以Java^{TM}實作的演算法設計與分析平台 (JGAP)
Java^{TM}-based graph algorithm design and analysis platform (JGAP)

陳定彝　　蔡錫鈞
國立暨南國際大學資訊管理所
Department of Information Management
National Chi-Nan University
Puli, Nantou, Taiwan, R.O.C
definite@im.ncnu.edu.tw　　tsai@csie.ncnu.edu.tw

摘要

本論文介紹筆者以 *Java*^{TM} 這個物件導向程式語言所開發的一個圖形 演算法發展與分析平台 *(Java-based Graph Algorithm design and analysis Platform,* 簡稱 *JGAP)*。除了以應用程式模式執行外，更可以使用 網際網路瀏覽器來使用本系統。此外，為了圖形演算法的執行效率，我們提出了一種圖形表示法—— *Adjacency ListArray*，融合了 *Adjacency Matrix* 表示法 與 *Adjacency List* 表示法的優點。

*JGAP*的設計理念就是要讓使用者用*JGAP*就像電視遊樂器般，只要插入演算法「卡匣」，就可以測試使用該演算法，內建的 *performance meter* 可以顯示演算法在不同大小的圖形下的執行效率。除此之外，*A* 演算法所產生出來的結果，可以餵給 *B* 演算法當輸入，使用 *History list* 可以方便地知道執行了那些演算法，以及演算法執行前和執行後之圖的變化。

在*JGAP*中，演算法發展者僅需要製作演算法「卡匣」，其他的瑣事，就交給*JGAP*。這對圖形演算法之測試與設計將有許多助益。

關鍵字： *Java, Graph Algorithm, JGAP, Performance meter, Adjacency ListArray.*

1 緒論

我們提供了一個以Java實作的圖形演算法發展與分析平台 (Java-based Graph Algorithms design and analysis Platform, JGAP) 以處理有向加權圖(directed weighted graphs)或無向加權圖 (undirected weighted graphs)。

設計圖形演算法需要哪些特性呢？首先，演算法設計者希望在設計演算法時，能夠輸入自己的圖形來做測試，並要以圖形來顯示結果。其次，要能夠測量演算法的執行時間。雖然 時間複雜度 (Time Complexity) 可以由數學方法計算出來，但是人們 將會對演算法的實際執行時間有興趣，並藉以評估演算法的優劣。演算法研發完成後，也希望它能具備網際網路的展示能力，以便教學及研究用。目前已有多種展示演算法動畫的環境，諸如JAWAA[4]、JSamba[6]以及GeoJava[1]等等。我們也要實作圖形演算法的展示與分析平台。

此外，在發展程式中，耗時最久的往往不是在程式功能本身，而在使用者介面。若是能夠有個可供套用的介面，演算法發展者將可以專注於 演算法本身的設計，將使用者介面的設計減低到最小的程度。

JGAP 設計的目的就是為了輔助演算法發展者的設計工作，他們僅僅需要 設計演算法本身以及處理演算法的參數設定 (如 Maximum Flow 演算法 要處理 Source 頂點和 Sink 頂點這兩個參數)就可以將他們所製作的演算法 套入 JGAP 中。JGAP 同時也提供了一些實用的工具(諸如 performance meter、history list) 以方便演算法的設計、驗證與測量。以下是 JGAP 的功能簡介：

1. 檔案處理：可將現在編輯的圖加以存檔，以備日後使用。
2. 圖形產生：可以設定如頂點數，邊的權重 (Weight of Edge) 的上下 限等規格，系統就會自己產生圖。
3. 圖形編輯：可以畫出自己想要的圖，以便執行演算法。
4. 常用演算法：本系統提供了 BFS、DFS、Minimum Spanning Trees、Shortest Path、All pair shortest path、Maximum Flow [5] 等數種圖形演算法以供圖形演算法套用。它們也是圖形演算法的範例程式，演算法發展者可以參考 其架構以便發展新的演算法。
5. 演算法執行與顯示：可以執行自己想要的演算法，並可以觀看並 儲存執行結果。
6. Performance meter：系統會自動產生大小不一的圖形，並自動統 計並繪圖。
7. 歷程紀錄(History list)：可留存演算法執行前後的圖，以便比較及 餵入其他演算法。
8. 網際網路顯示能力：可以將自己設計的演算法展示在WWW上，以便教學或研究使用。

2 Graph representation

首先我們要對我們所處理的圖形 (Graph) 下定義：

> 一圖形$G(V,E)$包含了一有限集合，由頂點 (Vertex) 或節點 (Nodes) 以及 聯結若干頂點對的 (Edges) 所構成。若我們以此頂點對指明一邊，則我們可以視 G 為$(V(G),E(G))$，其中 $V(G)$ 為一代表頂點的有限集合；而 $E(G)$ 代表邊，為 $V(G)$ 之若干個 2-元素子集合所成的集體。我們以 $\overline{V_1V_2}$ 表示聯結頂點 V_1 與 V_2 的邊。[8]

2.1 Abstract type of Graph

在物件導向程式設計中，圖形也是一個物件，由眾多的頂點與邊所構成。當然，頂點與邊也是物件。我們對

頂點與邊的類別 (Classes) 中的欄位 (fields) 分別定義如下：

```
class Vertex{
    int identity; // Identity of this vertex.
    int predecessorNode; // The predecessor of this vertex.
    int x; // The x position of this vertex.
    int y; // The y position of this vertex.

    // Status of this vertex. Use color represention.
    Color color;

    int hopDist; // Hop distance to this vertex.
    int inDegree; // Indegree of the vertex.
    int outDegree; // Outdegree of the vertex.
    double distance; // Distance to this vertex.
}


class Edge{
    // Indicate the node that this edge begin with.
    int fromNode;

    // Indicate the node that this edge end with.
    int toNode;

    // The weight or the capacity of the edge.
    double weight;

    // The flow of this edge.
    double flow;

    // Status of this edge. Use color represention.
    Color color;
}
```

而圖形類別的欄位定義如下：

```
class Graph{
    // The serial number of the vertex.
    // Indicate the number of vertices,
    protected int vertexSerialNo;

    // Store all vertices
    ListArray vertexList;
    // Store all edges
    ListArray2 edgeList;
    // Whether the graph directed.
    boolean directed;
}
```

我們使用了 ListArray 來表示頂點，而使用 ListArray2 來表示邊。那麼，什麼又是 ListArray 、 ListArray2 呢？

2.2 ListArray

ListArray 的架構如圖1， 可視為 doubly linked-list 和 array 的綜合體。它同時也具有 doubly linked-list 的特性：可以像 array 一樣使用 index 來隨機存取資料，也可以像 linked-list 一般可以循序存取資料。這種特性使得 ListArray 不論在存取隨機或循序資料的時間複雜度上都有相當的優勢。

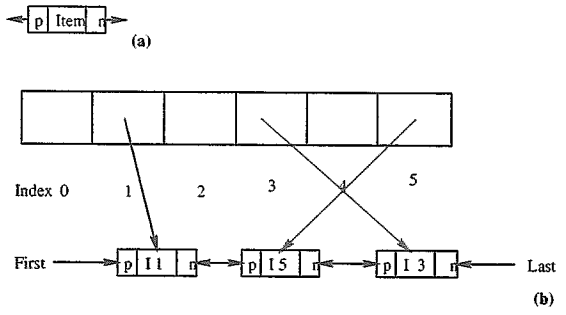

Figure 1: ListArray

ListArray 可分為兩部分：ListItem (Linked-list Item)與 ListArray 本體。ListItem 有點像是 Doubly Linked List 的 Link 。和一般的 Dobuly Linked List 相同，有向前一個 Link (prev) 及向後一個 Link (next) 的參考。放入 ListArray 的物件，事實上是先放入ListItem中，再把 ListItem 放入 ListArray 中。而 ListArray 本體是一個 ListItem 的 Array。ListArray 的 Array 功能實際上就是這部份所提供。
ListArray 最大的特點就是可同時提供 Linked List 和 Array 的存取方法。

- Array方法:
  - itemAt(x): 取出在 index x 的地方的物件。時間複雜度為$O(1)$。
  - setItemAt(o,x): 將 object o 設定在index x。時間複雜度為$O(1)$。
  - itemExistAt(x): 是否在 index x 的地方有物件。時間複雜度為$O(1)$。
- Linked List 方法:
  - getPrev,getPrevIndex: 取得前一個 ListItem 或其 index 。時間複雜度為$O(1)$。
  - getNext,getNextIndex: 取得下一個 ListItem 或其 ListItem 的 index。時間複雜度為$O(1)$。
  - getFirst,getFirstIndex: 取得第一個 ListItem 或其 index。時間複雜度為$O(1)$。
  - getLast,getLastIndex: 取得最後一個 ListItem 或其 index。時間複雜度為$O(1)$。
  - elements,nextElement,hasMoreElement: 使用 Interface Enumeration[2] ，可以循序將 ListArray 依據 Linked-List 的順序一個個取出。時間複雜度分別為$O(1)$,$O(1)$,$O(1)$。

在 ListArray 中加入一個新元素或刪除一個舊元素都像 doubly linked-list 一樣只需 $O(1)$就可完成。我們要如何把一個物件放入 ListArray 中呢？由於使用者可能會使用到 Array 的方法，所以要指定這物件放置的位置。然後將此物件放入 ListItem ，再將 ListItem 放入定位。最後再把相關 Link 連結起來。如此僅需$O(1)$即可完成加入元素的動作；而 刪除一個ListArray中的item也和 Doubly Linked List 類似，先把連結刪除， 再刪除該 Item 。同樣的，也僅需$O(1)$就可以刪除一個元素。

2.3 ListArray2

ListArray2 為 2 維的 ListArray， 其結構如圖2所示。基

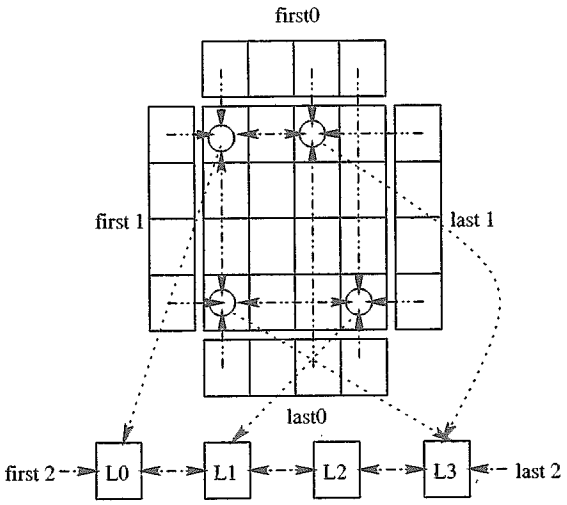

Figure 2: 一個ListArray2

本原理和 ListArray 相同，但多了 2 維的支援。由於 2 維有兩個方向，故在連結時也要連結兩個方向。

ListArray2 中所需要的 (prev,next) 有三對，一對 (prevX,nextX)，指出水平方向的前一個元素及後一個元素，可以用來取出某一列中的所有元素。用在圖形表示法中，就是取出所有射出邊。而 (prevY,nextY) 指出垂直方向的前一個元素及後一個元素。可以用來取出某一行的所有元素。用在圖形表示法中，就是取出所有射入邊。虛線所指示的是 (prevZ,nextZ) 指出全部元素中的前一個元素和後一個元素，可以用來取出 ListArray2 中所有的元素。用在圖形表示法中，就是取出所有邊。至於在 ListArray2 中加入或刪除元素的方法與 ListArray 類似，在此不加以贅述。

2.4 處理圖形的方法

我們定義了如下的方法，演算法發展者可以利用這些方法來處理圖形：(G為圖形物件 (Graph Object)，V為頂點物件(Vertex Object)，E為邊物件 (Edge Object))

頂點的處理方法

V.getInDegree()	得知頂點上的 in-degree
V.getOutDegreee()	得知頂點上的 out-degree
V.getColor()	得知頂點的顏色 (狀態可以由其顏色代表)
V.setColor()	設定頂點的顏色

邊的處理方法

E.getFromNode()	得知邊的起始點
E.getToNode()	得知邊的結束點
E.getColor()	取得邊的顏色 (狀態可以由其顏色代表)
E.setColor()	設定邊的顏色

圖形的處理方法

G.getNumOfVertices()	得知本圖共有多少頂點
G.getNumOfEdges()	得知本圖共有多少邊
G.verticesElements()	列舉所有的頂點
G.allEdgesElements()	列舉所有的邊
G.addVertex(v)	在圖 G 中加入頂點 v
G.deleteVertex(v)	在圖 G 中刪除頂點 v
G.addEdge(e)	在圖 G 中加入邊 e
G.deleteEdge(e)	在圖 G 中刪除邊 e
G.directedAdjVerticesElements(v)	列舉頂點 v 的相鄰節點。
G.undirectedAdjVerticesElements(v)	列舉頂點 v 的相鄰節點，不考慮邊的方向。
G.isEdge(u,v)	頂點 u,v 之間是否有一邊？
G.inEdgesElements(v)	列舉射入頂點 v 的邊
G.outEdgesElements(v)	列舉發出頂點 v 的邊

2.5 Adjacency ListArray 圖形表示法

以往我們在電腦上表示圖形，可以用三種表示法來表示：Edge List 表示法、Adjacency List 表示法、與 Adjacency Matrix 表示法。但是這些表示法各自都有缺點 都分別有各自的缺點，不是不利於 G.isEdge()，就是列舉相鄰邊時需要花上 $O(n)$。[7] 因此我們設計了 ListArray ——融合了 Linked-List 和 Array 的優點。為了討論方便起見，n 代表圖形的頂點數，m 代表圖形的邊數。

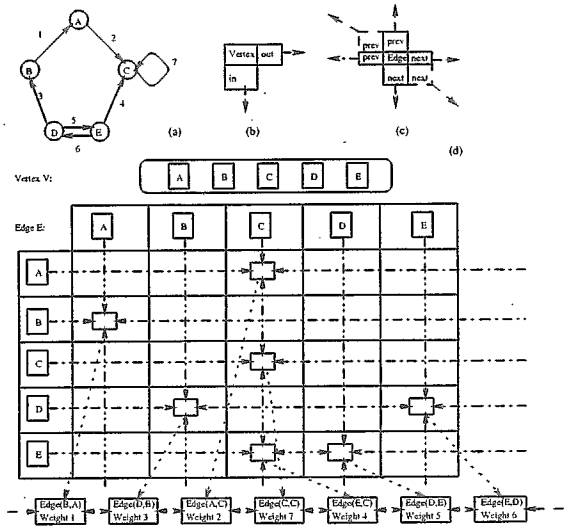

Figure 3: 以Adjacency ListArray 表示法表示的圖

Adjacency ListArray 可以看做 Adjacency list 和 Adjacency matrix 的綜合體。如圖3，(a)為一有向圖；(b)為

Vertex 的資料結構， in 指向由v射入的邊， out 指向由v射出的邊；(c)Edge 為的資料結構，上面和左邊的 prev 分別指向上一個射入／出邊，下 面和右邊 next 則分別指向下一個射入／出邊。而虛線框裡的 prev 則表示整個圖的上一個邊， next 表示下一個邊。沿著虛線可以取出所有邊。 (d)為(a)用 Adjacency ListArray 表示法表示的圖。 我們可以從圖中看到，我們既可以使用矩陣來取出我們的邊，也可以使用 Linked-list來取出我們的邊。這個表示法的特點如下：

兼具 Adjacency List 和 Matrix 的優點： Adjacency List 在處理點與相鄰邊只需要$O(deg(v))$ (e.g. directedAdjEdgesElements)， 而 Adjacency Matrix 在處理兩點之間是否存在一邊(isEdge)只需要$O(1)$。 Adjacency ListArray 表示法即可綜合這兩者之長，也就是可以如同 Adjacency List 只需要$O(deg(v))$ 就能取出點的所有相鄰邊，而需要$O(1)$就能知道某邊是否存在。

同時提供 Adjacency List 和 Matrix 的方法： 若是想要比較 Adjacency List 和 Adjacency Matrix 的效能，以前必須要 同時實作這兩個資料結構出來，再行比較。 然而使用 Adjacency ListArray 則可以模擬這兩個資料結構。

然而，此法的缺點和 Adjacency Matrix 相同：都需要$O(n^2)$的空間複雜度 (Space complexity)。 對於稀疏的圖來說，這種作法相當浪費空間。

我們根據前面所探討的各種圖形表示法的方法，經過比較後，整理如表 1。表中的 ideg 、 odeg 分別代表頂點的 in degree 及 out degree。 至於 deg 相當於in degree 或out degree，視所求為射入邊或是射出邊 而定。

3 系統介紹

作為一個圖形演算法發展平台，它應該有哪些功能？也就是，它該有哪些特性？

就圖形處理來說。首先，要能夠畫出圖，也就是要有畫出頂點與邊 以及讀檔、存檔等功能。當然，用手畫太累了， JGAP 也提供各種不同的隨機圖形產生器來產生圖。

另外，演算法發展者可能會把一個圖形演算法的輸出結果， 當成另一個演算法圖形的輸入，也就是可以把多個演算法組合成一個新的演算法， 在這些演算法執行過程中，演算法發展者有需要知道其產出過程，也就是圖 G_1 經由演算法 A_1 處理成為圖 G_2 ，再經由演算法 A_2 處理成為圖 G_3 。 此時發展者會時時查閱這些圖，以作為比較。 也就是要能夠保留「使用前」的圖和「使用後」的圖以便比對。 在演算法之間所運用的圖，例如 G_2，往往也 會使用其他的演算法來運算。這時 History List 將會可以提供此種處理方式。

就演算法執行方面，既然我們要處理圖形演算法，就需要圖形顯示能力，而且演算法 發展者不需要花上太多時間和精神去處理顯示介面， 最好有個現成的顯示方式可以使用。 演算法常常需要不同的參數，如 BFS 、 DFS 需要起始點； Kruskal 不需要起始點，但是需要有一個 Priority Queue 來取出最小邊； Prim 則兩者都需要.....。雖然每個演算法所需的參數各異，但是需要共通的介面或方法。 演算法實作完畢後，演算法發展者將對其執行效率有興趣， 用以比較相關的演算法， Performance Meter 將在此扮演一個重要的角色。 瞭解了這些需求之後，就可以發展我們的系統。 從上述的需求，我們的系統將有以下的功能：

- 在圖形處理方面：
  - 手繪圖形：包括畫點、畫邊、刪點、刪邊、改變邊的權重。
  - 檔案處理：讀檔，存檔。
  - 隨機圖形產生器：要能產生指定頂點數的圖，邊的密度， 隨機的邊之權重。使用者也可以決定要不要有self-looped cycle，欲產生有向圖還是無向圖。
  - History List：必須要記錄圖形在演算法執行前的狀況和 執行後的結果。
- 演算法處理：
  - 圖形顯示能力：提供方法使演算法的結果以圖形表示出來。
  - 不同參數的支援：演算法要提供處理不同的輸入參數的方法。
  - 演算法效率評估程式：要能夠產生數個不同大小、不同邊數的圖 ，自動餵給演算法，演算法執行結束後，傳回其執行時間以便 觀察演算法效率。

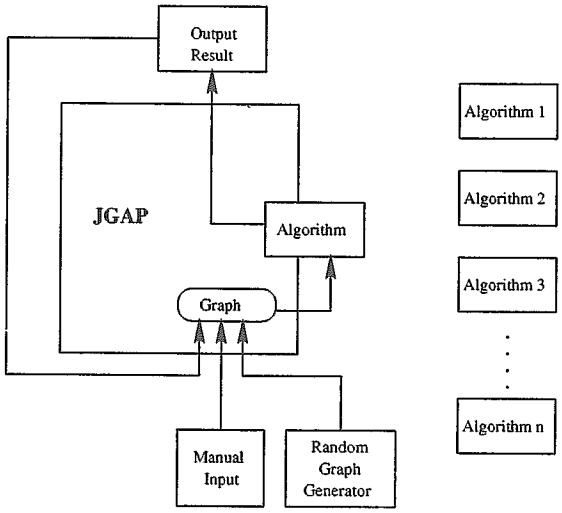

Figure 4: 系統架構示意圖

圖4顯示了 JGAP 的系統架構。 我們可以把 JGAP 想像成一臺電視遊樂器，圖形演算法就像是一塊塊卡匣一般， 現在要「玩」這個演算法，就可以把這個演算法卡匣插入JGAP中 ，並可以使用 JGAP 的「搖桿」(JGAP的使用者介面，圖中的 Manual Input) 來操作 JGAP ，並觀看結果 (Output Result) 。 演算法發展者可以按照既定的架構設計演算法卡匣， 而且，只需設計卡匣部份就好，不必花心思去設計整部機器。 插好演算法「卡匣」後，除了執行演算法外，還可以使用 performance meter 來更進一步分析其執行效率，設計者可以利用這個功能 來評估，改進演算法。

Performance meter 的架構如圖5， 它包含了 一個 random graph generator 以產生輸入圖形；一個 algorithm loader 以載入演算法， 我們可以利用它來載入我們想要測量的演算法； 一個 algorithm timer 收集整理 algorithm 所傳回來的執行時間。 最後 performance meter 可以將結果匯整並展示出來。

Graph Method	Graph Representation			
	Edge List	Adj. List	Adj. Mat	Adj. ListArray
getNumOfVertices,getNumOfEdges	$O(1)$	$O(1)$	$O(1)$	$O(1)$
verticesElements	$O(n)$	$O(n)$	$O(n)$	$O(n)$
allEdgesElements	$O(m)$	$O(m)$	$O(n^2)$	$O(m)$
addVertex	$O(1)$	$O(1)$	$O(1)$	$O(1)$
deleteVertex	$O(m)$	$O(odeg(v) + ideg(v))$	$O(n)$	$O(odeg(v) + ideg(v))$
addEdge	$O(1)$	$O(1)$	$O(1)$	$O(1)$
deleteEdge	$O(1)$	$O(1)$	$O(1)$	$O(1)$
getInDegree,getOutDegree	$O(1)$	$O(1)$	$O(1)$	$O(1)$
directedAdjVerticesElements(v)	$O(m)$	$O(deg(v))$	$O(n)$	$O(deg(v))$
undirectedAdjVerticesElements(v)	$O(m)$	$O(odeg(v) + ideg(v))$	$O(n)$	$O(odeg(v) + ideg(v))$
isEdge(u,v)	$O(m)$	$O(min(odeg(u), ideg(v)))$	$O(1)$	$O(1)$
inEdgesElements	$O(m)$	$O(ideg(v))$	$O(n)$	$O(ideg(v))$
outEdgesElements	$O(m)$	$O(odeg(v))$	$O(n)$	$O(odeg(v))$
getFromNode,getToNode	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Space require	$O(n+m)$	$O(n+m)$	$O(n^2)$	$O(n^2)$

Table 1: 各種圖形表示法的時間複雜度與空間複雜度

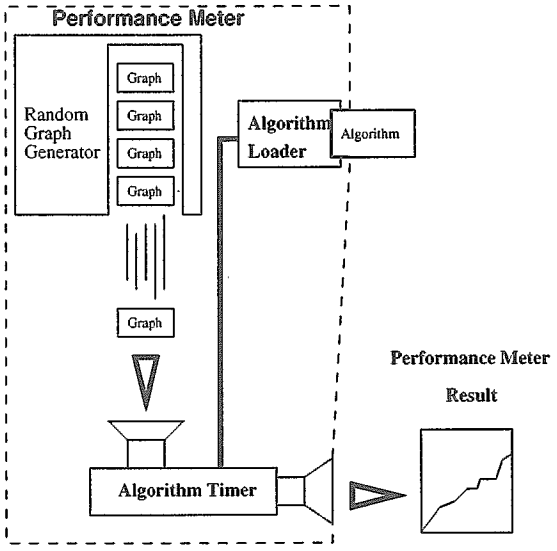

Figure 5: Performance Meter架構示意圖

4 設計與分析演算法

在前面的章節中，已經說明了本系統的資料結構與系統架構。演算法發展者 最有興趣的，莫過於如何使用並操作此系統。本系統可以使用 Java 應用程式 (application) 的方式來執行，也可以在網站上用 Java Applet 的方 式來呈現。 然而使用 web browser 來觀看 java applet 會受到 web browser 的安全性限制， 在 client 端存取檔案有可能不被允許，所以在 applet 執行模式中，我們將存取檔案的 功能取消。惟有在 application 執行模式中，才可以執行所有的功能。

4.1 圖形處理

JGAP 預設可以處理1至100個頂點的圖形。 如果嫌100個頂點太少，可以更改Graph類別中的MAX_VERTICES來放寬 圖形處理的上限。 本系統提供的幾項圖形處理的功能如下：

- 圖形產生：使用者可以手繪圖形，也可以指定想要的圖形種類， 交由圖形產生器產生圖形。
- 檔案處理：使用者可以將 canvas 上的圖形存檔，也可由既有的存檔 取出。
- 圖形運算：除了圖形演算法外，本系統也提供了聯集 (Union) 與差集 (symmetric difference) 可供圖形運算。
- History-list：可以追蹤演算法執行前的圖形狀態與演算法的執行結果。

4.2 圖形演算法

在 JGAP 中，演算法發展者除了可以自己撰寫演算法外， 甚至可以利用系統內建的演算法或其他人撰寫的演算法 來組合成新的演算法。

在緒論我們有提過， JGAP 設計的目的就是要讓演算法發展者僅需專注於 演算法本身的設計，處理使用者介面的部分越少越好。只要繼承 Algorithm 這個 class ，就可以使用JGAP所提供的各項功能。

我們可以依照下列步驟來實作我們的演算法：

1. **建立演算法類別：**
   建立一個名為 FindPath ，繼承 Algorithm 的類別。 在這個類別的 constructor 不必指定參數。但是要設定一下顯示的顏色。 在 JGAP 中 ，我們以顏色代表邊或點的屬性。

2. **實作演算法本體：**
   algorImpl() 這個方法代表演算法本體， 我們可將演算法的實作部分放置在 algorImpl() 這個方法。 algorImpl() 的傳回型態為 Object ，可以傳回任何東西，包括圖、陣列， 或真假 (true/false) 值。

3. **設定參數：**
   algorImpl() 並沒有指定任何參數，但是許多演算法要指定額外的參數 ， FindPath 也不例外，所以我們要實作 setArg(Object args[]) 對傳進來的參數進行分析並設定。
   如果要在 Algorithm 選單中的 Custom 選項中的 Execute 中的 Additional argument 可以設定額外的參數， 則 setArg 必須也要能夠處理 String。 如果要使用 Performance Meter ，則 setArg 也要能處理 null 。 當 args 為 null 時，演算法自己要設定預設值。

4. **設定偏好對話框 (prefered dialog) ： (optional)**
   在 Custom Algorithm 中 ，由於各種演算法需要的參數樣式不 同，所以所需的對話框也不同。發展者可以覆蓋 getPreferedDialog(Graph, Frame) 來定義此演算法所需的對話框。 但若不會使用到 custom algorithm ，就可以不覆蓋此對話框。

按照這個步驟下來，就可以設計能和 JGAP 配合的演算法。

4.3 Performance meter

performance meter 可以測量演算法在不同點數，不同邊數的圖的執行時間。 並依此繪出折線圖 (line chart) 。 選擇 Algorithm 選單中的 Performance Meter 就可以進入performance meter 。只要是 JGAP 可以用的圖形演算法，都可以使用內建的 performance meter。 我們可以運用 performance meter 來觀察此演算法在各種不同 圖形的執行效率，了解演算法在各種圖形所執行的特性， 進而改進成為更實用、更有效率的演算法。

5 結論與未來發展

JGAP 提供了一個較為通用的架構，也提供了現成的圖形方法 與演算法方法以便演算法發展者使用。 並且，只要在網頁中加入 applet 的 tag ，就可以將演算法發展者的作品送上網頁發表，接受使用者的測試與驗證。 並提供了 history list 記錄追蹤演算法的輸入圖形執行結果， 更增加其使用本系統的方便性。 此外，performance meter可以測試演算法的效率，以便演算法發展者 評估演算法的效率，進而改進演算法。
程式展示及說明文件 可以在
http://im.ncnu.edu.tw/
tsai/definite/JGAP/JGAP.html 看到。
目前 JGAP 只能處理簡單圖形，也就是頂點 A 到頂點 B 之間最多只有一條邊， 日後將加入 multi graph [3] 的支援，使得能處理的圖形更多樣化，並加入帳號管理的功能以便教學、討論與研究使用。

References

[1] Kiyoko F. Aoki and D.T. Lee. Towards web-based computing, 1999.

[2] Sun Microsystems Inc. Java platform 1.2 api specification: Class arraylist *http://java.sun.com/products/jdk/1.2/docs/api/java/util/enumeration.html.* Sun Microsystems Inc., Jun 1999.

[3] C. L. Liu. *Elements Of Discrete Mathematics.* McGraw-Hill Book Company, 2 edition, 1985.

[4] Will Pierson's Minimalist. Java and web based algorithm animation, *http://www.cs.duke.edu/wcp/jawaa.html*, 1998.

[5] Thomas H. Cormen & Charles E. Leiserson & Ronald L. Rivest. *Introduction To Algorithms.* McGraw-Hill Book Company, 2 edition, 1990.

[6] Stasku. Java version of the samba animation program , *http://www.cc.gatech.edu/gvu/ softviz/algoanim/jsamba/*, 1998.

[7] Michel T. Goodrich & Roberto Tamassia. *Data Structires and Algorithms in Java.* John Wiley & Sons, Inc., 1998.

[8] 劉睦雄, editor. 平行演算法與矩陣計算. 明文書局, May 1992.