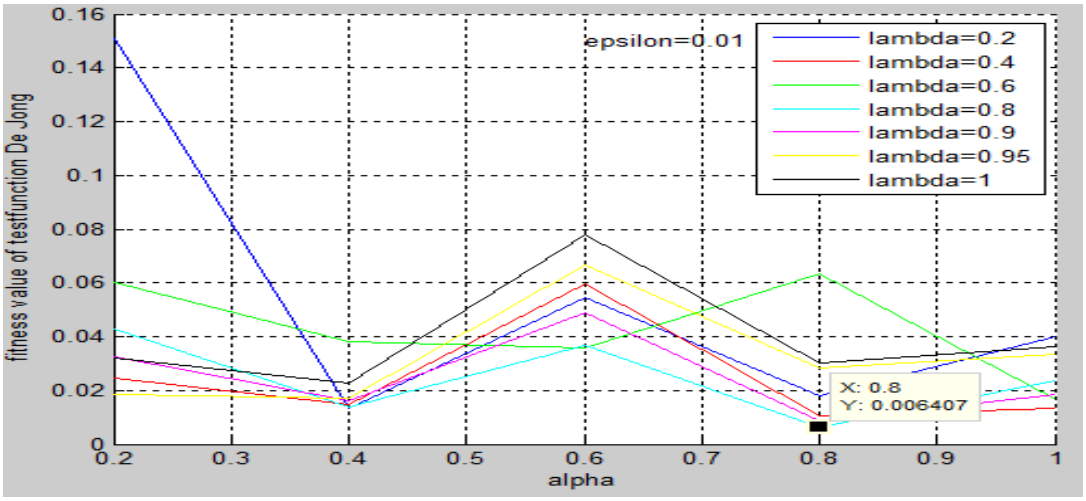


圖4.1測試函數為De Jong不同的學習參數下的結果(epsilon=0.01)。

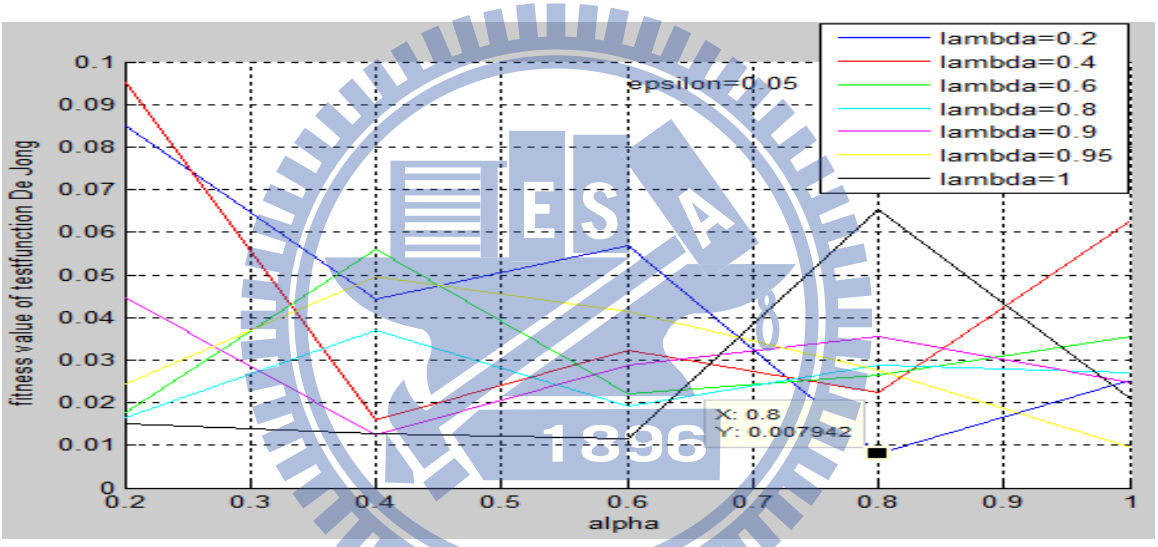


圖4.2測試函數為De Jong不同的學習參數下的結果(epsilon=0.05)。

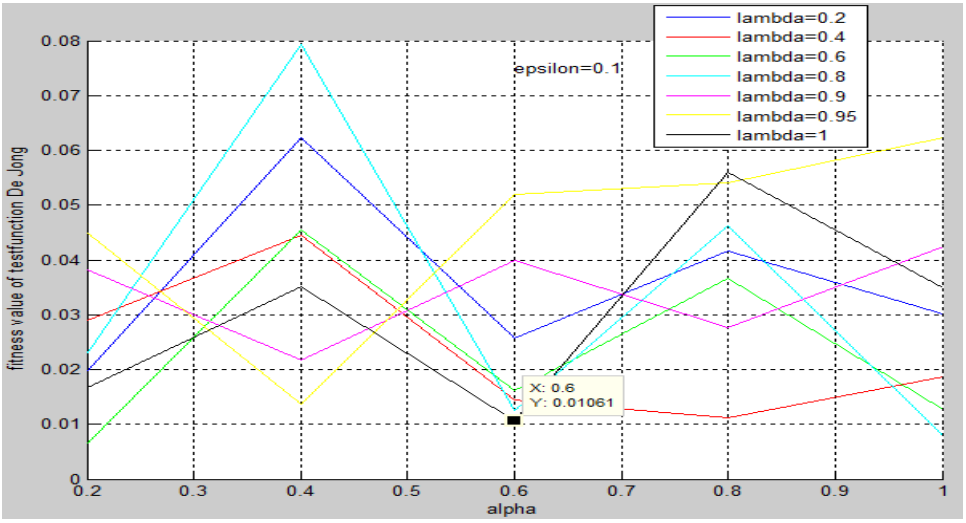


圖4.3測試函數為De Jong不同的學習參數下的結果(epsilon=0.1)。

由實驗得出圖4.1、4.2、4.3，經觀察得到最佳的參數如下:

$\gamma = 0.9$　折扣率(discount factor) 代表獎勵(reward)隨時間(訓練步驟)遞減的程度。

$\lambda = 0.8$ 資格追蹤(eligibility trace)的權重大小。

$\varepsilon = 0.01$ 代表隨機挑選行動的機率，值越大代表SARSA(λ)探索能力越大。

$\alpha = 0.8$ 學習效率(learning rate)代表SARSA(λ)在更新行動價值函數的快慢。

表 4.2 各測試函數的測試範圍

函數	函數維度	測試範圍
Ackley	30	[-32.76 32.76]
Griewangk	30	[-100 100]
Schwefel	30	[-500 500]
Rastrigin	30	[-5.12 5.12]

由於 PSO，CPSO-S，CPSO-H 及本論文之間演算法的差異每個演算法在位置與速度更新之間所需的適應值評估(function evaluation)次數並不相同為了公平起見，本實驗先用本篇論文題出的演算法來執行極值搜尋，並讓所有演算法執行的時間一致，來比較函數極值搜尋的結果好壞。

4.1.3 函數極值找尋實驗結果

表 4.3，4.4，4.5，4.6 分別是在各演算法在測試函式，f_1 (Ackley) ，f_2 (Griewangk)，f_3(Schwefel)，f_4 (Ratrigin)執行演算法時間相同下各演算法在不同粒子群數目下最優的解。圖 4.4，4.5，4.6，4.7 則分別是對應 f_1 (Ackley) ，f_2 (Griewangk)，f_3(Schwefel)，f_4 (Ratrigin)在維度二下，函數的圖形。

表 4.3 針對 Ackley 函數使用不同演算法之結果

測試函數	演算法	各維度粒子數	函數值
f_1	本論文	10	0.047
		15	0.0099
		20	3.67e-13
	PSO	10	3.99
		15	2.66
		20	2.09
	CPSO-S	10	1.75
		15	1.66
		20	1.9
	CPSO-H	10	1.37
		15	1.12
		20	0.51

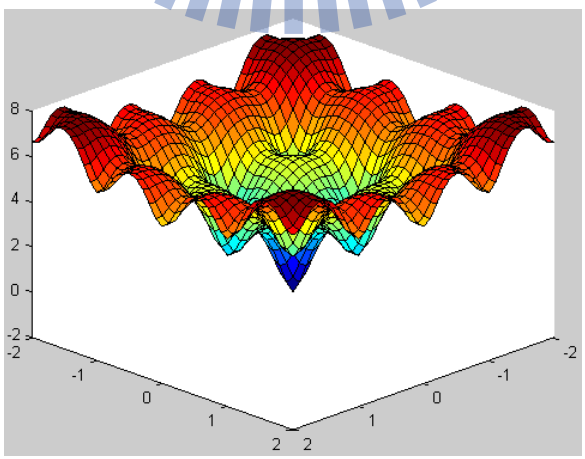


圖 4.4 Ackley 函數圖(維度為 2)。

表 4.4 針對 Griewangk 函數使用不同演算法之結果

測試函數	演算法	各維度粒子數	函數值
f_2	本論文	10	2.3e-6
		15	2.6e-10
		20	2.5e-12
	PSO	10	0.0806
		15	0.0296
		20	0.00417
	CPSO-S	10	12.36
		15	9.26
		20	7.53
	CPSO-H	10	2.17
		15	1.32
		20	4.68

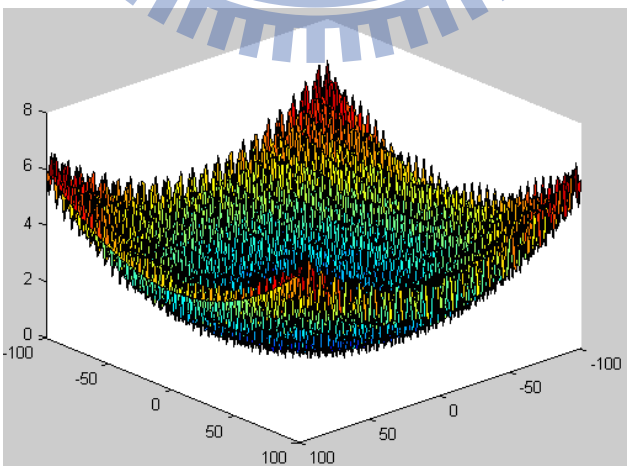


圖 4.5 Griewangk 函數圖(維度為 2)。

表 4.5 針對 Schwefel 函數使用不同演算法之結果

測試函數	演算法	各維度粒子數	函數值
f_3	本論文	10	-6476
		15	-7817
		20	-10610
	PSO	10	-5518
		15	-7041
		20	-8726
	CPSO-S	10	-3191
		15	-3548
		20	-4934
	CPSO-H	10	-2874
		15	-3614
		20	-4668

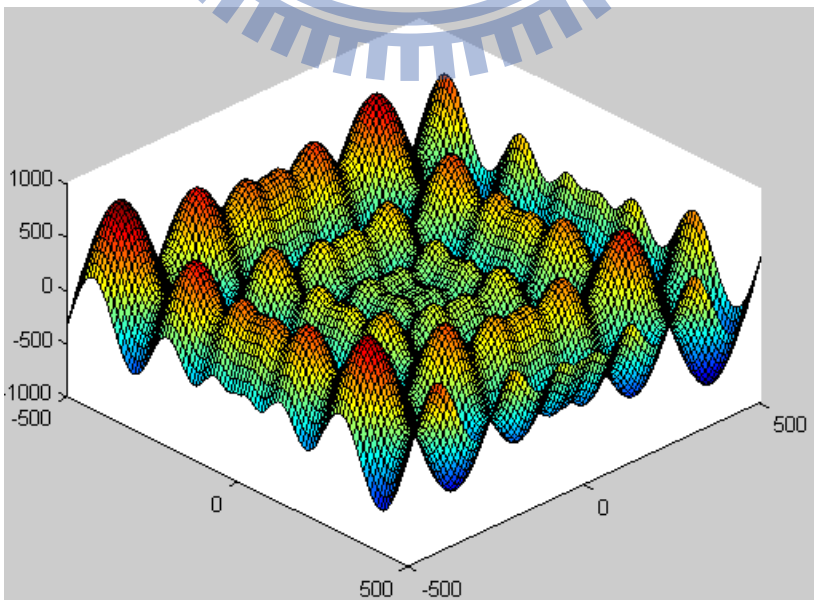


圖 4.6 Schwefel 函數圖(維度為 2)。

表 4.6 針對 Rastrigin 函數使用不同演算法之結果

測試函數	演算法	各維度粒子數	函數值
f_4	本論文	10	118.5
		15	75.4
		20	29.1
	PSO	10	2164.3
		15	2058.7
		20	1019.2
	CPSO-S	10	1345.6
		15	1152.5
		20	1053.2
	CPSO-H	10	553.6
		15	401.2
		20	398.7

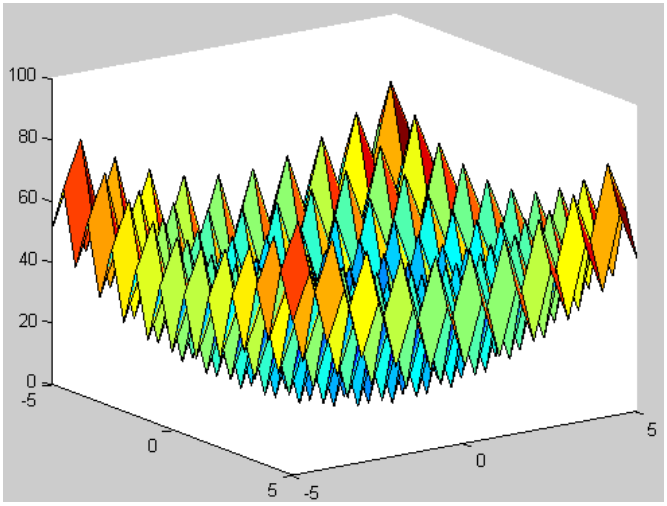


圖 4.7 Rastrigin 函數圖(維度為 2)。

4.2 結果分析

在表 4.3 與圖 4.4 及式(4.1)中可以看到由於函數之間的維度是不相關的，所以理論上應該對 CPSO-S、CPSO-H 這兩種演算法在尋找函數極值上較有利，但是由實驗可以知道本論文的表現為最佳，CPSO-S、CPSO-H 次之，主要由於此函數具有多個次級的區域極小值而這些區域極小值造成了 CPSO-S、CPSO-H 會發生落入假的最小值，CPSO-S 由於沒有使用 PSO 所以結果較 CPSO-H 差，而 PSO 在這個問題上結果最不理想。

在表 4.4 與圖 4.5 及式(4.2)中可以看到函數維度間是不相關的，所以理論上應該對 CPSO-S、CPSO-H 這兩種演算法在尋找函數極值上較有利，但卻出現了本論文結果最佳 PSO 次之，CPSO-S、CPSO-H 反而表現最不好，觀察圖 4.3 之後可以發現這個函數具有十分多的區域極小值，因此可以知道 CPSO-S、CPSO-H 很容易的就落入了區域的極小，又雖然 CPSO-H 方法中包含了 PSO 但是卻因為族群多樣性比 PSO 多即相同演算法的執行時間下粒子群位置及速度更新次數過少使得表現的比 PSO 還要差，而本論文由於透過了 SARSA(λ)每一次的粒子群未至極速度更新時，都可以藉由行動價值函數的值來得知使用協同式粒子群是否會落入區域極小，所以能夠有最好的表現。

在表 4.5 與圖 4.6 及式(4.3)中可以知道函數維度間是不相關的，理論上應該對 CPSO-S、CPSO-H 這兩種演算法在尋找函數極值上較有利，但卻出現了本論文結果最佳 PSO 次之，CPSO-S、CPSO-H 反而表現最不好，觀察圖 4.6 之後可以發現這個函數具有十分多的區域極小值，因此可以知道 CPSO-S、CPSO-H 很容易的就落入了區域的極小，而本論文比 PSO 表現優良的原因為此函數是一個欺騙函數，最佳解與次佳解之間分佈在極遠的兩邊，PSO 由於沒有行動價值函數使得它容易收斂到次佳解，因此本論文在這個函數最佳解找尋上表現最優。

在表 4.6 與圖 4.7 及式(4.4)中可以知道函數維度間並不相關，理論上對 CPSO-S、CPSO-H 這兩種演算法在尋找函數極值上較有利，由結果知本論文結果最佳

CPSO-S 次之，CPSO-H 再次之，PSO 表現最不好，觀察圖 4.7 之後可以發現這個函數具有十分多的區域極小值，因此可以知道 CPSO-S、CPSO-H 很容易的就落入了區域的極小，又雖然 CPSO-H 方法中包含了 PSO 但是由於本論文由於透過了 SARSA(λ)每一次的粒子群未至極速度更新時，都可以藉由行動價值函數的值來得知使用協同式粒子群是否會落入區域極小，所以能夠有最好的表現。

綜合以上所述，可以發現本篇論文提出的使用了 SARSA(λ)來決定 CPSO-S，PSO 切換時機的演算法在相同的執行時間下，與基本粒子群演算法(PSO)、協同式粒子群演算法(CPSO)、混合式粒子群演算法(CPSO-H)相比之下都有不錯的效果。因為它結合了 CPSO-S 具有比原始 PSO 演算法中還要多的群體多樣性，以及 PSO 不會落入假的最佳解(pseudominimizer)的特性，並能透過行動價值函數在適當的時候使用這兩樣方法，因此比起 CPSO-H 必須一定要 PSO、CPSO-S 交替的使用節省了許多不必要的時間浪費，所以在相同的執行時間下都有比 CPSO-H 要佳的表現。

第五章
結論

粒子群演算法是一種簡單有效的函數極值最佳化的演算法，雖然發表至今並沒有很長一段時間不過卻有許多廣泛的應用，其中以 PSO 以及其後被提出來的 CPSO-S 兩種方法最著名，兩種方法都有對方所沒有的優點與缺點，CPSO-S 由於切割了粒子群使得協同演化的想法可以改善 PSO 兩步向前一步向後的問題卻產生了只有 PSO 可以彌補的缺陷那就是落入假的最小值 (pseudominimizer) 的問題，為了處理這些問題必須將上述兩種方法融合起來而有了 CPSO-H，但是由於切割了粒子群使得粒子群多樣性變大同時也造成演算法在每一代的執行時間大幅度的增加，由第四章的實驗結果可以知道在相同的演算時間下，由於含有協同演化的演算法更新每一代粒子速度及位置的時間都太長，造成相同時間下演化代數比起 PSO 來的少,並且函數極值搜尋的效果也並沒有特別好。

本文即著眼於此利用 SARSA(λ)讓粒子群透過行動價值函數來採取 CPSO-S 或 PSO，在適當的狀態下發揮他們各自的特性進而避免掉為了產生不必要的粒子群多樣性拖緩了最佳解的搜尋。由於能夠適時的使用這兩種方法即可以在避免掉兩者的缺陷同時兼顧最佳解搜尋的效率。

未來希望能夠藉由函數近似方法以及特徵選取來構造價值函數(action-value function)使得價值函數能更精確的反應每個粒子群所在狀態的行動價值函數值 (SARSA(λ))，更加提升本篇論文在決策選擇時的精確度。

參考文獻

[1] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in Proc. of IEEE Int. Conf. on Neural Networks, vol. 4, pp. 1942-1948, 1995.

[2] Frans van den Bergh and A. P. Engelbrecht, "A Cooperative Approach to Particle Swarm Optimization," *IEEE Trans. Evol. Comput*. vol. 8, no. 3, pp.225-239, 2004.

[3] M. A. Potter and K. A. de Jong, "*A cooperative coevolutionary approach to function optimization*," in *The Third Parallel Problem Solving From Nature*. Berlin, Germany: Springer-Verlag, pp.249-257, 1994.

[4] M. A. Potter and K. A. de Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evolutionary Computation, vol.8, no. 1, pp.1-29, 2000.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning*. Cambridge, MA: MIT Press, 1998.

[6] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 834-836, 1983.

[7] C. J. Wakins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, U.K., 1989.

[8] C. J. Wakins and P. Dayan, "Technical note: Q-learning," Mach. Learn., vol. 8, pp. 279–292, 1992.

[9] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in

particle swarm optimization,” Proceedings of the IEEE Congress on Evolutionary Computation, pp.84-89, 2000.

[10] M. Clerc, “The swarm and queen: Toward a deterministic and adaptive particle swarm optimization,” Proceedings of the IEEE Congress on Evolution Computation, pp.1951-1957, 1999.

[11] M. Clerc and J. Kennedy, “The particle swarm: explosion, stability, and convergence in a multidimensional complex space,” *IEEE Trans. Evol. Comput.*, no. 6, pp58-73, 2002.

[12] N. Hansen and A. Ostermeier, “Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation,” Proceedings of the IEEE Conference on Evolutionary Computation, pp. 312-317, 1996.