採用失敗後跳躍的策略以改良爬山演算法

(An Improvement of Hill Climbing Algorithm with Jumping Strategy)

林伯陽 (Bo-Yang Lin)
金門技術學院 電資資訊所
yanghelp@gmail.com

陳鍾誠 (Chung-Chen Chen)
金門技術學院 資訊工程系 助理教授
ccckmit@gmail.com

摘要

爬山演算法 (Hill-Climbing，HC) 是廣為人知的一種基礎性演算法，具有簡單且快速的優點，但此方法落入區域最佳解後，就無法跳出，因而難以找到更好的解。有鑑於此，我們設計出一種隨失敗次數遞增的跳躍策略，這使得爬山演算法有機會脫離區域最佳解，以尋找更好的解答，此種方法我們稱為爬山跳躍演算法(Hill-Climbing with Jumping Strategies, HCJ)。

本文採用近來經常被使用到的三個非線性規劃(Nonlinear Programming)問題 – G1、G7、G9，來測試 HCJ 演算法的效果。結果顯示 HCJ 的表現在 G1、G9 上相當優秀，但在 G7 上則無法趨近最佳解，經過分析顯示，G7 可能是一個具有密集型深尖山谷的問題，這是 HCJ 目前所難以處理的問題，也是未來 HCJ 有待改進之處。

然而，HCJ 乃是單一粒子的演算法，與文獻上的粒子群方法相較之下簡單許多，與單一粒子的算法，像是 HC 與 SA 相比，HCJ 在效能與結果上有部分改進，這應該是因為 HCJ 跳躍策略所導致的改良，這代表 HCJ 的潛力值得進一步的探索。

關鍵詞：最佳化、爬山演算法、模擬退火法、粒子群演算法、非線性規劃、。

1. 前言

在最佳化問題上，目前常見的隨機型演算法大致可分為兩類，一類屬於單粒子型的演算法，像是『爬山演算法』與『模擬退火法』等，另一類屬於多粒子型的演算法，像是『遺傳演算法』、『粒子群演算法』與『蟻群演算法』等。

單粒子型的演算法通常較為簡單快速，但是對於具有多個山谷的的最小化問題，或多個山峰的最大化問題而言，很容易落入較差的區域，而多粒子型的演算法，由於粒子分散，相對而言較容易找到好的區域，但是其計算時間通常較長，實作也較為複雜。

『爬山演算法』[7] 是單粒子型算法中最簡單的一種，實作相當容易，且執行速度很快。因此，經常被用來作為各種最佳化演算法的比較基準 [1][3] 。但是，『爬山演算法』的缺點是容易落入到較差的區域，而且無力跳出該區域。

這是因為爬山演算法只尋找鄰近的點進行比較，而且不允許向較差的方向行走，這使得爬山演算法會落入山谷區(在最小化問題上)或山峰區 (在最大化問題上) 而無法跳出。這使得爬山演算法在較為複雜非線性規劃問題上，很容易陷落到較差的區域，難以找到全域最佳解。

模擬退火(Simulated annealing) 演算法 [8] 改進了爬山演算法的部分缺點，採用以溫度調控的機率特性，讓爬山演算法有機會跳脫較差的區域，因而找到更好的解。但是，當較好的區域距離目前所在地區較遠時，模擬退火演算法通常難以逆向爬升脫離較大的山谷。

於是，許多論文嘗試使用群體演算法，像是進化演算法(Evolutionary algorithm)[5]、粒子群優化 (Particle Swarm Optimization) [9] [10] 等演算法，以解決複雜型的非線性規畫問題。甚至，有些人更進一步採用『增廣式拉格朗日算法』 (Augmented Lagrangian Method，ALM) [11][4]，將非線性規畫問題轉化為對偶性的最大化-最小化 (Min-Max) 問題，以幫助這些方法能找到較好的解，像是 Co-PSO[11] 與 Co-Evolutionary[4] 等方法，就使用了 ALM 的轉化方式。然而，這種方法只適用在非線性規畫上，而無法使用於其他領域。

在本論文中，我們將不採用 ALM 的方法，而是直接對爬山演算法進行改良，於是，我們設計出了爬山跳躍演算法 HCJ，希望藉由跳躍機制讓爬山演算法更容易脫離山谷，找到更好的解。

爬山演算法、模擬退火法與 HCJ都是單一粒子型的演算法，在本論文當中，我們將主要著重在 HCJ 與其他兩者之間的比較，並且以近幾年來常被用來作為非線性規畫比較基準的三個問題，G1, G7, G9 進行實驗，以便分析 HCJ 的優缺點。

在後續小節中，我們將在第 2 節介紹『爬山演算法』第 3 節介紹『模擬退火法』、然後在第 4 節當中詳細說明如何在『爬山演算法』上加入跳躍策略，成為『爬山跳躍演算法』 (HCJ)。然後，在第 **錯誤! 找不到參照來源。** 節當中說明用來測試這些演算法的 G1、G7 與 G9 等非線性規畫問題，接著，在第 5 節當中，我們將以這三個問題測試前述三個單粒子型的演算法，並進行比較分析。然後再將 HCJ 的結果與文獻上群體演算法的結果進行比較，以觀察 HCJ 是否能與較複雜的群

體演算法相匹敵，最後在第 6 節當中進行結論。

由於HC、SA、GA、PSO 等數種演算法乃是在不同時期採用相異理念所發展出來的，因此，若採用原始的演算法會造成難以分析比較的情況，因此，在本文中，我們將以粒子 (Particle) 的角度，重新詮釋這些算法，以便呈現出算法之間的差異點，並有助於分析這些差異點所造成的效果。

在以下的演算法當中，我們會統一用粒子 p 代表目前解答，pb 代表到目前為止的最佳解，pn 代表被測試的鄰近解答。

2. 爬山演算法

『爬山演算法』是一種相當簡單的區域性搜尋演算法，由於其過程相當類似人類爬山時不斷向上爬的動作，因此稱為爬山演算法。爬山演算法可說是一種啟發式方法，其搜尋策略乃是不斷尋找周邊更好的解答，然後向更好的解答前進。換句話說，就是反覆的搜尋鄰居，一旦發現更好的鄰近點，就向該點前進，直到無法再改進為止。其演算法如圖 1 所示。

```
Algorithm Hill-Climbing(pi)
  p = pi // 設定粒子 p 為起始粒子 pi
  while not isEnd()
    pn = p.neighbor(step) //選擇粒子 p 的鄰居 pn
    if pn.fitness()>=p.fitness() //如果更好，就接受
      p = pn;
End Algorithm
```

圖 1、最大化版本的爬山演算法

爬山演算法的概念主要是針對最大化的問題而設想的。但是，如果是針對最小化問題，則該演算法必須做一點小小的修改。針對最小化的問題，比較適合使用能量函數的想法，其最佳化的目標，乃是尋找能量的最低點。在此種情況下，我們可以用能量函數 energy() 取代將圖 **1** 中的 fitness() 函數，將圖 **1** 的演算法稍作修改成為圖 **2** 的演算法。

```
Algorithm Hill-Climbing(pi)
  p = pi // 設定粒子 p 為起始粒子 pi
  while not isEnd()
    pn = p.neighbor(step) //選擇粒子 p 的鄰居 pn
    if pn.energy()<=p.energy() //能量更低，就接受
      p = pn;
End Algorithm
```

圖 2、最小化版本的爬山演算法

在後面的實驗中，我們用來測試的三個非線性規畫問題 G1、G7、G9 都是最小化問題。因此，我們將以圖 **2** 的最小化版本為主，並且採用能量的觀點，企圖找到最低的能量點。就像地心引力會吸引粒子向下掉落，直到谷底一樣，後續的演算法也都將採用此種思考模式，以便讓演算法呈現出一致的風貌。

首先，請比較上述兩個演算法，圖 **1** 的最大化版本與圖 **2** 的最小化版本是可以互相轉換的，只要將 energy() 設為 -1*fitness() 即可。這樣的轉換並不會影響到最後的結果，只是觀察的角度不同而已。

爬山演算法求解最小化問題的過程如圖 **3** 所示，爬山演算法會逐步的往區域最佳解的位置逼近，以尋找極小值。

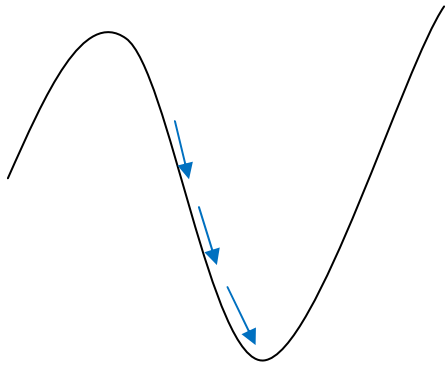


圖 3、以爬山演算法尋找最小值

由於爬山演算法只會的找尋它鄰近的解來做比較，一旦鄰近處沒有更好的解，爬山演算法就無法再改進解答了。在實作時，我們可以計算連續未改進的次數，如果連續失敗太多次時，就視為已經到谷底了，此時爬山演算法就可以結束了。

```
Algorithm Hill-Climbing(pi)
  failCount = 0;
  p = pi // 設定粒子 p 為起始粒子 pi
  // 當失敗次數超過上限 (failMax) 時，就結束
  while failCount < failMax
    pn = p.neighbor() //選擇粒子 p 的鄰居 pn
    if pn.energy()<=p.energy() //能量更低，就接受
      p = pn
    if pn.energy()<p.energy() //找到更好的
      failCount = 0 // 將失敗次數歸零
    else
      failCount ++// 否則，將失敗次數加一
    end if
End Algorithm
```

圖 4、利用連續失敗次數判斷是否應該結束

圖 **4** 的爬山演算法中的鄰居選擇函數 neighbor(p)，通常有兩種實作方式，第一種是使用系統式的搜尋法，第二種是採用隨機式的挑選法。通常，針對離散性的問題，只要鄰居的數量不多，可以採用系統式的搜尋法。而針對鄰居數量太多時，會有難以完全列舉的狀況，此時適合採用隨機式的挑選法。這種採用隨機是挑選法的爬山演算法，稱為『隨機爬山演算法』(Stochastic Hill-Climbing Algorithm)。

對於有多個山谷的問題，爬山演算法很容易陷入到較差的區域當中，而無法找到更好的區域，圖

5 顯示了此種情況，此時爬山演算法會被困在較差的區域，無法跳脫。

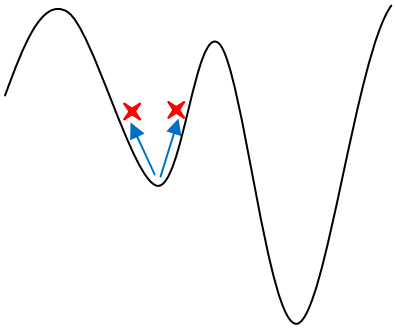


圖 5、爬山演算法無法跳出區域最佳解

3. 模擬退火演算法

模擬退火演算法可視為爬山演算法的一種改良。其改進的方式是利用一個隨著溫度而調整的機率函數 exp(-ΔE/k T)，以決定是否要接受新的解(pn)，其演算法如圖 6 所示。

```
Algorithm SimulatedAnnealing(pi, Ti)
  p = pi // 設定粒子 p 為起始粒子 pi
  T = Ti // 設定初始溫度
  while not isEnd()
    pn = neighbor(p) //選擇粒子 p 的鄰居 pn
    ΔE = pn.energy() – p.energy()
    random = 0 到 1 之間的隨機實數
    if random < exp(-ΔE / k T)
      p = pn
    T = T * ratio;
End Algorithm
```

圖 6、模擬退火演算法

假如新粒子 pn 比目前的粒子 p 好(能量更低)，那麼，該機率函數 exp(-ΔE/k T) 會大於 1，因此 pn 一定會被接受。若 pn 的能量 p 高，則接受的機率將由能量差 ΔE 與溫度 T 而定，其中的 k 乃是一個常數，用來調整 exp 函數的形狀，k 值愈大，則該函數愈平緩，反之，則越陡峭。

由於模擬退火演算法允許向較差的方向移動(在最小化問題當中就是向上爬)，因此，有機會跳出區域最佳解而找到更好的解。但是，對於較深、較大的山谷而言，模擬退火演算法必須靠著一連串的向上跳躍，才能跳向另一個山谷。但是，一連串向同一個方向的跳躍，其機率是通常是非常低的，尤其是在溫度越低的時候，更為困難。因此，模擬退火演算法在實務上很難跳越較大的山谷，以找到更好的解。

4. 爬山跳躍演算法

為了讓爬山演算法能跳脫較大的山谷，我們設計了爬山跳躍演算法 HCJ (Hill-Climbing with Jumping Strategy)，其演算法如圖 7 所示。

```
Algorithm HCJ(pi)
  p = pi // 設定粒子 p 為起始粒子 pi
  while not isEnd()
    //根據跳躍距離 step 選擇粒子 p 的鄰居 pn
    pn = p.neighbor(step)
    if pn.energy()<=p.energy() //能量更低，就接受
      p = pn
    step = adjustStep() // 調整跳躍範圍
End Algorithm
```

圖 7、爬山跳躍演算法

HCJ 與傳統爬山演算法的不同點在於 HCJ 的跳躍腳步 adjustStep()函數，該函數會使得跳躍腳步 step 隨著失敗的次數而增加。於是，當目前粒子 p 位於谷底時，會造成連續性的跳躍失敗，此時，HCJ 會逐步加大跳躍的範圍，如圖 8 所示。

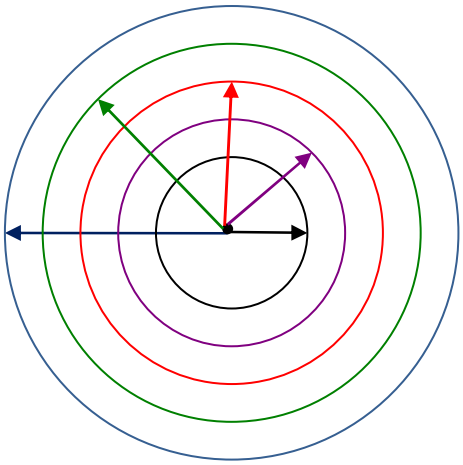


圖 8、逐次加大搜索範圍

藉由加大跳躍的範圍，HCJ 有機會能夠跳脫目前的山谷，向更低的山谷前進。我們希望藉由這樣的設計，讓 HCJ 有機會找到更好的解，甚至找到全域最佳解。

圖 **9** 顯示了 HCJ 很容易找到最佳解的一種情況。因為 HCJ 在谷底時會逐漸加大跳躍範圍，於是，當 HCJ 的粒子 p 位於圖 9 左右兩邊的山谷時，很容易就能跳到旁邊更深的山谷中，因而找到更好的解。然後，再度透過相同的程序，不斷向更好的解邁進，這可能會增加找到全域最佳解的機會。

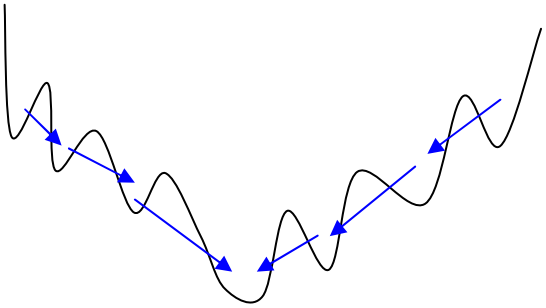


圖 9、HCJ 透過漸進式的跳躍尋找最佳解

但美中不足的是，在某些狀態下爬山跳躍演算

法並不容易發現最佳解，像是圖 10 的山谷區域又尖又小，以爬山跳躍演算法的跳躍能力來看，非常容易就跳過頭，忽略了虛線底下區域的存在，因而錯過了最佳解。

圖 **11** 則顯示了另一種難以找到最佳解的狀況。雖然該最佳解所在的區域並不小，但是當最佳化問題是高維度的情況時，由於隨機跳躍在高維空間中的自由度太大，HCJ 很容易像不正確的方向探索，這使得跳入右邊最佳解區域的機率相對變小，因而容易造成漏失的情況。

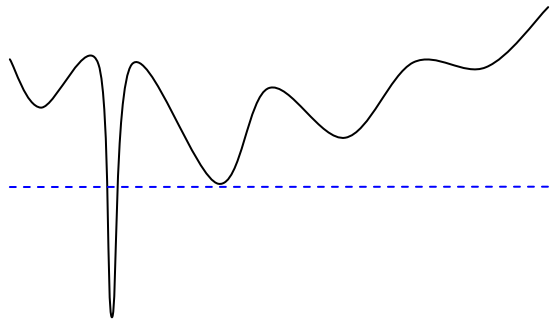


圖 10、HCJ 容易漏失的狀況-最佳值區域太小

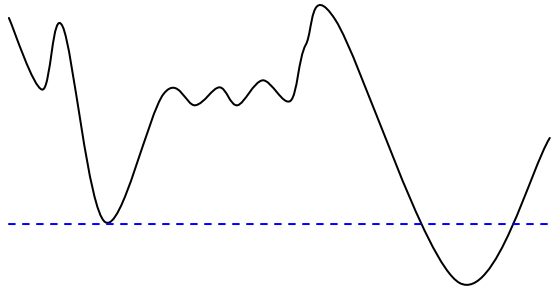


圖 11、HCJ 容易漏失的狀況 - 最佳值區域太遠

5. 實驗方法

5.1 實驗問題

為了測試 HCJ 的效能，我們找了近年來較常被非線性規畫領域用作測試基準的三個最小化問題 G1、G7、G9 來進行實驗 [2]、[3]、[4]、[5]。

第一個問題 G1 包含 13 個實數值的變數，其題目如下：

$$f(x) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

這些變數需要遵守以下條件

$$2x_1 + 2x_2 + x_{10} + x_{11} \le 10$$
$$2x_1 + 2x_3 + x_{10} + x_{12} \le 10$$
$$2x_2 + 2x_3 + x_{11} + x_{12} \le 10$$
$$-8x_1 + x_{10} \le 0$$
$$-8x_2 + x_{11} \le 0$$
$$-8x_3 + x_{12} \le 0$$
$$-2x_4 - x_5 + x_{10} \le 0$$
$$-2x_6 - x_7 + x_{11} \le 0$$
$$-2x_8 - x_9 + x_{12} \le 0$$

而且其變數值不可脫離以下範圍

$$0 \le x_i \le 1, \qquad i = 1,\ldots,9$$
$$0 \le x_i \le 100, \quad i = 10,11,12$$
$$0 \le x_i \le 1, \qquad i = 13$$

此題的全域最佳解已知是

$$x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$$
$$f(x^*) = -15$$

第二個問題 G7 包含 10 個實數值的變數，其題目如下：

$$f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)$$
$$+ 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^{22}$$
$$+ 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

需要遵守以下條件

$$105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \ge 0$$
$$-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \ge 0$$
$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \ge 0$$
$$-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \ge 0$$
$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \ge 0$$
$$-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \ge 0$$
$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \ge 0$$
$$-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \ge 0$$

而且其變數值不可脫離以下範圍

$$-10 \le x_i \le 10, \qquad i = 1,\ldots,10$$

此題的全域最佳解已知是

$$x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548,$$
$$1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$$
$$f(x^*) = 24.3062091$$

第三題 G9 包含 7 個實數值的變數，其題目如下：

$$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

需要遵守以下條件

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0$$

$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0$$

$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0$$

$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$$

而且 x 的值不可脫離以下範圍

$$-10 \leq x_i \leq 10, \qquad i = 1, \ldots, 7$$

此題的全域最佳解已知是

x^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)

$f(x^*) = 680.6300573$

5.2 實驗方法

在本實驗當中，我們採用下列實務上的作法 (1) 在能量函數上，以懲罰函數的方式，避免演算法最後仍在不合法的範圍。(2) 在選取鄰居函數 p.neighbor() 的實作上，以『隨機挑選法』選取鄰居。以下我們將詳細說明本實驗在這兩點上採用的作法，以及其設計原理。

(1) 在能量函數上，我們採用如圖 **12** 的演算法，其中『penalty *超出範圍的差距總和』乃是懲罰函數，我們在實驗中設定懲罰系數 penalty 為一百萬。

```
Algorithm energy(x[1..n])
  return f(x) + penalty *超出範圍的差距總和
End Algorithm
```

圖 12、本實驗中能量函數 energy() 的設計

(2) 在隨機挑選鄰居時，我們採用圖 14 的 neighbor() 演算法，而非圖 13 的演算法，其原因說明如下。

在多變數的非線性規畫問題上，採用隨機式挑選法選取鄰居時，可能會導致一些問題，主要的是因為變數通常具有範圍限制，因此，當爬山演算法位於邊界時，很容易就超出範圍，這很可能會導致太多的失敗，而讓爬山演算法提早結束。

假如我們的選擇鄰居函數如圖 13 所示，那麼，當最佳值位於邊界區域時，所產生的鄰居很容易落到邊界以外的區域，尤其當變數的個數很多，而且大部分的變數都已經調到邊界時，此時隨機選擇的鄰居落在不合法區域的機會很大，而落在合法區域的機率會很低，這會造成爬山演算法失敗太多次而結束。

更明確的說，如果有 x[1..n] 等 n 個變數，其中除了 x[1] 之外，其他的 x[2..n] 這 n-1 個變數都已經到達邊界，此時，圖 13 的演算法執行時，只有 $1/2^n$ 的機率可以進入合法範圍內，因此，在 n 較大時，很容易導致爬山演算法因為失敗太多次而結束。

```
Algorithm neighbor(x[1..n], step)
  for i= 1 to n
    x[i]=x[i]+(-step 到 step 之間的隨機實數)
End Algorithm
```

圖 13、在邊界區域容易失敗的鄰居選擇函數

要克服邊界區域所產生的失敗情況，可以改用圖 14 的函數，這個函數首先以隨機的方式決定應調整的變數個數，然後再隨機挑選出 x 當中的 k 個進行調整的動作，調整的範圍大小由 step 決定。

```
Algorithm neighbor(x[1..n], step)
  k = 1 到 n 之間的隨機整數
  s[1..k] = 隨機選取 k 個介於 1 到 n 的整數
  for i=1 to k
    x[s[i]]=x[s[i]]+(-step 到 step 之間的隨機實數)
End Algorithm
```

圖 14、本實驗中鄰居函數 neighbor() 的設計

使用圖 14 的演算法，選出的 k 值為 1 的機率是 1/n，其中只選到第 1 項被調整，也就是 s[1]=1 的機率為 $1/n * 1/n = 1/n^2$。

當這種鄰居選擇法使用在爬山演算法上時，由於調整的步伐 step 很小 (假設在這麼小的範圍中不會有兩次以上的轉折)，那麼，由於 x[1] 可以向兩個方向調整，因此，將有一半的機率會向正確方向調整。於是，整體的成功機率將會是 $1/2n^2$。在 n 數值大的時候，圖 14 所選鄰居不超出邊界的機率 $1/2n^2$ 比圖 13 的 $1/2^n$ 會大的多，因此，在邊界時，圖 14 的鄰居選擇法有較大的機會向邊界內調整。

另外，在跳躍範圍的設定上，『爬山演算法』與『模擬退火法』均採用固定跳躍範圍 step=0.001 的方式，而 HCJ 的跳躍範圍的設定則需要採用隨失敗次數增長的跳躍策略。目前，我們設計了兩個版本的跳躍策略以實作 adjustStep() 函數，其演算法如圖 15 與圖 16 所示。

```
Algorithm adjustStep()
  step = stepMin + (stepMax × failCount) / failMax
  return step
End Algorithm
```

圖 15、HCJ1 中所用的步伐調整函數

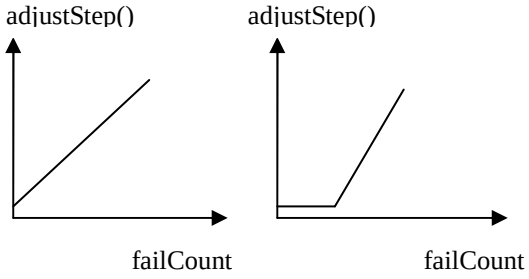
```
Algorithm adjustStep()
  step = ((1 + r) × failCount − r × failMax) × stepMax / failMax
  return max(stepMin,step)
End Algorithm
```

圖 16、HCJ2 中所用的步伐調整函數

在這兩個 adjustStep() 函數中，failCount 乃是失敗次數，failMax 乃是失敗次數的上限，而 stepMin 是步伐下限，stepMax 則是步伐上限，而圖 16 當中的參數 r，則是傾斜度參數。

圖 15 中顯示了較簡單的步伐調整函數，但是該函數可能會遭遇到跳躍步伐隨失敗次數 failCount 快速增長的問題，該函數的形狀如圖 17 (a) 所示。

圖 16 的顯示了較複雜的步伐調整函數，由於該函數的形狀如圖 17 (b) 所示，因此避免了 failCount 成長過快的問題，這種作法讓 HCJ 在緊鄰的區域進行較多次的搜尋，假如還一直找不到更好的解，才開始調大步伐大小。



(a) 圖 15 的函數　　(b) 圖 16 的函數

圖 17<圖 15、圖 16>的步伐調整函數之形狀

在 5.3 的實驗中，我們分別對這兩個版本進行測試，其中 stepMin 設定為 0.001, stepMax 設定為 10.0，傾斜度參數 r 設定為 0.2。

在 5.3 的實驗中，四種被測試的演算法都採用**圖 12** 的能量函數與**圖 14** 的鄰居選擇法，以及圖四的結束判定策略，其失敗上限 failMax 被設定為十萬次。

5.3 實驗結果

表 1 列出與 HCJ 同樣是以單一粒子進行運算的爬山演算法及模擬退火法，執行 10 次實驗運算後的結果。

表 1 中的 B 為 10 次實驗結果中最好的值 (best)，M 為 10 次實驗結果的平均值(mean)，W 為 10 次實驗結果中最差的值 (worst)，而 T 則是演算法的平均執行次數 (mean times)。

表 1、HCJ 與單一粒子演算法之比較

測試問題		G1	G7	G9
最佳解答		-15.000	24.306	680.630
爬山 HC	B	-13.82807	25.75052	680.64980
	M	-12.77340	26.02496	680.66446
	W	-12.65618	26.26909	680.69226
	T	18740364	17569175	7484751
退火 SA	B	-13.82810	25.94367	680.64347
	M	-12.65614	26.21404	680.65593
	W	-12.77339	26.03041	680.67990
	T	16363114	18745291	8837745
爬山跳躍 HCJ1	B	-14.99410	26.33808	680.77165
	M	-14.98262	27.42273	681.36563
	W	-14.96480	26.88230	682.31886
	T	2053210	658076	44650
爬山跳躍 HCJ2	B	-14.99996	26.06275	680.77372
	M	-14.99987	26.51113	680.80775
	W	-14.99969	27.51367	680.84613
	T	3949977	4507608	2453098

表 1 的結果顯示，爬山跳躍演算法 HCJ1 與 HCJ2 在 G1 上明顯比爬山演算法 (HC) 和模擬退火法 (SA) 好，因為 HCJ1, HCJ2 幾乎都非常接近最佳解，分析其結果發現，所找到的解與最佳解相同，HCJ1 在小數點以下第 2 位以後有所差異，而 HCJ2 則在小數點以下第 4 位以後有所差異，這是因為 HCJ2 在步伐調整函數 adjustStep() 前段為水平的影響。

在 G9 問題上，幾乎所有方法都逼近了最佳解，其差異不大，只有 HCJ1 表現較差，這應該是因為 HCJ1 步伐調整函數調整太快，以至較容易在邊界上出現失敗次數過多的情況所導致的。

在 G7 問題上，HCJ1、HCJ2 則比 HC 和 SA 稍差，在表 2 中我們分析 HCJ2、HC、SA 在 G7 上最好一次的解，發現 G7 的標準解答與所有方法找出來的最優解答之間，都具有比 0.001 (step) 大上許多的差異，這應該代表 G7 在最佳解的附近，有許多區域最佳解，而且這些最佳解在各個變數上都不相同，這也是 G7 問題為何會如此困難的原因。

表 2、各個方法的最優解與 G7 解答之間的差異

G7	HCJ2		HC		SA	
解答	最優	差異	最優	差異	最優	差異
2.172	2.191	-0.019	2.197	-0.025	2.201	-0.029
2.364	2.689	-0.326	2.649	-0.286	2.670	-0.306
8.774	8.203	0.571	8.249	0.525	8.198	0.576
5.096	5.186	-0.090	5.190	-0.094	5.203	-0.107
0.991	1.020	-0.029	1.035	-0.044	1.038	-0.047
1.431	1.375	0.056	1.419	0.012	1.420	0.010
1.322	1.151	0.171	1.176	0.146	1.167	0.155
9.829	9.582	0.246	9.611	0.218	9.594	0.234
8.280	8.299	-0.019	8.152	0.128	8.166	0.114
8.376	8.674	-0.298	8.242	0.134	8.283	0.093

從表 1 與表 2 的實驗數據，我們可以看出下列結果。對於 G1 與 G9 兩個問題而言，HCJ2 幾乎都可以找到最佳解，這應該是因為 HCJ2 對於解答分散，但是山谷較大的情況處理得很好，但是對於山谷又深又尖的 G7 而言，就無能為力了，更困難的是，表 2 的數據顯示了 G7 的解答與 HCJ2、HC、SA 等各方法的最優解之間有明顯的差異，這很可能使得 HCJ2 的鄰居選擇函數，難以同時都向解答的方向改變，並且跳出適當的距離。這也是 G7 問題為何如此困難的原因。這個線索 HCJ 的進一步

改良，提供了重要的依據。

表 3 列出了 HCJ2 與使用多粒子演算法之間的比較；其中 Co-PSO 為 Yuhui 等人發表的協同粒子群演算法[11]，CEALM 為 Tahk 發表的協同演化算法[4]， Dynamic Penalties、Annealing Penalties、superiority of feasible points 及 Death Penalty 為 Michalewicz 等人於廣泛比較各方法優劣的論文中，所進行的實驗結果[12]。

表 3、HCJ 與群體演算法之比較

測試問題		G1	G7	G9
最佳解答		-15.000	24.306	680.630
HCJ2	B	-15.000	26.063	680.774
	M	-15.000	26.511	680.808
	W	-15.000	27.514	680.846
Co-PSO	40	-13.964	24.700	680.873
	80	-14.037	24.543	680.709
	120	-14.160	24.478	680.668
CEALM	B	-15.000	24.306	680.630
	M	-15.000	24.306	680.630
	W	-15.000	24.308	680.630
Dynamic Penalties	B	-15.000	25.486	680.787
	M	-15.000	26.905	681.111
	W	-14.999	42.358	682.798
Annealing Penalties	B	-15.000	18.917(X)	680.642
	M	-15.000	24.418	680.718
	W	-15.000	44.302	680.955
superiority of feasible points	B	-15.000	17.388(X)	680.805
	M	-15.000	22.932(X)	682.682
	w	-14.999	48.866	685.738
Death Penalty	B	-15.000	25.653	680.847
	M	-14.999	27.116	681.826
	W	-13.616	32.477	689.417

註：表中打(X)的地方代表該解答落在不合法區域

在表 **3** 中，由於 Co-PSO 之作者並未於論文上提供最佳解及最差解，而是提供 Co-PSO 分為 40 群、80 群、及 120 群的數據，故表 **3** 中無法列出 B 與 W 的值，而改以 40, 80, 120 代替。

從表 **3** 的結果中，我們看到 Co-PSO 與 CEALM 等採用 Augmented Lagrangian Method (ALM) 的方法，其優勢乃在像 G7 這樣具有密集性深谷的問題時，仍然能找到最佳解，而其他沒有採用 ALM 者，則通常無法找到。

另外，由於我們實驗中的 HC、HCJ1、HCJ2、SA 等方法，採用了較類似 Death Penalty 的強烈懲罰函數，因此，解答最後全都落入合法區域中。

6. 結語

進行實驗測試後發現，HCJ2 在 G1 及 G9 兩個問題上表現優異，並且在 G1 上明顯比 HC 和 SA 好。但是在 G7 上則還有改進空間，結果甚至比 HC 和 SA 差。據我們的分析結果顯示，這可能是因為 HCJ2 在『密集的深尖形山谷群』中，難以跳入最佳區域的緣故。

另外，由表 2 的結果顯示，由於 HCJ2 所找到的最優解在每一個變數上都與 G7 的解答有明顯差異，這很可能使得 HCJ2 的鄰居選擇函數，難以同時都向解答的方向改變，並且跳出適當的距離。

經過這些實驗，可以給與後續的研究者一些方向。因為，即便是使用像單一粒子的演算法，也能在這些被視為困難的非線性規畫問題上，具有良好的表現，似乎不一定要靠群體演算法才能達到良好的效果。

G1 的實驗顯示了 HCJ 的潛力值得探索，但是，我們將考慮如何讓 HCJ 克服像 G7 這種具有密集山谷的問題，並且設法讓 HCJ 能在此種情況下仍然能向更深的谷區邁進，採用統計或轉換的方式，增進 HCJ 演算法的能力。

我們在未來將以更多樣化，更為實務性的測試問題，探索 HCJ 的能力，並進一步改進單一粒子最佳化演算法的能力。因為，我們認為，演算法的簡單性，對其應用面而言是很重要的，甚至，對其理論分析也相當重要，有助於指出進一步的改良方向。

參考文獻

[1] A. Juels, A., M. Watenberg, “Stochastic Hill-Climbing as a Baseline Method for Evaluating Genetic Algorithms” , Tech. Report, University of California at Berkeley, 1994.

[2] E. Aarts, and J.Korst, Simulated Annealing and Boltzmann Machines. John Wiley & Sons, 1989.

[3]M. Mitchell, J. H. Holland, S. Forrest, “When Will a Genetic Algorithm Outperform Hill Climbing?” , Advances in Neural Information Processing Systems, 6, Cowan, J. D., Teasuro, G., Alspector, J., (Eds.), Morgan Kaufmann, 1994.

[4]M.J.Tahk and B.C.Sun, "Co-evolutionary augmented Lagrangian methods for constrained optimization," IEEE Transactions on Evolutionary Computation, Vol.4, No.2, pp.114-124, July 2000.

[5] M. Zbigniew、S. Marc, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary. Computation*, vol. 4, pp. 1-32, 1996.

[6] R. C. Eberhart、Y. Shi, "Computational Intelligence: Concepts to Implementations," 1 ed: Morgan Kaufmann, 2003, pp. 125-133.

[7] S.J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Second Edition, pp. 111-114, by Prentice Hall, 2003, ISBN 0-13-790395-2.

[8] S.Kirkpatrick, C.D.Gelatt, M.P.Vecchi, “Optimization by simulated annealing”, Science Vol 220, pp. 671-80, May, 1983.

[9] Thomas Weise. Global Optimization Algorithms – Theory and Application.Thomas Weise, October 4, 2007 edition, July 2007. Online available at

http://www.it-weise.de/ (version October 4, 2007)

[10] X Hu, R Eberhart, Solving constrained nonlinear optimization problems with particle swarm optimization, - Proceedings of the sixth world multiconference on systemics, 2002.

[11] Yuhui Shi , R. A. Krohling, Co-evolutionary particle swarm optimization to solve min-max problems, Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress, p.1682-1687, May 12-17, 2002

[12] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," Evolutionary Computation, vol. 4, no. 1, pp. 1-32, 1996.