台灣地理資訊學刊　第四期：13-26 (2006)
JOURNAL OF TAIWAN GEOGRAPHIC INFORMATION SCIENCE (4) : 13-26 (2006)

Cantor 空間填充曲線之演算法探討

On Cantor's Space-Filling Curve

李佩芝[*]　黃中偉[**]　史天元[***]

Pei-Zhi Lee　Chong-Wei Huang　Tian-Yuan Shih

摘　要

SFC (Space-Filling Curve) 爲於二維空間中，對網格內的節點進行一維排序，即二維空間一維編碼法。一般資料庫內的資料儲存皆爲一維形式的安排，若藉由二維方式適當處理，當可提昇資料存取的效率。

SFC 有許多種，依性質可分爲「遞迴式」與「非遞迴式」兩大類。迄今已有不少文獻探討其路徑編碼的演算法，但大多敘述複雜且不易明瞭，少見細膩的說明步驟。於九種 SFC 的近鄰性 (proximity) 研究顯示，以非遞迴式的 Cantor SFC 爲最優，故選取該 SFC 加以闡述。

關於 SFC 的評估，本研究採用四種指標：路徑全長、單位長度種類、平均四強鄰距離和與平均四弱鄰距離和，各評估指標並以多項式函數形式表示，引數爲網格大小 (c)，其係數則以多項式迴歸求得之，其中部分的係數爲正合 (exact)。

至於 SFC 的演算法，其時間的複雜度與設計的方式有關。分析結果顯示，捷徑法的複雜度爲 O (1)；逐步法則爲 O (c^2)。

關鍵字：空間填充曲線、演算法、複雜度

*　國立嘉義大學土木與水資源工程學系碩士
Master, Department of Civil and Water Resources Engineering, National Chiayi University.

**　國立嘉義大學土木與水資源工程學系副教授
Associate Professor, Department of Civil and Water Resources Engineering, National Chiayi University.

***　國立交通大學土木工程學系教授
Professor, Department of Civil Engineering, National Chiao-Tung University.

Abstract

A space-filling curve is a way of mapping the two-dimensional space into one-dimensional. Because of data arrangement in the database being in one-dimensional form, using an appropriate SFC may improve the query efficiency.

There are a lot of SFCs in the literatures. Each SFC has its advantages and disadvantages. An evaluation index of a comprehensive study on some typical SFCs has shown that Cantor's SFC has the best proximity. This study focuses on the encoding algorithm and analysis of Cantor's SFC.

An evaluation index has been made on Cantor's SFC, namely "average sum of eight nearest neighbors distance". The index is concluded as a linear function. The argument is the size of domain (c in short) while c is up to 4096. Both the coefficients are solved by means of regression analysis.

As for the time complexity of algorithms for Cantor's SFC, it is obviously depends on the designing scheme. Results show that the complexities of the short-cut approach is O (1) while the stepwise approach is O (c^2) .

Keywords: space-filling curve, algorithm, complexity.

前　言

十九世紀末，Peano 發現 Space-Filling Curve (Sagan, 1994)，簡稱 SFC，Laurini and Thompson (1992) 稱其爲空間路徑 (path through space)。SFC (Space Filling Curve) 之譯名有「空間塡充曲線」(唐中實等，2004) 與「滿佈空間之曲線」(史天元，2005)。其應用方面頗爲多元，其中在資料庫管理部分，由於資料在計算機內係以一維方式儲存，若選擇二維空間以上之一維編碼法，將有助於資料搜尋的效率 (史天元，2005)。

SFC 之定義爲於二維空間中，對網格內的節點 (node) 進行一維排序 (Laurini and Thompson, 1992; Rigaux *et al.*, 2002)，即二維空間一維編碼法。其特性猶如以針線穿過網格內所有的節點，且每個節點恰僅經過一次。SFC 的種類繁多，依路徑走向的自我相似性 (self-similarity) －以 2*2 的網格爲基礎所延伸，可分爲「遞迴式 SFC」(Recursive_SFC，簡稱 RSFC)，如圖 1 (e) ~ (i)；「非遞迴式 SFC」(non-Recursive_SFC，簡稱 NRSFC)，如圖 1 (a) ~ (d) 所示 (Breinholt and Schierz, 1998；Mokbel and Aref, 2001/2003)。本文採用 order 表示網格大小 (size of domain)，並命名爲變化幅度 (variation state)。以網格大小 c 的擴張性 (extension) 觀之，其擴張速率爲 2^{order}。

演算法之定義爲求解計算問題的過程 (Manber, 1989)。Cormen *et al.*（1991) 則將之定義爲求解特定數學問題的工具；或給定輸入値 (input) 藉由一系列計算程序求得輸出値 (output)。而時間複雜度 (time complexity) 則用以表示當演算法施行 (performance) 時的計算時間與處理規模 (input size) 間的函數關係。迄今已有不少文獻探討其路徑編碼的演算法 (Butz, 1971; Mark and Goodchild, 1986; Breinholt and Schierz, 1998)，其內容少見細膩的說明步驟。故本研究針對兩種編碼方式：一維空間二維編碼法與二維空間一維編碼法，皆輔以細緻的計算步驟，其中輸入値可爲節點序號 N 或 x、y 座標，而輸出値則爲 x、y 座標或 N。

表 1 爲李佩芝 (2005) 評估九種 SFCs 的近鄰性 (proximity) 成果：當網格大小爲 c * c，c = 4096 (c = 2^{order}, order = 12) 時，各 SFC 的平均八鄰距離和顯示，以 Cantor SFC 的爲最佳。故本研究針對 Cantor SFC 的演算法進行探討，尤其著重於其複雜度的設計與分析。

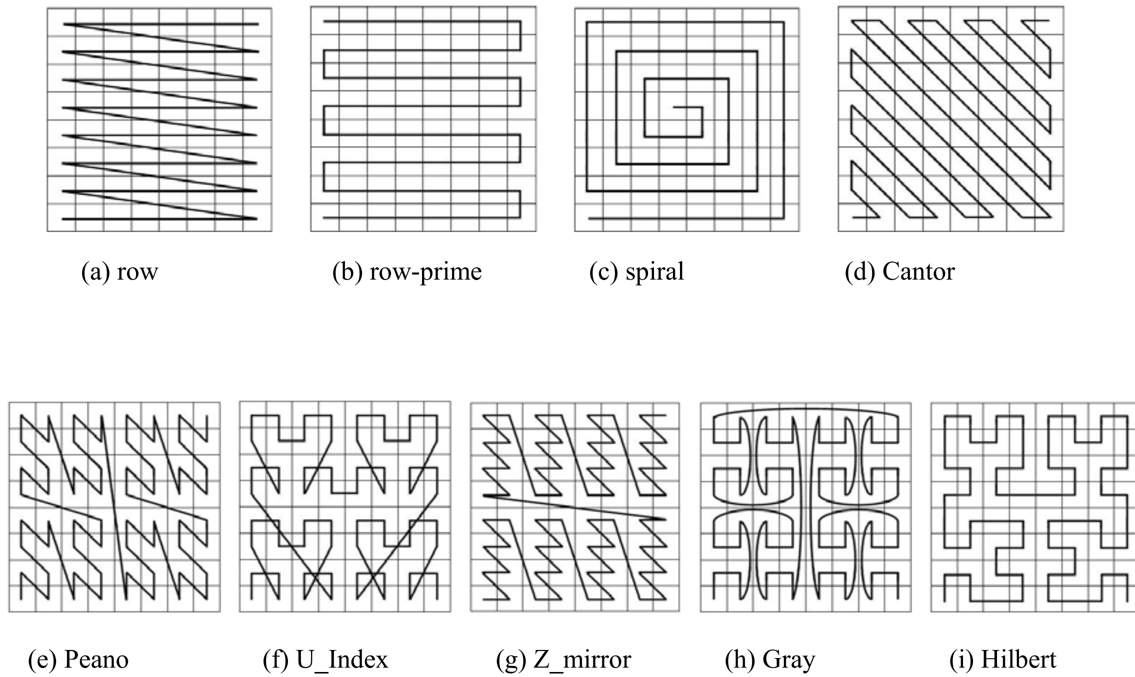

圖 1　常見之 SFCs (order = 3, c = 8)

Cantor SFC 又稱爲 Cantor_diagonal order (Mark and Goodchild, 1986; Samet, 1990；Laurini and Thompson, 1992) 、 diagonal SFC (Mokbel and Aref, 2001) 、 Cantor_original 排序 (唐中實等，2004)、對角線排列 (史天元，2005)，如圖 2。其節點排序，以左下角由 0、座標 (1,1) 開始起算，如圖 2 (c) 所示。

關於 Cantor SFC 的評估，採用四種指標進行探討，分別爲路徑全長、單位長度種類、平均四強鄰距離和與平均四弱鄰距離和。另將各評估指標以多項式函數形式表示，弧數爲 c，其係數則利用迴歸分析求得。四強鄰乃指網格式分布邊相鄰之四個節點，一般又稱爲第一類鄰近 (Type I neighbor)。四弱鄰乃指網格式分布角相鄰之四個節點，一般又稱爲第二類鄰近 (Type II neighbor)。平均四強鄰距離和乃將所測試網格所有節點四強鄰之 N 值與中央節點 N 值之差，取其總和，再求平均。平均四弱鄰距離和之計算方法亦同。

至於演算法的分析，依本研究設計 Cantor SFC 的兩種型態演算法進行複雜度分析，藉以探討演算過程的效率問題。

表 1　九種 SFCs 評估指標計算結果
(order = 12, c = 4096)

指標 / SFC	路徑全長	單位長度種類	平均八鄰距離和
row	33542145.50	2	24578.00
row-prime	16777215.00	1	24578.00
spiral	16777215.00	1	65498.01
Cantor	23723172.58	2	21852.66
Peano	28050569.80	13	24558.20
U-Index	24131784.22	12	28648.08
Z-mirror	28050569.80	13	24561.10
Gray	33542145.00	12	36830.96
Hilbert	16777215.00	1	29817.51

(a) order = 2, c = 4

(b) order = 3, c = 8

35	36	48	49	57	58	62	63
21	34	37	47	50	56	59	61
20	22	33	38	46	51	55	60
10	19	23	32	39	45	52	54
9	11	18	24	31	40	44	53
3	8	12	17	25	30	41	43
2	4	7	13	16	26	29	42
0	1	5	6	14	15	27	28

(c) order = 3 之排序

圖 2　Cantor SFC

Cantor SFC 之演算法

本研究依 Cantor SFC 之路徑設計兩種形態的演算法：二維空間一維編碼法與一維空間二維編碼法。設計時，可分爲捷徑方式 (short-cut approach) 與逐步方式 (stepwise approach)，皆以細緻的演算步驟說明表示之。

```
Algorithm Cantor_ XY _to_N (x, y, c, N) ;  {short-cut approach}  ……….Alg. 1
Input :
        x, y：location of a given node
        c：size of domain (c*c)
Output :  N：the serial number of the specific node
begin
   k ← x + y - 1          {the node is located at the k-th diagonal from the origin}
   if  k > c  then             {the node is located at the upper triangular part}
     x ← c – x + 1          {redefine the nodes's location in a new coord. system}
     y ← c – y + 1          {in terms of the origin being as at the upper right cell}
     k ← x + y - 1                 {the k-th diagonal counted from the upper right}
     qty1 ← k* (k-1) div 2 {quantity of nodes in the upper right of the k-th diagonal}
     nodes ← c * c                              {quantity of nodes in the domain}
     if  odd (k)  then
           N ← nodes – qty1 – x                                          {done}
     else
           N ← nodes – qty1 – y                                          {done}
   else    {the node is located at the main diagonal or at the lower triangular part}
     qty1 ← k* (k-1) div 2 {quantity of nodes in the lower left of the k-th diagonal}
     if  odd (k)  then
           N ← qty2 + x -1                                               {done}
     else
           N ← qty2 + y -1                                               {done}
end;
```

(一) 二維空間一維編碼法

捷徑方式的演算法設計如 Alg. 1 所示，首先由 x、y 座標計算節點座落之對角線數 k (k 從左下角起算)，再將網格以主對角線爲界 (k = c)，分成上三角、下三角 (包含 k=c) 二部分進行求解。在上三角部分，以右上角爲原點，重新計算節點 x、y 座標與對角線數 k；下三角部分，則不需重新計算。此二部分皆藉由 k (k-1) ***div*** 2 之計算，可得第 k-1 條對角線內所涵蓋的節點數，最後判斷 k 的奇偶數，計算其與 x、y 之關係求解 N。

逐步方式的演算法設計如 Alg. 2 所示，首先由原點 (N = 0) 座標爲 (1,1) 開始，隨 x、y 值的變化，逐步計算所對應的 N 值。藉由三個布林參數 (Boolean Parameter) 控制路徑沿 x 軸 (Alg. 3)、y 軸 (Alg. 4) 及對角線方向 (Alg. 5) 之行走步法。

```
Algorithm Cantor_XY_to_N (x, y, c, N) ;  {stepwise approach}  ................  Alg. 2
Input :
        x, y：location of a given node
        c：size of domain (c*c)
Output :
        N：the serial number of the specific node
begin
  N ← 0                                                  {N is initialized as 0}
  i ← 1 ; j ← 1                           {the node's location is initialized as (1,1) }
  X_move ← .true.       {a Boolean variable which controls the moving direction}
  Dia_move ← .false.            {the moving direction is horizontally initialized}
  Dia_up ← .true.              {while Dia_move, it controls upward or downward}
  while (i ≠ x) or (j ≠ y) do                               {stop going if N is found}
    if Dia_move then                          {the moving direction is along diagonal}
      Algorithm Move_Along_Dia                           {compute the node's location}
    else                      {diagonal moving is superior than horizontal moving}
      if X_move then                              {move to the next node horizontally}
        Algorithm Move_Along_X                           {compute the node's location}
      else                                          {move to the next node vertically}
        Algorithm Move_Along_Y                           {compute the node's location}
end;
```

```
Algorithm Move_Along_X (x, y, c, i, j, X_move, Dia_move) ;  …………..  Alg. 3
Input :
        x, y, c, i, j, N, X_move, Dia_move
Output :
        x, y, i, j, N, X_move, Dia_move
begin
  if (i+1) > c then                    {the current node is on the right edge}
    Algorithm Move_Along_Y             {so the moving is vertically turned}
  else                                 {the horizontal moving is legal}
    i ← i + 1                          {define the new node's location}
    N ← N + 1                          {keep going}
    if (i = x) and (j = y) then stop   {done. (x, y) =N}
    else
      X_move ← .false.  {horizontal moving is changed vertically next time}
      Dia_move ← .true.        {the moving direction is turned to be diagonal}
  return
end;
```

```
Algorithm Move_Along_Y (x, y, c, i, j, N, X_move, Dia_move) ;  ………..  Alg. 4
Input :     x, y, c, i, j, N, X_move, Dia_move
Output :    x, y, i, j, N, X_move, Dia_move
begin
  if (j = c) then                      {the current node is on the upper edge}
    Algorithm Move_Along_X             {so the moving is horizontally turned}
  else                                 {the vertical moving is legal}
    j ← j + 1                          {define the new node's location}
    N ← N + 1                          {keep going}
    if (i = x) and (j = y) then stop   {done. (x, y) =N}
    else
      X_move ← .true.     {vertical moving is changed horizontally next time}
      Dia_move ← .true.        {the moving direction is turned to be diagonal}
  return
end;
```

```
Algorithm Move_Along_Dia (x, y, c, i, j, Dia_up, Dia_move, X_move) ;
                                                                        ……….    Alg. 5
Input :    x, y, c, i, j, Dia_up, Dia_move, X_move
Output :   x, y, i, j, Dia_up, Dia_move, X_move
begin
   if Dia_up then                  {the moving direction is along diagonal upward}
     if (i = 1) or (j = c) then                  {the current node is on the boundary}
       Dia_up ← .false.       {diagonal moving is changed downward next time}
       Dia_move ← .false.     {the moving direction is turned to be horizontal}
       if (j = c) then                          {the current node is on the upper edge}
         Algorithm Move_Along_X     {so the moving is horizontally turned}
       else                                     {the current node is on the left edge}
         Algorithm Move_Along_Y         {so the moving is vertically turned}
     else                                       {the upward diagonal moving is legal}
         i ← i – 1                                    {define the new node's location}
         j←j+1
         N ← N + 1                                                      {keep going}
         if (i = x) and (j = y) then stop                           {done. (x, y) =N}
   else                            {the moving direction is along diagonal downward}
     if (j =1) or (i = c) then                  {the current node is on the boundary}
       Dia_up ← .true.        {diagonal moving is changed upward next time}
       Dia_move ← .false.     {the moving direction is turned to be horizontal}
         if (j = 1) then                        {the current node is on the lower edge}
           Algorithm Move_Along_X   {so the moving is horizontally turned}
         else                                   {the current node is on the right edge}
           Algorithm Move_Along_Y       {so the moving is vertically turned}
     else                                       {the downward diagonal moving is legal}
        i ← i – 1                                     {define the new node's location}
        j←j+1
        N ← N + 1                                                       {keep going}
        if (i = x) and (j = y) then stop                             {done. (x,y) =N}
   return
end;
```

(二) 一維空間二維編碼法

捷徑法之演算法設計如 Alg. 6 所示，首先以主對角線涵蓋總節點數，將網格劃分上下兩三角。於下三角部分，對角線 k 取 $\sqrt{2N}$ 之整數部分，其尚需考量捨去之小數部分，藉由 k 涵蓋的節點數與 N 判斷 k 是否增 1；上三角部分，以右上角為原點起算，因此需重新計算 N，其 k 之計算與下三角部分相同。當確定節點所在的對角線 k，由 k* (k-1) ***div*** 2 計算第 k-1 條對角線涵蓋的節點數，再依 k 的奇偶數，分別計算節點之 x、y 座標。

```
Algorithm Cantor_N_to_XY (N, c, x, y) ;  {short-cut approach} ............... Alg. 6
Input :
        N：the serial number of a specific node
        c：size of domain (c*c)
Output :
        x, y：location of the given node
begin
  qty_lower_trian. ← c* (c+1) div 2   {quantity of nodes in the lower triangular part}
  if  N < qty_lower_trian.  then      { the node is located at lower triangular part}
    k ← floor (√2N ) {the node is located at the k-1 or k-th diagonal from the origin}
    if  N ≧ k* (k+1) div 2  then  k ← k + 1     {the k-th diagonal is conformed}
    qty1 ← k* (k-1) div 2  {quantity of nodes in the lower left of the k-th diagonal}
    if  odd (k)  then
        y ← k – N + qty1
        x ← k – y + 1                                                       {done}
    else
        x ← k – N + qty1
        y ← k – x + 1                                                       {done}
  else                                       {the node is located at upper triangular part}
    N ← c * c – 1 – N       { in terms of the origin being as at the upper right cell}
    k ← floor (√2N ) {the node is located at the k-1 or k-th diagonal from the origin}
    if  N ≧ k* (k+1) div 2  then  k ← k + 1     {the k-th diagonal is conformed}
    qty2 ← k* (k-1) div 2      {quantity of nodes in the upper right of the k-th diagonal}
    if  odd (k)  then
        x ← c – N + qty2
        y ← 2c – k - x + 1                                                  {done}
    else
        y ← c – N + qty2
        x ← 2c – k - y + 1                                                  {done}
end;
```

逐步法之演算法設計如 Alg. 7，由原點 i = 0、座標爲 (1,1) 開始，隨 N 值的變化，逐步計算所對應的 x、y 座標。其演算法與二維空間一維編碼法相似，皆使用三個布林參數控制曲線沿 x 軸 (Alg. 8)、y 軸 (Alg. 9) 及對角線 (Alg. 10) 的路徑方向行走。

```
Algorithm Cantor_N_to_XY (N, c, x, y) ;  {stepwise approach}  ................  Alg. 7
Input :
        N：the serial number of a specific node
        c：size of domain (c*c)
Output :
        x, y：location of the given node
begin
  i ← 0                                          {working variable for N}
  x ← 1                              {the node's location initialized as (1,1) }
  y ← 1
  X_move ← .true.    {a Boolean variable which controls the moving direction}
  Dia_move ← .false.          {the moving direction is horizontally initialized}
  Dia_up ← .true.          {while Dia_move, it controls upward or downward}
  while  i ≠ N  do                                        {stop going if i = N}
    if Dia_move then                       {the moving direction is along diagonal}
      Algorithm Move_Along_Dia                      {compute the node's location}
    else                     {diagonal moving is superior than horizontal moving}
      if X_move then                          {move to the next node horizontally}
        Algorithm Move_Along_X                      {compute the node's location}
      else                                      {move to the next node vertically}
        Algorithm Move_Along_Y                      {compute the node's location}
end;
```

```
Algorithm Move_Along_X (x, y, c, i, j, X_move, Dia_move) ;  ………….  Alg. 8
Input :
        x, y, c, i, j, X_move, Dia_move
Output :
        x, y, i, j, X_move, Dia_move
begin
  if (x = c) then                       {the current node is on the right edge}
    Algorithm Move_Along_Y              {so the moving is vertically turned}
  else                                  {the horizontal moving is legal}
    x ← x + 1                           {define the new node's location}
    i ← i + 1                           {keep going}
    X_move ← .false.     {horizontal moving is changed vertically next time}
    Dia_move ← .true.    {the moving direction is turned to be diagonal}
  return
end;
```

```
Algorithm Move_Along_Y (x, y, c, i, j, X_move, Dia_move) ;  ……………..  Alg. 9
Input :  x, y, c, i, j, X_move, Dia_move

Output :  x, y, i, j, X_move, Dia_move
begin
  if (y = c) then                       {the current node is on the upper edge}
    Algorithm Move_Along_X              {so the moving is horizontally turned}
  else                                  {the vertical moving is legal}
    y ← y + 1                           {define the new node's location}
    i ← i + 1                           {keep going}
    X_move ← .true.      {vertical moving is changed horizontally next time}
    Dia_move ← .true.    {the moving direction is turned to be diagonal}
  return
end;
```

```
Algorithm Move_Along_Dia (x, y, c, i, j, Dia_up, Dia_move, X_move) ;  ..  Alg. 10
Input :   x, y, c, i, j, Dia_up, Dia_move, X_move

Output :   x, y, i, j, Dia_up, Dia_move, X_move
begin
  if Dia_up then                    {the moving direction is along diagonal upward}
    if (x = 1) or (y = c) then                   {the current node is on the boundary}
      Dia_up ← .false.        {diagonal moving is changed downward next time}
      Dia_move ← .false.       {the moving direction is turned to be horizontal}
      if (y = c) then                          {the current node is on the upper edge}
        Algorithm Move_Along_X        {so the moving is horizontally turned}
      else                                     {the current node is on the left edge}
        Algorithm Move_Along_Y          {so the moving is vertically turned}
    else                                       {the upward diagonal moving is legal}
      x ← x - 1                                     {define the new node's location}
      y ← y + 1
      i ← i + 1                                     {keep going}
  else                    {the moving direction is along diagonal downward}
    if (y = 1) or (x = c) then                   {the current node is on the boundary}
      Dia_up ← .true.           {diagonal moving is changed upward next time}
      Dia_move ← .false.       {the moving direction is turned to be horizontal}
      if (y = 1) then                          {the current node is on the lower edge}
        Algorithm Move_Along_X        {so the moving is horizontally turned}
      else                                     {the current node is on the right edge}
        Algorithm Move_Along_Y          {so the moving is vertically turned}
    else                                     {the downward diagonal moving is legal}
      x ← x + 1                                     {define the new node's location}
      y ← y + 1
      i ← i + 1                                                     {keep going}
  return
end;
```

評估分析

Cantor SFC 的評估，採用四種指標進行評估，並藉由迴歸分析，以多項式函數形式表示。另根據本研究所設計的演算法細部程序，藉由複雜度的分析，探討各法的時間複雜度，明顯地其下限爲 Ω (1)。

(一) SFC 之評估

Laurini and Thompson (1992) 提出三種指標，分別爲路徑全長、單位長度種類與平均四強鄰距離和，再由本研究設計之演算法編寫程式，另增加評估指標的計算程式，即可進行評估。此外，增一評估指標爲平均四弱鄰距離和。

以 Cantor SFC (order = 2, c = 4) 爲例，如圖三：

(1) 路徑全長

係指從起點至終點所經路徑長。由⓪至①、②至③、⑤至⑥、⑨至⑩、⑫至⑬與⑭至⑮距離均爲 1，餘皆爲 $\sqrt{2}$，因此路徑全長爲 $(6\times1+9\times\sqrt{2})$ 約等於 21.49。

(2) 單位長度種類

乃指由一個節點到下一個節點的路徑長。單位長度種類有兩種，一爲行間的增加，其值爲 1，另爲換行，其值爲 $\sqrt{2}$。

(3) 平均四強鄰 (strong neighbor；type Ⅰ neighbor) 距離和

由一節點至其它四個鄰近節點的平均距離和。由⑤至④與⑥之距離均爲 1，至①、⑨皆爲 4，故其值爲 (1+1+4+4) = 10，於⑥、⑨、⑩之值也與⑤相同。平均四強鄰距離和爲 (10+10+10+10) /4 = 10。

(4) 平均四弱鄰 (weak neighbor；type Ⅱ neighbor) 距離和

即一節點至其它四個對角節點的平均距離和。由⑤至②與⑧之距離均爲 3，至⓪與⑩皆爲 5，故其值爲 (3+3+5+5) = 16，於⑥、⑨、⑩之值也與⑤相同。平均四弱鄰距離和爲 (16+16+16+16) /4 = 16。

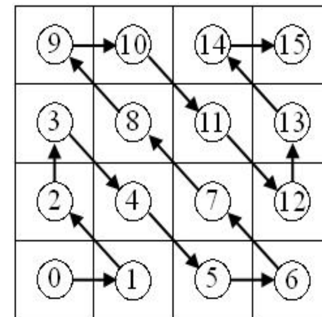

圖 3　Cantor SFC 之路徑排序

依據 Cantor 之演算法，分別撰寫評估指標的計算程式。表 2 爲 order 由 2 至 12 (c = 4..4096) 的評估指標計算成果。將各評估指標 (E.I.) 進行歸納，尋求以 E.I. = f (c) 的函數式表示。其中部分函數式的係數可由歸納法得知爲正合 (exact)，餘則以迴歸分析求得，如表 3。另將平均四強鄰距離和與平均四弱鄰距離和相加，即平均八鄰距離和，亦以多項式表示之。

表 2　四種評估指標計算結果

網格大小 \ 指標	路徑全長	單位長度種類	平均四強鄰距離和	平均四弱鄰距離和
order = 2, c = 4	18.73	2	13.00	13.50
order = 3, c = 8	83.30	2	23.89	25.28
order = 4, c = 16	348.2	2	45.29	47.01
order = 5, c = 32	1421.06	2	87.98	89.85
order = 6, c = 64	5739.01	2	173.32	175.26
order = 7, c = 128	23063.85	2	343.99	345.96
order = 8, c = 256	92469.24	2	685.33	687.31
order = 9, c = 512	370302.86	2	1368.00	1369.99
order = 10, c = 1024	1482061.51	2	2733.33	2735.33
order = 11, c = 2048	5929944.40	2	5464.00	5466.00
order = 12, c = 4096	23723172.58	2	10925.33	10927.33

表 3　四種指標之函數式

函數式 / 指標	函數式		備　註
路徑全長	$2\ (c\text{-}1) + (c\text{-}1)^2\sqrt{2}$		**exact**
單位長度種類	2		**exact**
平均四強鄰距離和	2.66669c+2.60027		σ_{c1} = 0.0000249
	σ_0	0.1002745	σ_{c0} = 0.0354459
平均四弱鄰距離和	2.66681c+4.27210		σ_{c1} = 0.0001375
	σ_0	0.5545909	σ_{c0} = 0.1960416
平均八鄰距離和	5.33350c+6.87237		σ_{c1} = 0.0001623
	σ_0	0.6546241	σ_{c0} = 0.2314022
補充說明	σ_{c1}：一次式係數的標準誤差 σ_{c0}：常數的標準誤差		

(二) SFC 演算法之分析

二維空間一維編碼法中，捷徑法如 Alg. 1 所示，其計算皆在固定且有限的步驟內計算完成，故複雜度爲 O (1)。O (1) 在此的物理意義爲，本研究找出了輸出的未知數與輸入參數間之數學函數式，即計算的效率與輸入値的大小無關。逐步法的主演算法爲 Alg. 2，藉由三個子演算法控制 SFC 的走向，其計算由原點依 x, y 座標値，逐步推算至 N，因 x 與 y 的複雜度爲 O (c)，故逐步法的複雜度爲 O (c^2)。

於一維空間二維編碼法中，捷徑法如 Alg. 6 所示，其計算均爲固定且有限之步驟內完成，故複雜度明顯地爲 O (1)。而逐步法中，其複雜度仍如二維空間一維編碼法，皆爲 O (c^2)。

兩種轉換型態的演算法進行複雜度分析，其結果如表 4。

表 4　Cantor SFC 演算法的複雜度

複雜度 / 型式	二維空間一維編碼法	一維空間二維編碼法
捷徑法	O (1)	O (1)
逐步法	O (c^2)	O (c^2)

結　語

1. 當網格大小 (c, c = 2^{12}= 4096) 時，九種常見 SFCs 的近鄰性以 Cantor 爲最優，並可以一線性函數式表示，平均八鄰距離和爲 5.3c + 6.87。

2. 本研究設計 Cantor SFC 的演算法，依其空間的轉換分爲兩種型式－二維空間一維編碼法與一維空間二維編碼法，設計時，又分爲捷徑方式與逐步方式。捷徑式的設計，表示了輸出的未知數與輸入參數之間已然建立數學函數式。

3. 演算法的分析結果顯示：Cantor SFC 捷徑方式設計的複雜度爲 O (1)，已達最佳化 (asymptotically optimal)，近鄰性的指標亦顯示爲最優，當網格大小爲特定已知時 (如遙測影像的 6000x6000)，似可考慮應用於資料庫的設計。

引用文獻

史天元、何心瑜 (2005) *二維空間一維編碼法之評估*，地籍測量，24 (3) : 46-59。

李佩芝 (2005) *空間填充曲線之演算法探討*，國立嘉義大學土木與水資源工程學系碩士論文。

唐中實、黃俊峰、尹平、朱麗云、王越國、王淑伟、李

小乙譯 (2004) ，地理信息系統（上卷）—原理與技術（第二版），北京：電子工業出版社。

Breinholt, G. and Schierz C. (1998) *Algorithm 781: Generating Hilbert's Space-Filling Curves by Recursion*, ACM Trans. On Mathematical Software, 24 (2) : 184-189.

Butz, A. R. (1971) *Alternative Algorithm for Hilbert's Space-Filling Curve*, IEEE Trans. Comput., C-20: 424-426.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1991) *Introduction to Algorithms*, MIT Press, 5-th printing, New York.

Laurini, R. and Thompson, D. (1994) *Fundamentals of Spatial Information Systems*, The A. P. I. C. Series, # 37.

Mark, D. M. and Goodchild, M. F. (1986) *On the Ordering of Two Dimensional Space：Introduction & Relation to Tesseral Principles*, Proceedings of the Workshop on Spatial Data Processing Using Tesseral Methods, Swindo, UK: Natural Environment Research Conncil,.

Manber, U. (1989) *Introduction to Algorithms：A Creative Approach* , Addison-Wesley, New York.

Mokbel, M. F. and Aref, W. G.. (2001) *Irregularity in Multi-Dimensional Space-Filling Curves with Applications in Multimedia Databases*, Proc. of the 2nd Intl. Conf. on Information and Knowledge Management, Atlanta, GA, 512-519.

Mokbel, M. F., Aref, W. G., and Kamel, I. (2003) *Analysis of Multi-Dimensional Space-Filling Curves*, GeoInformatica, 7 (3): 179-209.

Rigaux, P., Scholl, M. and Voisard, A. (2002) *Spatial Databases with Application to GIS*, Morgan Kaufmann, New York.

Sagan, H. (1994) *Space-Filling Curve*, Springer-Verlag, New York.

Samet, H. (1990) *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading Mass, New York.

94 年 12 月 27 收稿

95 年 03 月 14 修正

95 年 04 月 25 接受