第二十屆旺宏科學獎

成果報告書

參賽編號：SA20-656

作品名稱：圖論演算法學習用之繪圖程式

姓　　名：盧沛宏

關 鍵 字：圖論、Biconnected Component、繪圖演算法、
演算法學習

摘要

本研究之動機源於演算法競賽中的圖論（graph theory）問題。撰寫程式時常常需要將圖手繪出來以幫助思考，而繪圖程式能解決手繪的缺點，作為輔助繪圖的工具。此外，近年來資訊教育日益盛行，演算法的學習越來越受到重視，因此我們針對圖論演算法之學習，設計一套使用者友善的繪圖軟體，解決手繪的缺點，提供高可讀性的繪製結果，作為輔助學習的工具。

本研究針對現有繪圖軟體的缺點，如功能不完整、操作不易或是無自動優化繪製結果，進行整合與改善，設計一套使用者友善的繪圖程式，將其命名為 Graphene。Graphene 除了輔助繪圖之外，也加入時間軸、自訂外觀、參數調整、匯出圖片等功能，用最直覺的方式幫助學習者理解圖論演算法，實作時更能清楚其背後的原理，並提升除錯效率。Graphene 針對演算法學習者與演算法競賽選手，提供可直接輸入圖論題目的文字格式測試資料以產生繪圖結果的功能，並結合現有繪圖演算法，改善、優化樹（tree）及類樹圖（tree-like graph）的繪製結果。此外，Graphene 也提供匯出成圖片的功能，使其成為製作演算法講義附圖的利器，幫助教師製作教材，有助於圖論演算法教學。

Graphene 採用的繪圖演算法以 force-directed graph drawing 演算法為基礎，實作節點的分布。此方法近似現實生活的物理模型，將節點想像成電子，彼此間有排斥力，而邊則想像成連接電子的彈簧，將連接的兩點拉近。然而初始的節點分布會影響繪圖結果，因此，我們先利用 biconnected component、block-cut tree、等圖論結構及 radial tree 的布局方式，配置節點的初始位置，對繪製結果進行優化。

壹、研究動機

在或大或小的演算法競賽中，和圖論演算法相關的題目數見不鮮，甚至幾近每一場程式競賽都會遇見關於圖論的題目。而平時到各大線上解題系統訓練時，題目列表上也可見到不少標記「圖論」的題目，尤其以樹或類似樹（tree-like，指關節點數量較多的圖）的圖居多。當練習圖論題時，若是程式出錯，就得把圖繪製出來尋找錯誤，然而題目通常是要能夠簡易、快速輸入給程式處理的資料，測試資料的形式不可能是一張能夠清楚表達圖的結構的圖片，大都只是描述節點間關係的文字，如圖 1 便是個常見的圖論題目測試資料輸入格式（例如：2 號點連接到 4 號點，便以「2 4」表示）。然而對人們來說，僅靠那些文字讓腦中浮現圖的樣貌幾乎不可能；縱使利用紙筆將圖手繪出來，通常也得花不少時間調整，才能讓圖清楚且完整的呈現。此時若有一個程式能夠讀取這些圖的文字形式資料，便能自動繪製出一張清晰、可讀性極高的圖，而且還能夠讓使用者自行調整節點位置、大小、距離或是在圖上加註、標記，將能夠大幅提升除錯的效率，是相當方便的工具。此外，若能更加清楚呈現類樹圖的繪製結果，不只能一眼看出關節點的分布，尋找兩節點之間的路徑也變得相當快速，這對於演算法競賽圖論題目的觀察與分析非常有幫助。

輸入說明

測試資料第一行有兩個數字 n, m，第一個數字 n 表示圖中的節點數，第二個數字 m 表示圖中的邊數，接下來會有 m 行，每行有兩個正整數 i, j，$i \neq j$，$1 \leq i \leq n$，$1 \leq j \leq n$，表示節點 i 和 j 之間有一條邊。

範例輸入

```
4 4
1 2
4 3
2 4
1 3
```

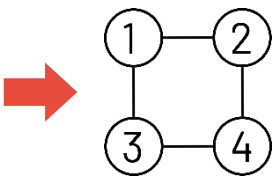


圖1. 常見的圖論題目輸入格式

此外，近年來資訊教育逐漸盛行、發展，演算法的學習越來越受到重視，甚至已經被納入新課綱。平時學習圖論演算法時，幾乎都須以視覺化的圖像作為輔助。比起閱讀生硬的文字，完整且清晰的圖不僅能幫助學習者對於圖論知識的理解，對於程式的撰寫更有非常大的幫助。正因如此，若有一套能將圖清楚、簡潔的呈現在眼前，且能讓使用者動手操作、隨著思維彈性地改變圖的樣貌，甚至能搭配各式演算法，將每個演算法的步驟利用類似動畫的方式完整演繹，不管對於

學習者還是教學者，都會是一個效益極高、非常方便的工具。

然而，大部分現成的繪圖程式功能並沒有那麼符合演算法的學習需求，且多數在操作上對使用者不夠友善，例如：Graphviz、TikZ、Mathematica 等以語法為主的軟體，雖然這類軟體功能強大，但必須學習新的語法，對初學者而言門檻較高，而且此類軟體並未提供使用者直接在圖形介面上自訂修改的功能，只能回頭修改生成圖的原始碼，再令它生成新的圖，相當不方便。另一類常見的繪圖軟體提供使用者繪製多種類型的圖，如：Ipe、Inkscape，及大家常用的 Office 軟體繪圖功能，這類軟體和手繪過程相仿，仍需自行拖曳節點位置校正，並未針對圖的視覺化進行自動優化。CS Academy 的 Graph Editor 提供以文字輸入產生圖的功能，也可以在圖形介面上拖曳節點或新增、編輯、刪除節點及邊，再進行自動化布局，是個相當不錯的輔助工具。然而 Graph Editor 功能較陽春，能夠做的不多。在眾多繪圖布局軟體中，yEd 和 Gephi 算是功能最齊全的，它如同一般的繪圖軟體提供互動式的圖形介面，並提供許多不同的繪圖演算法供使用者選擇以完成自動布局的功能。不過，如果要以文字輸入圖形資料，yEd 必須要先存成檔案再讀入並執行布局功能，Gephi 則是在使用者介面上操作比較麻煩。這些繪圖程式都無法以簡單快速的方式輸入圖論題目常見的資料格式繪製圖形。

由於大多數提供圖布局功能的軟體都不是針對圖論演算法的學習而設計的，所以在繪製圖形時，常常會覺得不方便。因此，此研究的目的是針對演算法常見的圖形結構，希望能開發一個符合大部分需求的繪圖軟體。本研究以現有的繪圖演算法 force-directed graph drawing algorithm 為基礎，結合 biconnected component、block-cut tree 等圖論結構及 radial tree 的布局方式，改善圖的繪製結果，以樹及類樹圖（tree-like graph）的優化為主軸。接著針對演算法所使用的參數進行測試，找出其最佳組合。最終設計出一個易於操作的繪圖程式，提供友善且完整的使用者介面和自定義選項，讓使用者彈性地根據需求，調整各項參數、指定圖的輸出格式（如 SVG、PNG 等）。提供學生學習演算法、練習競賽題目或是教師準備教材的一個簡單方便的程式使用。

貳、研究目的

一、針對學習圖論演算法的需求，設計一套使用者友善的繪圖軟體，解決手繪的缺點，提供高可讀性的繪製結果，作為輔助學習的工具。

二、利用 force-directed graph drawing 等繪圖演算法和 biconnected component、block-cut tree、radial tree 等圖論結構，改善樹及類樹圖（tree-like graph）的繪製結果。

三、設計易於操作的使用者介面，加入學習的功能及幫助撰寫程式與除錯的工具。

四、提供使用者自定義選項，例如：調整外觀樣式、參數、節點位置等，並提供匯出成圖片的功能，給予使用者彈性操作空間。

參、研究設備及器材

一、 硬體

(一) 電腦

二、 軟體

(一) 程式語言：C++

(二) 開發與測試環境：

1. IDE：CLion by JetBrains
2. 編譯器：mingw-64 x86_64-8.1.0-posix-seh-rt_v6-rev0
3. 作業系統：Windows 11

(三) 使用的外部程式庫：

1. Dear ImGui：測試、開發用的 GUI。ImGui 表示「即時圖形使用者介面」，優點是較靈活、開發效率較高。因此本專案使用開源的 Dear ImGui 作為使用者介面的方案。
2. JSON：將使用者設定序列化為可儲存為檔案的文字格式，並以清楚的語法方便使用者直接閱讀與修改。
3. Binn：將圖形的所有資料序列化為二進位檔，節省儲存空間占用。
4. simple-svg：提供簡單的函式產生 SVG 向量圖
5. luna-svg：將 SVG 向量圖轉換為 PNG 點陣圖

(四) 版本控制：Git

Git 是一個免費、開源的版本控制系統。其最大的特色在於，透過分散式的版本控制（distributed version control），讓每位專案合作者都能在本地各自建立存放庫（repository），並且自由地進行開發、對自己的存放庫進行操作，而不需連接至中央伺服器。再者，Git 擁有非常強大、快速的分支管理、合併能力，讓開發者與他人的開發結果整合時變得非常方便。儘管 Git 也有缺點例如需要更多儲存空間，但考量到上述所列的強大的優點，以及其受歡迎的程度，還是選用 Git 作為此專案的版本控制器。

肆、研究過程及方法

一、 先備知識

(一) 圖布局演算法 (Graph Layout Algorithm)

以下介紹幾種常見的繪圖布局演算法：

1. Force-directed Graph Drawing

是一種常用且簡易的繪圖演算法。將每一個點視為一個電子，兩個距離為 d 的點之間有大小為 $f_r(d)$的排斥力，並且將每一條邊視為一個彈簧，提供兩邊的電子大小為 $f_a(d)$的吸引力，接著反覆計算每一個點受到的合力並移動，直到平衡為止。

Algorithm 1 Force-directed Graph Drawing (Kobourov, 2012)

1: **Input**: graph $G = (V, E)$
2: **Output**: the final position of each vertex
3: place each vertex randomly
4: **for** $i = 1$ to M **do**
5: **for** all vertex $v \in V$ **do**
6: calculate the net force $\overrightarrow{F}$ applied on v
7: move v by $c\overrightarrow{F}_4$
8: **end for**
9: **end for**

$f_r(d)$ 和 $f_a(d)$ 可以仿照現實中的庫侖定律和虎克定律的定義，令 $f_r(d) \propto \frac{1}{d^2}$、$f_a(d) \propto d$，也可以用其他的定義，例如 Eades (1984) 提出的：

$$f_a(d) = c_1 \log\left(\frac{d}{c_2}\right)$$

$$f_r(d) = \frac{c_3}{d^2}$$

在這裡取 $c_1 = 2$, $c_2 = 1$, $c_3 = 1$, $c_4 = 0.1$, $M = 100$ 可滿足大多數圖形繪製。

2. Orthogonal layout

在這個方法中，圖的邊必須水平或垂直於座標軸。這個方法最初是為 VLSI 和 PCB 布局問題而設計的，但它也可以適用於圖的繪製。典型的 Orthogonal layout 方法使用多階段處理，首先計算圖中的邊的交叉點，通過用節點替換交叉點來平面化輸入圖。其次計算圖中的彎曲，選擇邊緣方向以最小化彎曲。最後是將圖壓縮以減少繪圖區域，確定最終坐標。

Orthogonal layout 布局演算法非常適合中等大小的稀疏圖。 它可以生成沒有重疊、很少交叉和很少彎曲的緊湊圖。

3. Tree layout

主要用於具有唯一根元素的有向樹。從根節點開始，節點從上到下、從左到右、從右到左或從下到上排列。圖的邊可以是直線，也可以是正交方式。

Tree layout 主要為純樹設計，它也可以用於非樹，即循環圖。在這種情況下，演算法計算並使用圖的生成樹，忽略不屬於生成樹的邊來產生布局。此外，如果圖沒有連通，布局演算法會單獨處理每個 connected component 而形成樹林 (forest)。

4. Layered graph drawing (hierarchical graph drawing)

適合有向無循環(acyclic)圖或接近無循環的圖，例如軟體系統中模組或功能之間的依賴關係圖。在這個方法中，可以使用 Coffman-Graham 演算法等方法將圖的節點排列成水平階層，大多數邊從上一層連向下一層，並盡可能減少交叉。

5. Circular layout

將圖的節點放置在一個圓上，仔細選擇圓周上節點的順序以減少交叉並使相鄰節點彼此靠近。邊可以繪製為圓的弦，也可以繪製為圓內或圓外的弧線。在某些情況下，也可以使用多個圓。

6. Radial tree layout

是一個從中心向外擴展的樹結構，所有節點依照階層置於以根節點為圓心的同心圓上，同一階層的節點位於同一個圓形軌道上。由於每個軌道的長度隨著半徑的增加而增加，因此隨著樹的階層增加，節點也會有更多的空間可以擺放。

圖 2 為以上六種圖布局演算法繪製的圖，不同的布局方式適用不同類型的圖，其中 force-directed 布局沒有特殊的適用情況，較適合大部分圖論裡

見到的圖。因此，在本研究中我們先採用 force-directed 布局演算法。

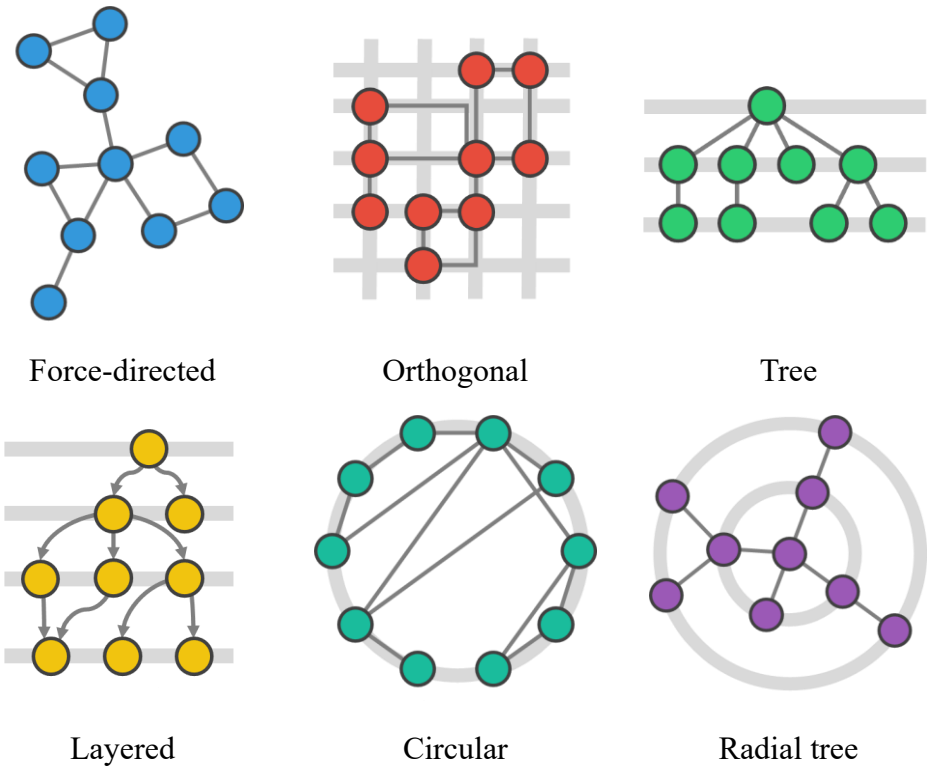


圖2. 不同布局演算繪製的圖

(二) Biconnectivity

對於一張連通無向圖 G，若不存在任何一個節點 v，使得將 v 從圖上移除後，G 會變得不連通，就稱 G 為 biconnected graph；如果存在這樣的節點 v， v 就稱為關節點（articulation vertex, cut-vertex or cutpoint）。

對於圖 G 中的點集 V' 的導出子圖（induced subgraph）是 biconnected graph，且不存在 $v \notin V'$ 滿足 $V' \cup \{v\}$ 的導出子圖是 biconnected graph，就稱 V' 為 G 的一個 biconnected component。

Biconnectivity 相關的一些性質如下：

1. 對於每一個關節點 v，都存在兩個節點 a, b，滿足 a 到 b 的簡單路徑上必有 v。
2. 兩個 biconnected component 至多共用一節點，且該節點為關節點。
3. 每個關節點必在至少兩個 biconnected component 中，非關節點必在恰一個 biconnected component 中。

要求出圖中的關節點，可以利用 Tarjan's algorithm（Tarjan, 1972）：在圖上進行 DFS，並對每一個節點 v 記錄兩個數值 $in(v)$ 和 $low(v)$，其中 $in(v)$是

時間戳記，*low*(*v*) 則是 *v* 在只能沿 DFS 生成樹上的邊往子孫節點走、再經過最多一條 back edge 的情況下，能到達的節點 *u* 中，最小的 *in(u)*。*low*(*v*) 的計算方法是，對於 *v* 的每一條鄰邊 *e*，在 *in*(*v*) 和以下取的值中取最小值：

1. 如果 *e* 是通往父節點的樹邊，則無視這條邊。

2. 如果 *e* 是通往 *u* 的 back edge，則取 *in*(*u*)。

3. 如果 *e* 是通往子節點 *u* 的樹邊，則取 *low*(*u*)。

如此一來，對於每一個關節點 *v*，都會存在一個它的子節點 *u*，滿足 *low*(*u*) ≥ *in*(*v*)。特別的是，根節點只要有子節點就會滿足這個條件，但根節點是關節點的條件是若且唯若它有至少兩個子節點。

Algorithm 2 Tarjan's Algorithm (Tarjan, 1972)

```
1:  Input: graph G = (V, E)
2:  Output: biconnected component
3:  index ← 0
4:  Stack ← empty
5:  for all vertex v ∈ V do
6:      if in(v) is undefined then
7:          BCC(v)
8:      end if
9:  end for
10: function BCC(v)
11:     in(v) ← low(v) ← index
12:     index++
13:     Stack.push(v)
14:     for all (v, u) ∈ E do
15:         if u is v's parent then
16:             continue
17:         end if
18:         if u is visited then
19:             low(v) ← min (low(v), in(u))
20:             continue
21:         end if
22:         BCC(u)
```

```
23:          low(v) ← min (low(v), low(u))
24:          if low(u) >= in(v) then
25:              repeat
26:                  t = Stack.pop()
27:                  add t to current biconnected component
28:              until t = u
29:              add v to current biconnected component
30:              output the current biconnected component
31:          end if
32:      end for
33: end function
```

(三) Block-cut tree

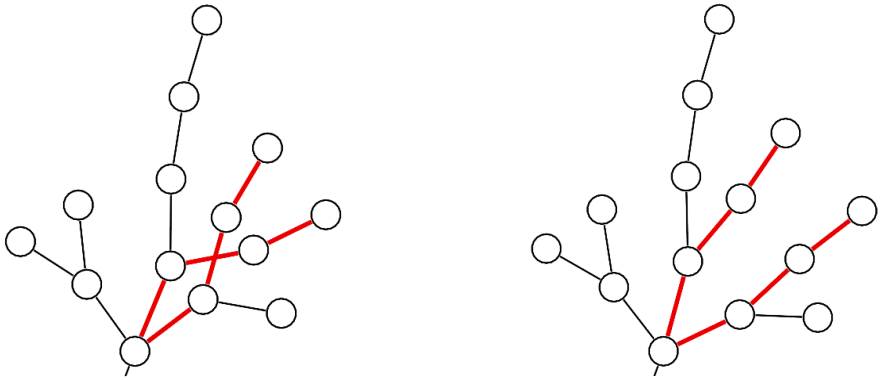
Block-cut tree 是一種根據 biconnected component 和關節點將圖轉換成樹的方式，作法是將每個 biconnected component 和關節點 視為樹上的一個節點，將每一個關節點和包含它的 biconnected component 在樹上的點連一條邊，而結果保證為一棵樹。

若要將 block-cut tree 上的節點和原圖上的節點對應，可以視為在樹上表示 biconnected component 的節點，對應到其中包含的所有非關節點，而表示關節點的節點，就對應到其代表的關節點。不過因為可能有 biconnected component 全由關節點構成，這樣在 block-cut tree 上代表它的節點就不會對應到任何節點。

二、 點的布局

要畫出一張圖最重要的就是決定每個節點的位置，在 force-directed graph drawing algorithm 中，一開始先隨機擺放每個節點的位置，再反覆計算每一個點受到的合力並移動，直到平衡為止。我們發現，最終的繪圖結果，會與一開始時節點擺放的位置有關，若節點的初始位置不佳，則可能會得到不好的繪圖布局結果，例如：交錯的情況，如圖 3(a)。

因此，我們希望在套用 force-directed graph drawing algorithm 之前，預先處理節點的初始放置位置，避免交錯，得到較佳的繪製結果，如圖 3(b)。方法說明如下：



(a) (b)

圖3. (a)交錯的樹枝 (b)沒有交錯的樹枝

(一) 基本架構

將圖內各個連通塊（connected component）各自處理，在一個 connected component 當中可能包含多個 biconnected component，在處理整個 connected component 之前，先找出圖中的每一個 biconnected component。

Block-cut tree 將每個 biconnected component 和關節點視為樹上的一個節點（稱為一個 block），並且將本來有邊直接連接的 block 所變成的節點之間加上邊形成一棵樹。在本研究的架構中，我們對 block 的定義稍做修改，這裡的 block 與原始 block-cut tree 裡的 block 不同，此處的 block 可能是

- 一個關節點（articulation）
- 一個扣除關節點的 biconnected component，可能是空的

一個 connected component 中的 block 會以 radial tree 的布局方式放在數個同心圓上，中間的 block 是根，同深度的 block 會被放在同一個圓上，如圖 4，每個節點為一個 block。

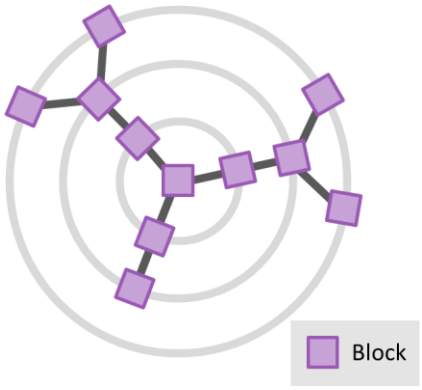


圖4. Block-cut Radial Tree 示意圖

(二) Block 內部節點配置

定義：一個 block 的寬度 $w = \begin{cases} \left\lceil \frac{size}{\pi} \right\rceil \times VC & , size \geq 2 \\ 0 & , size \leq 1 \end{cases}$，此處 *size* 是該 block 中的節點數，*VC* (Vertex Circle) 是一個常數，意義是「一個節點所佔的空間直徑」，其值必須略大於節點直徑。

Block 寬度的用途有：

- 決定所佔軌道的最小長度
- 決定 block 內節點放置的圓形直徑

block 內的節點會被放在一個圓形上，圓心是該 block 的中心，直徑是它的寬度，所有節點會等距放在圓周上，如圖 5。

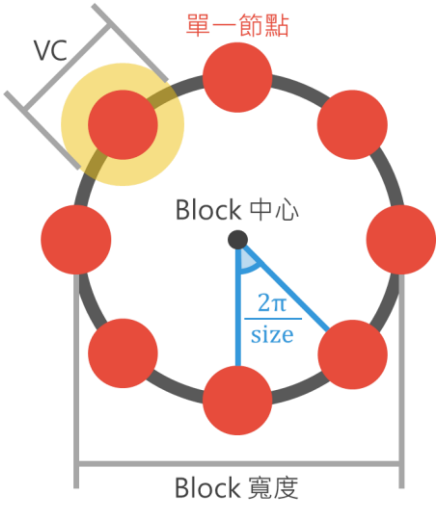


圖5. Block 內部節點配置

(三) 根的選擇

定義：兩個 block *a*, *b* 的距離是在 block-cut tree 上，*a* 到 *b* 的簡單路徑上所有經過的 block 寬度總和（含 *a*, *b* 本身），再加上 block-cut tree 路徑上的邊的數量（等於 block 數−1）乘上 *VC*。

一個 block 為根時的樹深度定義為「離它最遠的 block 距離」，而能使樹深度最小的 block 就有機會被選為根。

(四) 軌道半徑

Block-cut tree 裡同一深度的 block 會被放在同一個圓形軌道上，一個軌道有三個屬性：軌道半徑、內圍半徑、外圍半徑，內外圍半徑分別是軌道半徑減去和加上「這個軌道上的最大 block 寬度的一半」，因此內圍半徑和外

圍半徑的差剛好是最大 block 寬度。軌道的內圍半徑至少要是「內圈軌道的外圍半徑 $+ VC$」，所以軌道半徑至少是「內圈軌道的外圍半徑 $+ VC$ + 軌道上最大 block 寬度的一半」，如圖 6。

一個 block 所佔的軌道長度至少要是它的 block 寬度 $+ VC$，所以如果軌道半徑是 r、它的子樹角大小是 θ、block 寬度是 w，那麼 $r\theta \geq w + VC$。軌道半徑是滿足以上所有條件時的最小值。

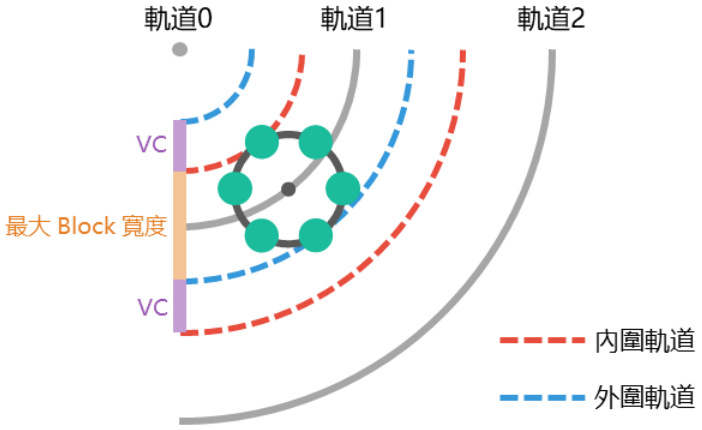


圖6. 軌道半徑

(五) 子樹的配置

定義：一個 block 的子樹寬度為

$max(VC,$ 該 block 寬度, 它的子節點的子樹寬度總和$)$。

Block-cut tree 的一個子樹在 radial tree 上的位置會在一個固定的角度範圍內，這個角稱為子樹角。一個 block 的中心位置會在該層軌道上的子樹角範圍的中點，換句話說，如果 block 所在的軌道半徑是 r、子樹角範圍是 θ_1 到 θ_2，那麼它的中心位置會在極座標 $\left[r, \frac{\theta_1+\theta_2}{2}\right]$。

一個 block 的子樹角會根據子節點的子樹寬度比例，分配給子節點，例如子樹角範圍是 $[\theta,\ \theta + \theta']$，三個子節點的子樹寬度比是 1:2:3，那麼它們的子樹角範圍會分別是 $\left[\theta,\ \theta + \frac{\theta'}{6}\right]$、$[\theta + \frac{\theta'}{6},\ \theta + \frac{\theta'}{2}]$、$[\theta + \frac{\theta'}{2},\ \theta + \theta']$。如圖 7，紅色線是紅色 block 的子樹角，藍色 block 是紅色 block 的子節點，藍色線分隔它們的子樹角。

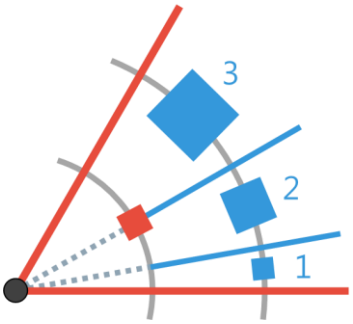


圖7. 子樹角的分配

圖 8 為一個 connected component 內部劃分示意圖，紅色節點為關節點，藍色色塊包起來的是一個 biconnected component，圖 9 中紫色色塊包起來的是與圖 8 相對應的 block。此圖中共有 8 個 block，包含 3 個關節點及 5 個不含關節點的 connected component，其中一個 block 為空的（圖 8 中橘色線框起來的 biconnected component 只包含 2 個關節點，關節點移除後，剩下空的 block）。配置後的結果顯示於圖 9。

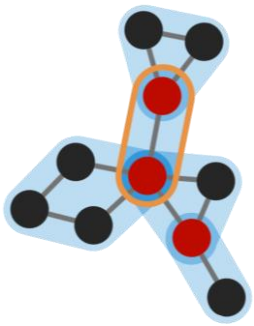


圖8. 一個 connected component 內部劃分示意圖

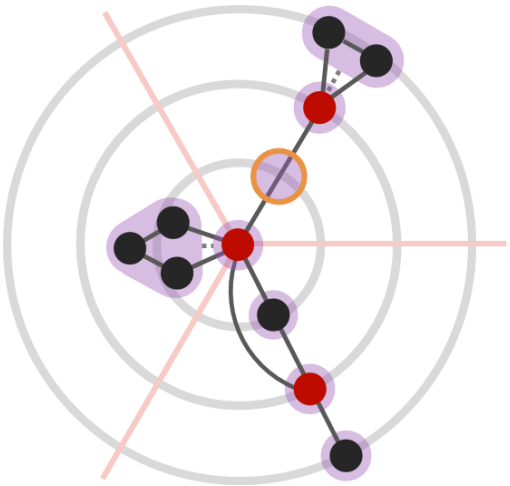


圖9. Radial tree 配置後的示意圖

三、 專案架構

我們將此專案命名為 Graphene，以下為 Graphene 的架構：

(一) Core：紀錄圖形、進行繪圖演算法等。可接受檔案與指令的輸入，適合與其他程式整合，省去繁瑣的檔案管理、圖形顯示等。

1. Graph：保存圖形的結構、轉換圖形形式，使用唯一辨識碼 UUID 進行存取，作為文字指令與內部物件的橋樑。

4. Structure：同 Graph 一樣紀錄圖形的結構，但是以快速的資料結構與指標形式儲存，以利繪圖演算法以最快效率執行。

5. Properties：紀錄所有節點與邊的性質，如位置、受力的大小、標籤、顏色、大小等等。

6. Placement：使用繪圖演算法配置點和邊的位置。

7. Interface：接受指令輸入，並串流圖形結構、性質給編輯器。

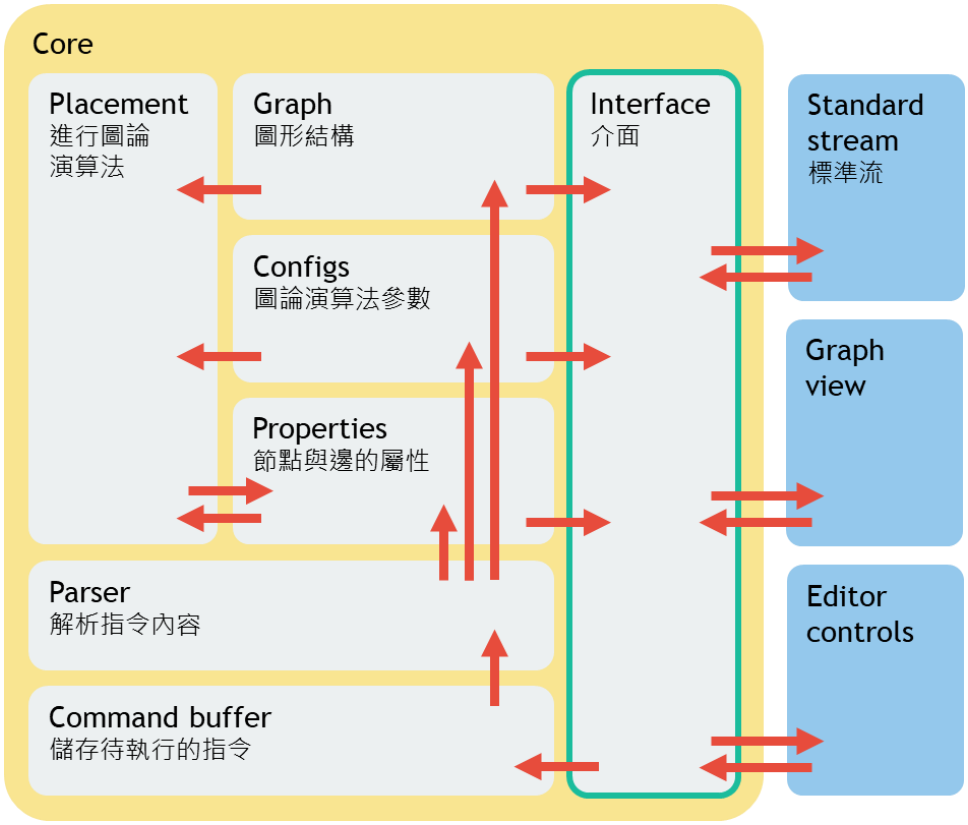


圖10. Core 架構圖

(二) Editor：編輯器，包含檔案管理、圖形及時渲染、參數修改的介面等，為所有功能的使用者圖形介面。

1. Document：Graphene 裡的檔案物件。Editor 中有一個 Terminal document 與一般的 document。Terminal document 為終端機中預設的對象，在想要快速的將自己的程式視覺化為圖形的時候比較方便，省去給定檔名、存檔的麻煩。每個 Document 都持有一個 Core 的物件。

2. Preferences：整個編輯器的偏好設定管理，包含各種動作對應的快速鍵等，方便讓使用者調整為最適合自己的操作邏輯。

3. Graphics 與 Theme：底層視窗管理等。

伍、成果與討論

本研究的構想主要是希望在學習圖論演算法時能夠輕易的繪製出所需的圖，無須太多的動作，可以很直覺的操作，並能夠在點或邊異動時，即時更新布局，讓使用者立刻看到結果。

一、 Graphene 使用者介面

使用者介面的設計與預期使用情形的需求有很大的關係，以下是我們的繪圖程式 (Graphene) 的介面設計：

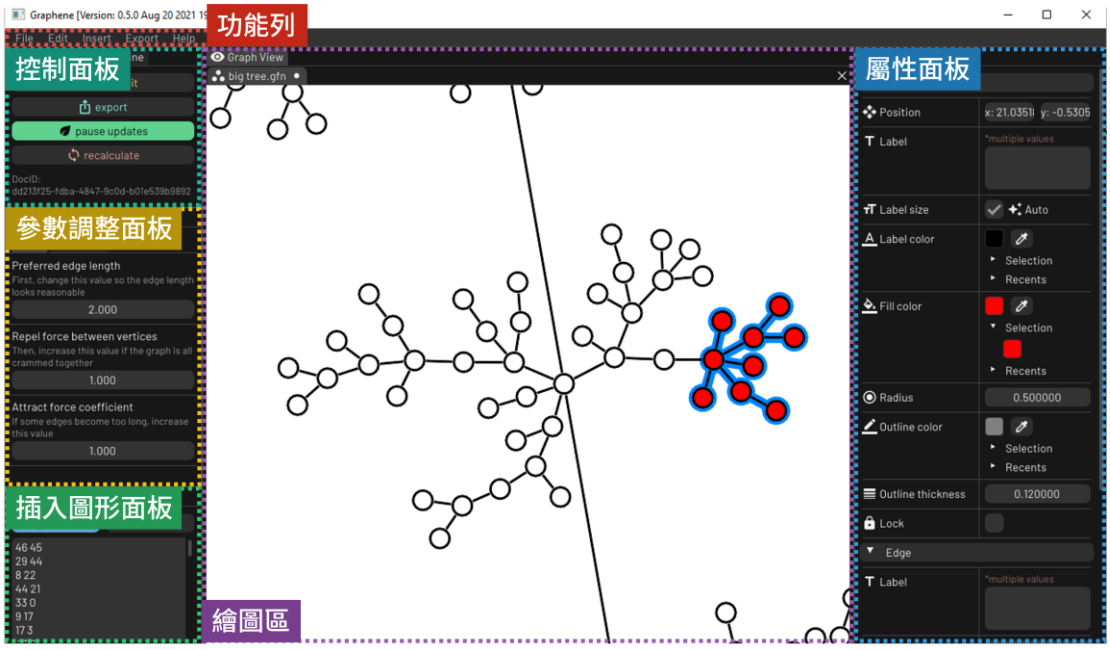
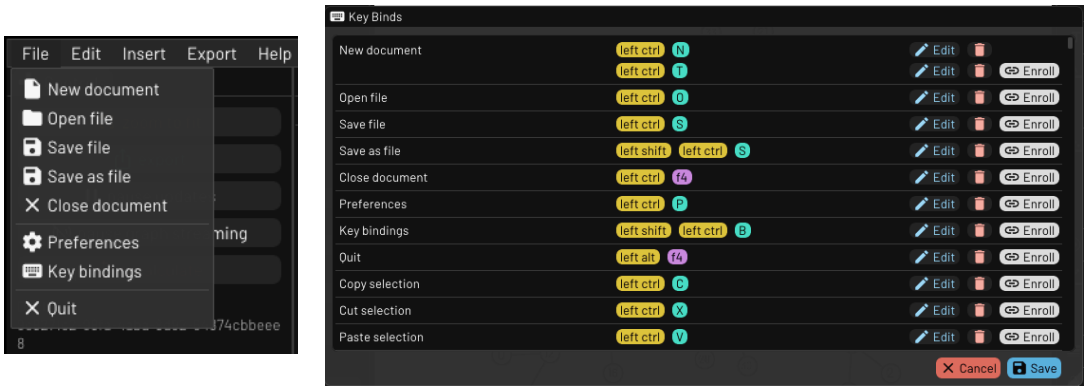


圖11. Graphene 介面設計

(一) 功能列

檔案操作、編輯、插入、匯出圖形、自訂快捷鍵等功能。Graphene 中的每

一個按鍵配置都可以自由修改，讓使用者根據自己的使用習慣調整操作邏輯，讓 Graphene 成為更使用者友善的程式。



(a) (b)

圖12. (a) 檔案功能列 (b) 快捷鍵設置

(二) 繪圖區

提供使用者繪製圖形，有以下幾個功能：

1. 新增節點：按住鍵盤 ctrl 鍵，同時使用滑鼠左鍵按一下繪圖區空白處即可新增一個節點。
2. 新增邊：按住鍵盤 ctrl 鍵，滑鼠從一個節點拖曳至另一個節點即可新增一條邊，連接到同一個節點可新增自環，並且支援重邊。
3. 放大或縮小：滾動滑鼠滾輪即可將圖放大或縮小。
4. 移動畫面範圍：按住滑鼠右鍵並移動滑鼠即可平移畫面範圍。
5. 選取節點或邊：在節點或邊上點一下即可選取該節點或邊，按住 shift 可選取多個節點或邊。此外，若要選取一群節點和邊，可自由圈選要選取的區域，不受選取區域的形狀限制，比常見的長方形選取範圍更方便，如圖 13。
6. 移動節點位置：選取要移動的節點，按住滑鼠左鍵拖曳即可移動節點的位置。
7. 刪除節點或邊：選取要刪除的節點或邊後按下 delete 鍵。
8. 複製與貼上：選取要複製的節點與邊後按下 ctrl＋c，接者可以按下 ctrl＋v 在繪圖區貼上。另外，複製後可以直接以文字形式（點及邊的數目與每個邊的端點編號格式，如圖 1 的格式）在任何地方貼上，方便將 Graphene 中繪製的圖作為其他程式之輸入，可大大加速在解題時自己產生測試資料進行偵錯等。（其他貼上功能見下面的「插入圖形面板」）。

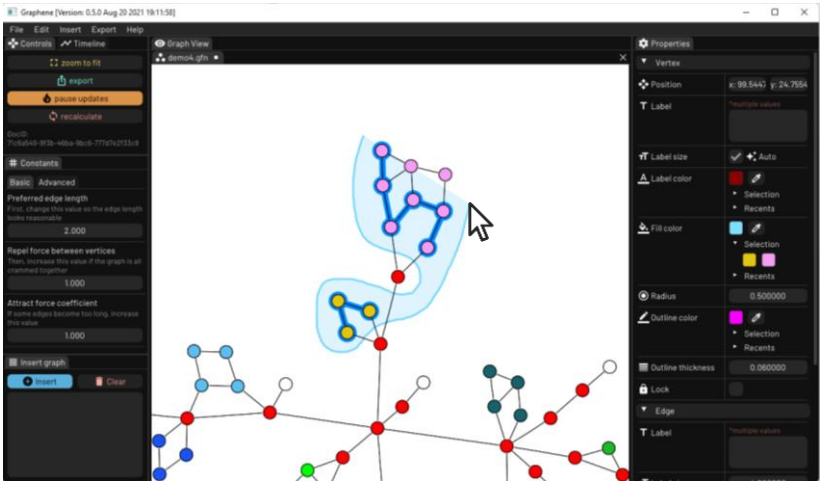


圖13. 自由圈選節點及邊

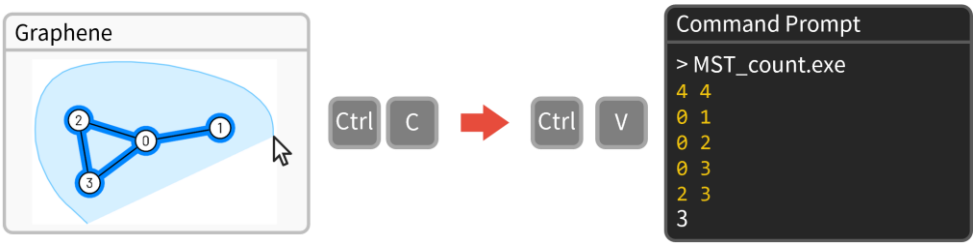
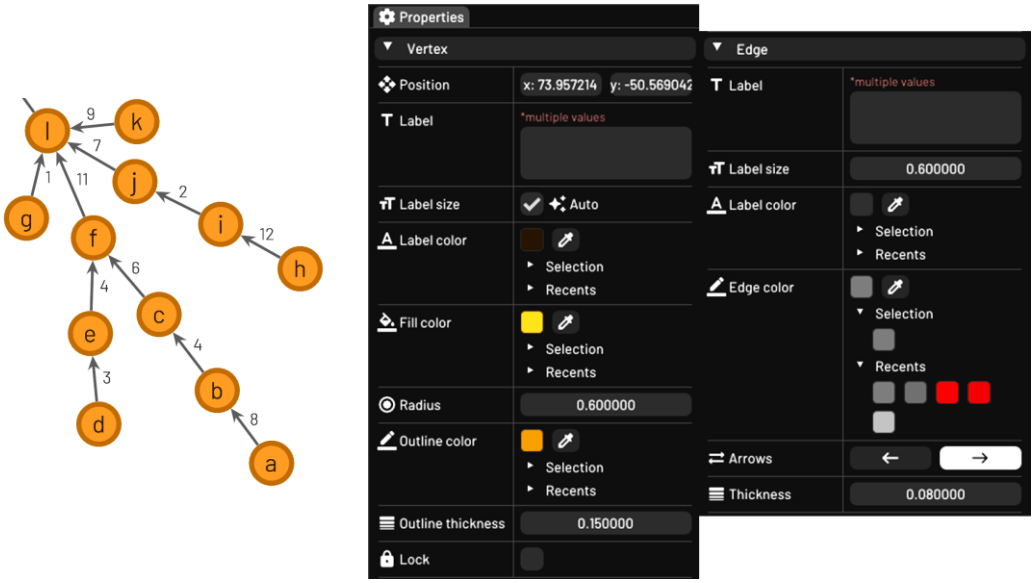


圖14. 以文字形式貼上 Graphene 圖形功能示意圖

(三) 屬性面板

選取繪圖區中的節點或邊，即可在屬性面板設定其屬性。

1. 節點：可設定節點的位置、標籤文字、文字大小、文字顏色、節點顏色、半徑、框線顏色及粗細，並可鎖定節點位置。
2. 邊：可設定邊的標籤文字、文字大小、文字顏色、邊的顏色、邊的方向及邊的粗細。



(a) (b)

圖15. (a) 屬性設定效果 (b) 屬性設定面板

(四) 控制面板

提供使用者常用的控制功能，包括：

1. zoom to fit：將畫面縮放，使整個圖形包含在畫面上。
2. export：匯出圖形為 TikZ picture、SVG 或 PNG 格式。

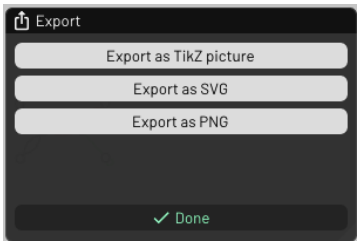


圖16. 選擇匯出格式

編撰講義的插圖可以使用 Graphene 設計，由電腦計算並繪製可讀性較高的圖，並可根據自己的需求自訂圖形樣式，方便教學者設計教材。因此除了將繪製結果即時更新顯示在視窗上，我們也能把圖形匯出，作為講義的附圖等。而以下是 Graphene 支援的匯出格式：

(1) TikZ picture：TikZ 為 LaTeX 中用於繪製圖形的套件。在 LaTeX 中以 TikZ picture 插入 Graphene 的圖形更方便、快速。

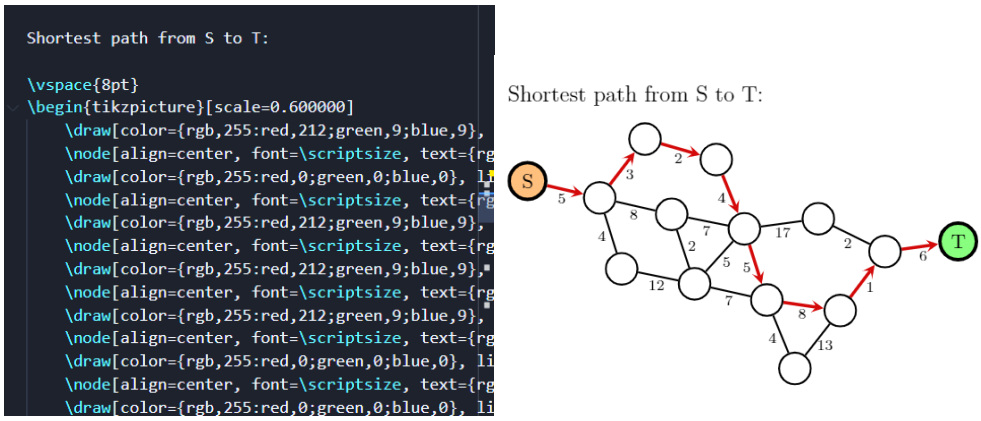


圖17. TikZ 輸出結果

(2) SVG 圖片：向量形式的圖片格式，可隨意縮放不損失細節。

(3) PNG 圖片：適合講義、文件、簡報與網站的圖片格式。

3. pause updates：暫停套用布局演算法。

在手動繪圖一開始加入邊的過程中，若即時套用 force-directed graph drawing 布局演算法，會使節點在每次操作時都會移動位置，影響使用者的操作。因此，可先暫停布局演算法，待完成基本邊的加入再套用以得到布局後的結果。

4. recalculate：重新初始化節點位置。

節點的初始位置會影響布局結果，因此若對生成的圖形不滿意，或是圖形有所異動，可選擇 recalculate，將節點依 Graphene 布局架構重新排列，設定其初始位置並套用繪圖演算法計算繪圖結果。

(五) 參數調整面板

利用參數調整面板可以調整各種參數以改變節點和邊的位置，並可即時顯示於繪圖區。參數調整有基本和進階二種方法：

1. 基礎：將 force-directed graph drawing 的參數簡化為邊的偏好長度、節點之間的排斥力與邊的吸引力，並說明應如何進行調整。

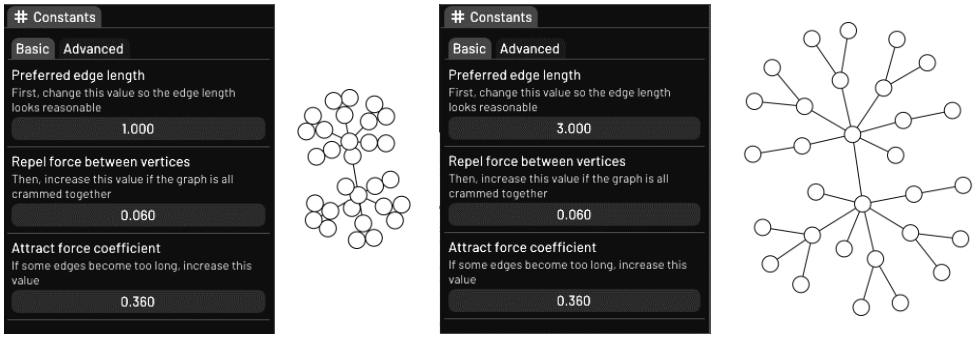
說明：邊的偏好長度：首先調整這個數值使邊的長度看起來合理。

節點之間的排斥力：接著，若圖形都擠在一起則提高這個數值。

邊的吸引力：如果有些邊變得過長，提高這個數值。

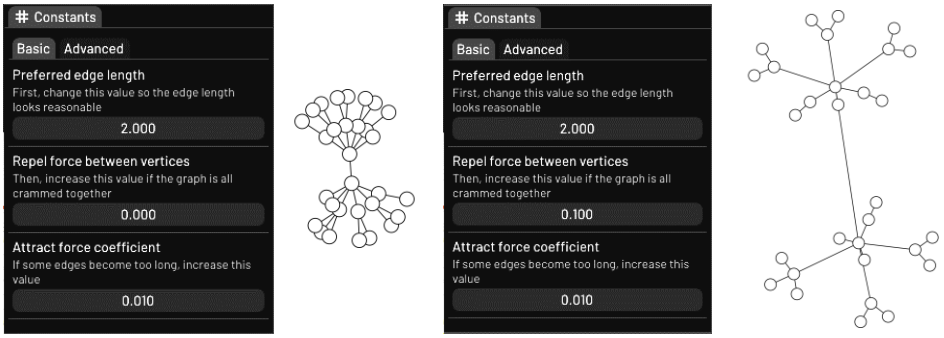
2. 進階：Force-directed graph drawing 演算法使用的常數調整。

圖 18、圖 19 與圖 20 分別為三個基本參數（邊的偏好長度、節點之間的排斥力、邊的吸引力）調整面板及調整前後的對照。圖 21 為進階參數調整面板。



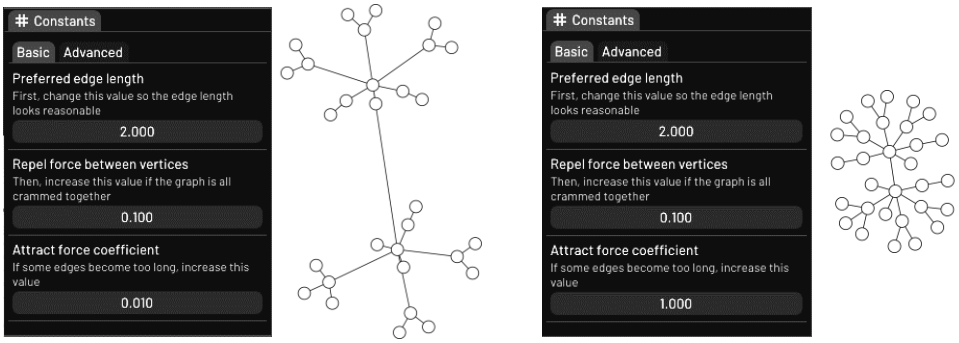
(a)邊長為 1　　(b)增加邊長為 3

圖18. 邊的偏好長度調整



(a)無排斥力　　(b)增加排斥力為 0.1

圖19. 節點之間的排斥力調整



(a)吸引力為 0.01　　(b)增加吸引力為 1

圖20. 邊的吸引力調整

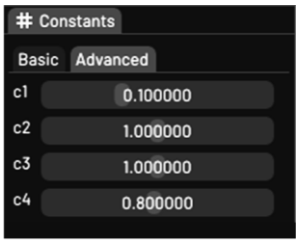


圖21. 進階參數調整

(六) 插入圖形面板

如圖 22（上），可直接在此輸入節點和邊的數量，接著輸入每個邊的端點，同程式競賽題目中常見的輸入格式，如圖 1，即可插入圖形。另外，也可以以文字複製上述格式的資料後直接在 Graphene 繪圖區按下 ctrl + v 貼上，如圖 22（下）。

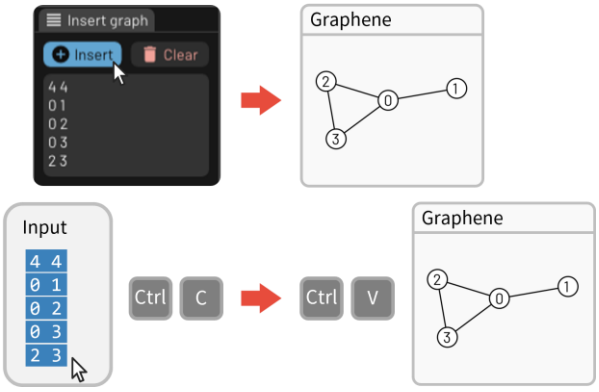


圖22. 文字形式圖形輸入功能示意圖

(七) 時間軸

學習圖論演算法最需要的就是視覺的回饋，很多概念是圖一畫出來便能顯著幫助理解的，因此在學習圖論演算法的過程中，Graphene 會是非常有幫助的工具。我們利用時間軸的功能紀錄一連串圖形的變化，幫助演算法學習，將每個演算法的步驟完整演繹。在執行演算法程式時，可以在程式碼中添加一些輸出指令，並在執行時將指令重新導向給 Graphene。Graphene 會把過程記錄下來，使用者就可以利用 Timeline 功能觀看圖形變化，或一步一步進行分析。此功能可讓教師演示演算法執行步驟，使學生學習圖論演算法時能夠清楚的了解其過程，或者用於記錄並展示學生自己撰寫的程式執行步驟，便於除錯，對學習圖論演算法有很大的幫助。

圖 23 為利用時間軸功能呈現 Bellman-Ford 最短路徑演算法的每一個步驟，以節點 0 為起點。學習者可利用時間軸功能一步一步觀看圖形，了解演算法的執行過程。圖 24 為加入了 Graphene 指令以實現圖 23 的程式碼片段。

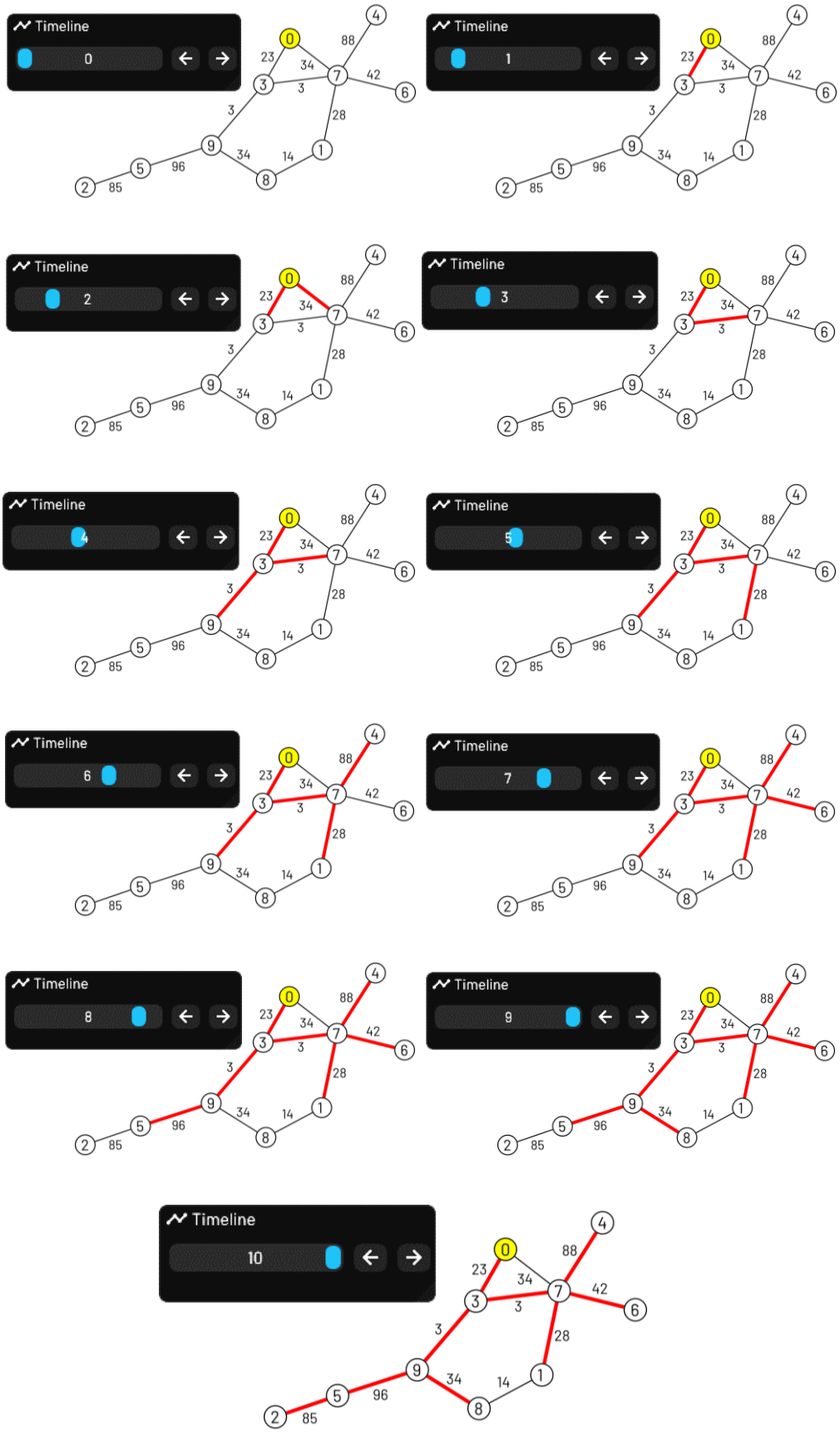


圖23. 時間軸－以 Bellman-Ford 最短路徑演算法為例

```cpp
for (int i = 0; i < v - 1; i++) {
    for (int a = 0; a < v; ++a) {
        for (int b = 0; b < v; ++b) {
            if (dist[a] != 1e9 && adj[a][b] != 1e9 && dist[a] + adj[a][b] < dist[b]) {
                dist[b] = dist[a] + adj[a][b];
                std::cout << "setedgeprops -name=" << min(parent[b], b) << "," << max(parent[b], b)
                          << " -key=edgeColor -value=rgb(0,0,0)\n";
                std::cout << "setedgeprops -name=" << min(parent[b], b) << "," << max(parent[b], b)
                          << " -key=thickness -value=0.06\n";
                // 設定非最短路徑邊為黑色，粗細為預設0.06
                parent[b] = a;
                std::cout << "setedgeprops -name=" << min(a, b) << "," << max(a, b)
                          << " -key=edgeColor -value=rgb(255,0,0)\n";
                std::cout << "setedgeprops -name=" << min(a, b) << "," << max(a, b)
                          << " -key=thickness -value=0.2\n";
                // 設定最短路徑邊為紅色，粗細為加粗0.2
                std::cout << "timeline -new\n";
                // 在時間軸中加入一幀
            }
        }
    }
}
```

圖24. Bellman-Ford 的程式碼片段

(八) 終端機模式

Graphene 除了圖形介面之外，也提供終端機模式，進階使用者可根據自己的需求，在終端機模式中下指令，結果會即時顯示於繪圖區，如圖 25。

終端機模式允許使用者的程式透過標準輸入輸出流對 Graphene 進行操作。這讓演算法程式視覺化更即時，更彈性，也可與時間軸功能結合，紀錄使用者程式的執行過程的每一步。終端機模式也支援存檔與讀檔，可以系統化的批量產生精美的圖形，提供無限可能。如圖 25，左邊為終端機畫面，指令依序為「新增節點，名字為 1」、「新增節點，名字為 2」、「新增邊，連接名字為 1 與 2 的節點」、「將名字為 1 的節點的填充顏色設為青色」，右邊為即時的執行結果。

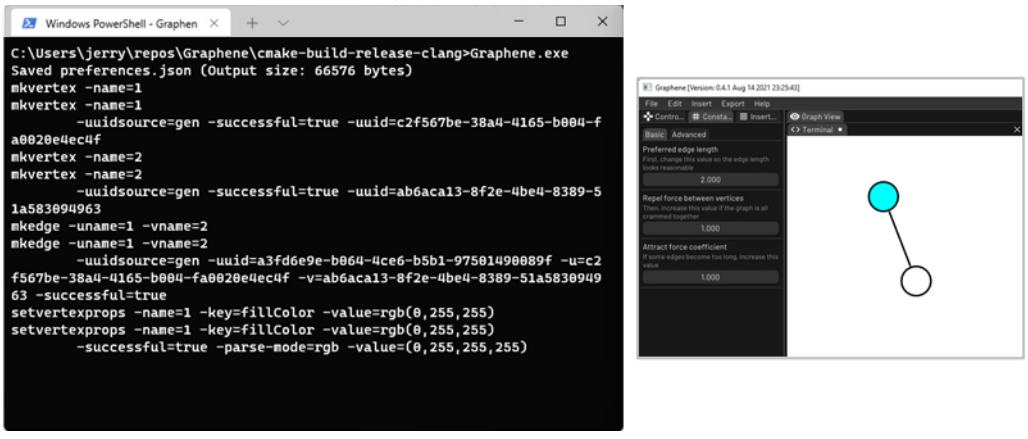


圖25. 終端機模式

二、 Graphene 基本繪圖結果

(一) 無向圖

無向圖是指一個圖的所有邊都是沒有方向性的，也就是邊的二端點之間是雙向互通的。Graphene 預設產生無向圖，可在繪圖區利用 Ctrl 鍵及滑鼠操作新增節點及邊，也可在文字圖形輸入區輸入資料以產生圖。節點可根據需求設定標籤文字、大小、顏色及框線粗細，邊也可以設定顏色、粗細。如對布局結果不滿意，也可在參數設定區調整邊的長度及作用力的大小。

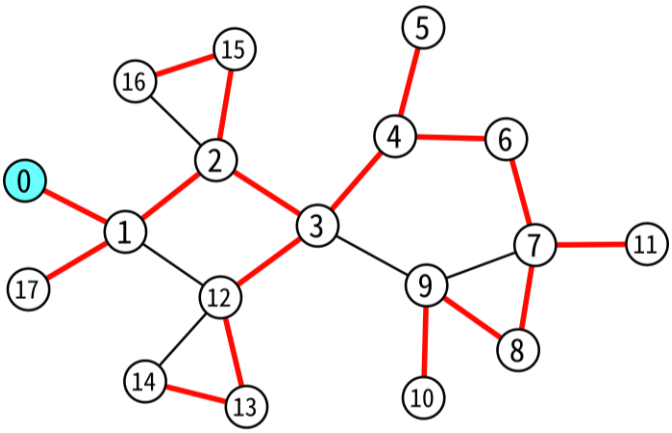


圖26. 無向圖

(二) 有向圖

有向圖是指圖中具有方向性的邊，以具有箭頭的邊來表示其方向。要繪製有向圖時，可在加入無向邊後，選取該條邊，或一次圈選多條邊，於屬性設定區設定箭頭的方向，同樣的，也可以設定邊的粗細、顏色等。

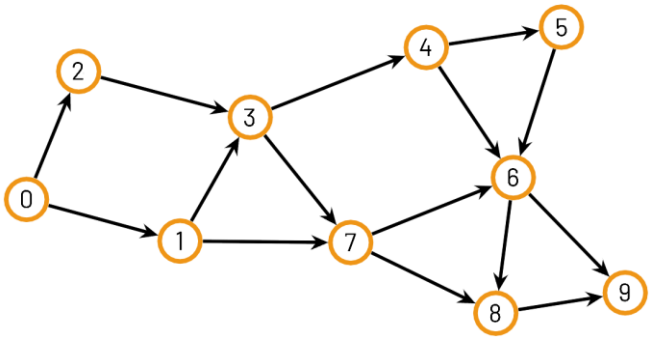


圖27. 有向圖

(三) 帶權圖

邊加上權重，代表兩點之間的關係，例如：距離、時間、成本等。具有權重的圖應用在很多圖論演算法中，如最小生成樹、最短路徑等都需要用到

權重。在 Graphene 中要在邊上加上權重，只要選取該條邊，於屬性設定區設定標籤的值即可。

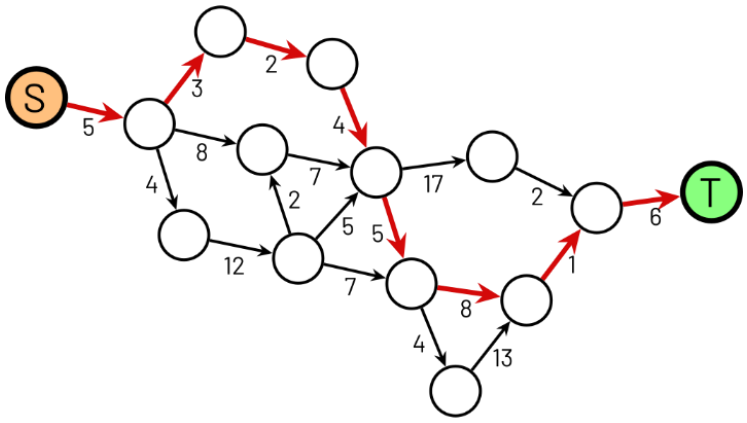


圖28. 具有權重的有向圖

(四) 具有重邊或自環的圖

在圖論中，重邊（multiple edges）是兩條或多條與同一對節點相連接的邊，自環（loop）則是一條節點與自身連接的邊。Graphene 除了用來繪製簡單圖外，也可繪製具有重邊或自環的圖。只要在節點間重複拖曳，便可以產生重邊，重邊中的每一條邊都可設定它專有的屬性。若滑鼠拖曳的起點和終點為同一個節點，則可繪製自環，自環同樣可以設定它的屬性。

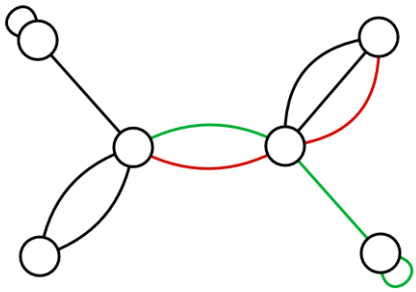


圖29. 具有重邊及自環的圖

(五) 複雜的圖

Graphene 能完整利用電腦硬體的效能，支援將計算節點位置的工作分配給多個執行緒，使 Graphene 能輕鬆應付大型的圖。圖 30 是 Sierpiński triangle 的繪製結果。(7 階，1095 個節點，2187 條邊)

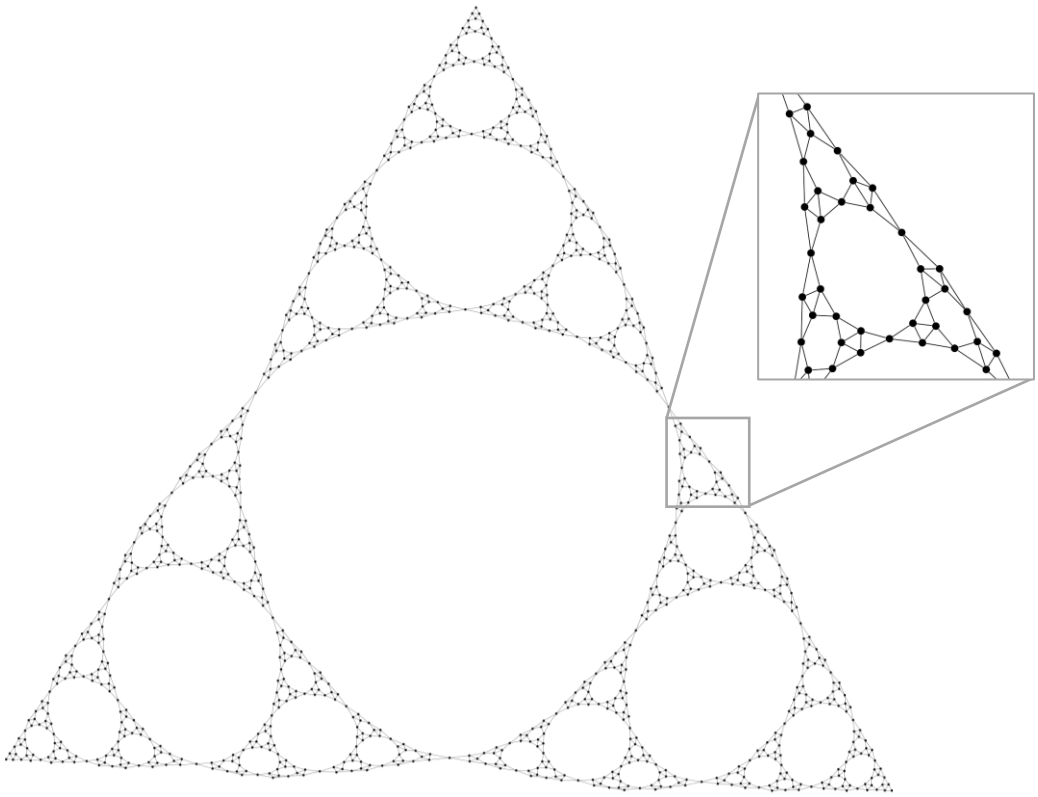
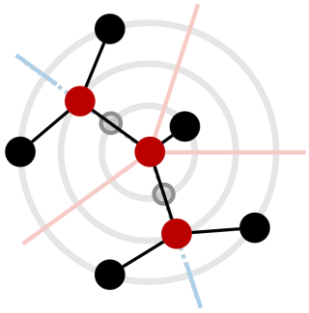


圖30. Sierpiński triangle 繪製結果

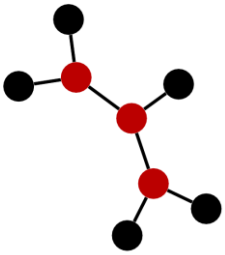
三、 Graphene 的布局

根據 Graphene 的基本架構，先找出 biconnected component 和關節點，計算每個 block 的寬度，以 block-cut tree 的概念及 radial tree 的布局方式，安排每個節點的初始位置，再套用 force-directed drawing algorithm 施力於各節點，達到平衡後即產生最終繪圖結果。圖 31(a)、(c)為 Graphene 初始節點配置的實際繪製結果，每個 block 依照子樹角分配於 radial tree 軌道上。圖 31(b)、(d)為圖 31(a)、(c)套用 force-directed drawing 後的最終布局結果。分別說明如下：

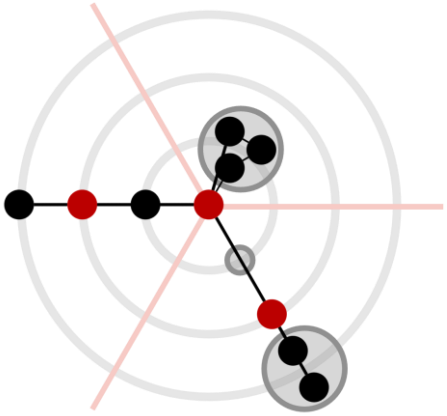
圖 31(a)、(b)為一棵樹（tree），紅色節點為關節點，圈起來的部分為 block。從 0°起算，根節點的 3 個子樹寬度分別為 1:2:2，因此子樹角分別為 72°，144°，144°，其範圍分別為[0°, 72°], [72°, 216°], [216°, 360°]。以左上方關節點為例，位於 72°和 216°中點 144°的軌道上，它的二個子樹寬度為 1:1，因此[72°, 216°]再被分為二等分：[72°,144°], [144°, 216°]，該二子樹分別置於 108°與 180°的軌道位置上，如圖 31(a)。圖 31(c)、(d)為一類樹圖（tree-like graph），同樣的，紅色節點為關節點，圈起來的部分為 block。從 0°起算，根節點的 3 個子樹寬度分別為 1:1:1，因此子樹角皆為 120°，節點配置如圖 31(c)。執行結果與我們的預期相符（圖 8、圖 9）。其餘較複雜的圖的初始節點配置與最終布局的實際執行結果請參考附錄。



(a) 初始節點配置 (Tree)



(b) 最終結果 (Tree)



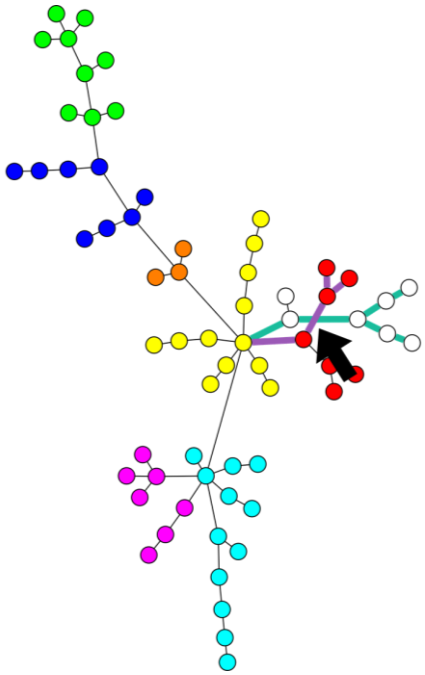
(c) 初始節點配置 (Tree-Like Graph)

(d) 最終結果 (Tree-Like Graph)

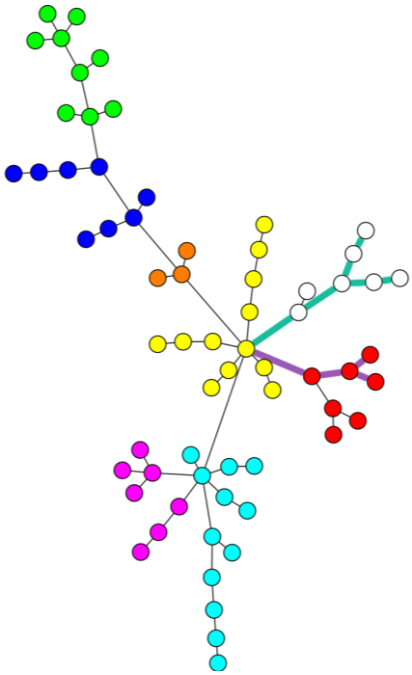
圖31. Graphene 布局結果

四、 優化結果

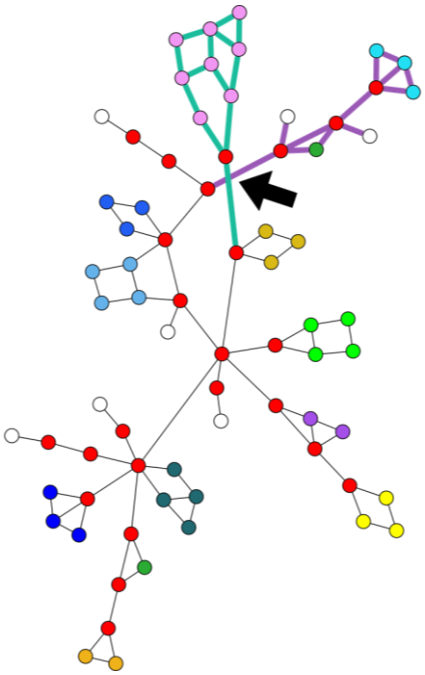
在樹(tree)及類樹圖(tree-like graph)的繪製中，如果僅使用 force-directed drawing algorithm，有時會產生樹枝交錯的情況，如圖 32 (a)、(c) 箭頭所指之處。我們利用修改版 block-cut tree 的概念，搭配 radial tree 的布局方式先行處理，可以改善交錯的情況，如圖 32 (b)、(d) 所示。在圖 32 (c)、(d)中，紅色節點表示關節點，屬於同一個 block 的點以相同顏色標記。



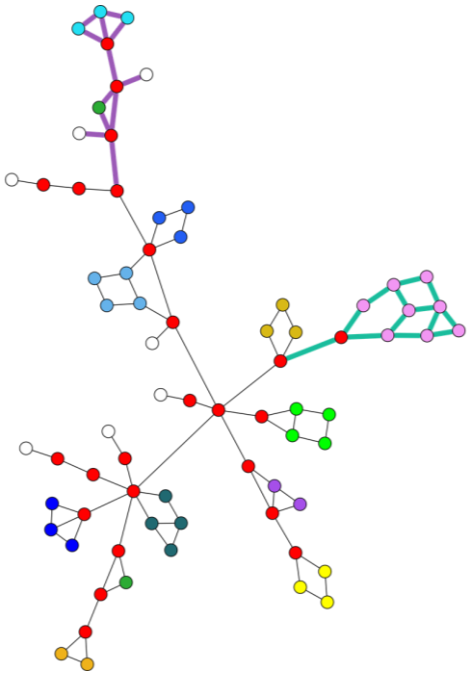
(a) 僅用 Force-direct (Tree)



(b) Graphene 優化 (Tree)



(c) 僅用 Force-direct (Tree-Like Graph)



(d) Graphene 優化 (Tree-Like Graph)

圖32. 優化結果

陸、結論及應用

本研究針對學習圖論演算法的需求，設計一套使用者友善的繪圖軟體 Graphene。Graphene 繪圖程式除了能解決手繪的缺點，提供高可讀性的繪製結果，作為輔助繪圖的工具外，也加入時間軸、標記、自訂外觀等功能，用最直覺的方式幫助學習者理解與研究圖論演算法，實作圖論演算法時能更清楚其背後的原理，並提升除錯效率。Graphene 針對演算法學習者及教學者與演算法競賽選手，設計常用功能，可直接輸入競賽題目的文字格式測試資料產生繪圖結果，並結合現有繪圖演算法，改善、優化圖的繪製結果，貼近使用者需求。此外，Graphene 也提供匯出成圖片的功能，使其成為製作演算法講義附圖的利器，幫助教師製作教材，有助於圖論演算法教學。

Graphene 採用的繪圖演算法以 force-directed graph drawing 演算法為基礎，實作節點的分布。然而初始的節點分布會影響繪圖結果，因此，我們利用 biconnected component、block-cut tree 等圖論結構對圖的繪製進行優化。首先找出圖的 biconnected component 及關節點，重新定義 block-cut tree 裡的 block，計算出 block 的寬度及子樹角，利用 radial tree 的布局方式將每個 block 依照其子樹角，分布在同心圓軌道上。接著再套用 force-directed graph drawing 演算法，得到最後的布局結果。如此可以減少不同 block 之間的交錯，得到較佳的結果。

雖然 Graphene 沒有像一些現有的繪圖軟體一樣，提供多種節點形狀、連線形狀或許多不同的布局方式，但是我們以學習者的角度出發，設計提供了學習圖論演算法所需要的各種功能，希望能幫助演算法的學習者與教學者，讓更多人加入學習演算法的行列，得到更好的學習成效。

柒、參考資料

[1] Kobourov, S. G. (2012). *Spring Embedders and Force Directed Graph Drawing Algorithms*. Retrieved from https://arxiv.org/pdf/1201.3011.pdf

[2] Eades, P. (1984). A heuristic for graph drawing. *Congressus numerantium, 42*, 149-160.

[3] Kulli, V. R., & Biradar, M. S. (2014). The Point Block Graph of A Graph. *International Journal of Mathematics and Computer Science, 5* (5), 476-481.

[4] Tarjan, R. (1972). Depth-First Search and Linear Graph Algorithms. SIAM *Journal on Computing, 1* (2), 146-160.

[5] Pavlo, A. (2006). Interactive, tree-based graph visualization.

[6] Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms, third edition*. Massachusetts: MIT Press.

[7] Block forest, https://oi-wiki.org/graph/block-forest/

Dependencies

liteserver/binn: Binary Serialization (github.com)

mapbox/earcut.hpp: Fast, header-only polygon triangulation (github.com)

Dav1dde/glad: Multi-Language Vulkan/GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs. (github.com)

glfw/glfw: A multi-platform library for OpenGL, OpenGL ES, Vulkan, window and input (github.com)

ocornut/imgui: Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies (github.com)

aiekick/ImGuiFileDialog at 7b96209e5e96caff9cfeaf3e67dc7cb7d686ae74 (github.com)

nlohmann/json: JSON for Modern C++ (github.com)

sammycage/lunasvg: lunasvg is a standalone SVG rendering library in C++ (github.com)

adishavit/simple-svg: Easy to use SVG library for C++ (fork of legacy Google code project archive) (github.com)

nothings/stb: stb single-file public domain libraries for C/C++ (github.com)

mariusbancila/stduuid: A C++17 cross-platform implementation for UUIDs (github.com)

progschj/ThreadPool: A simple C++11 Thread Pool implementation (github.com)

捌、附錄

Graphene 實際執行結果：

(一) 樹

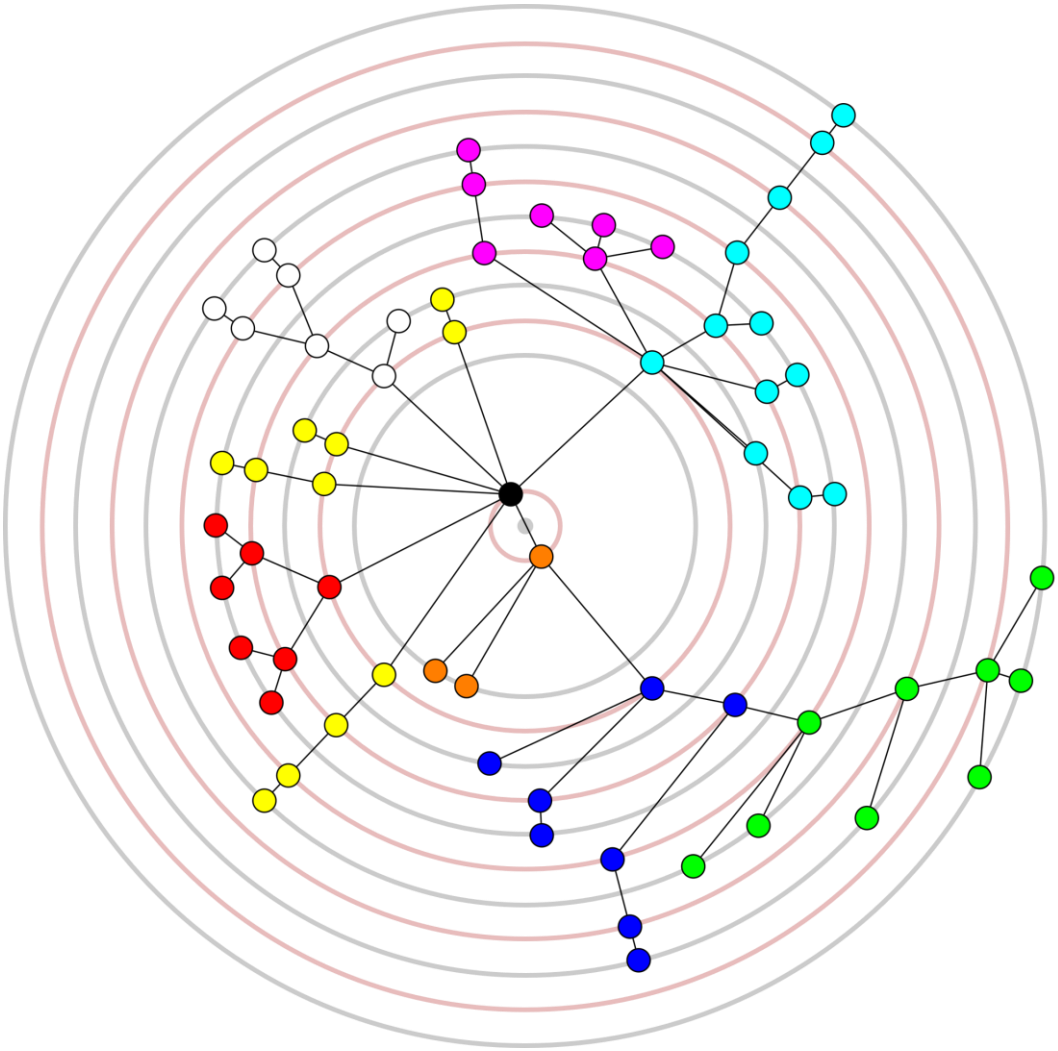


圖33. 樹的初始節點配置

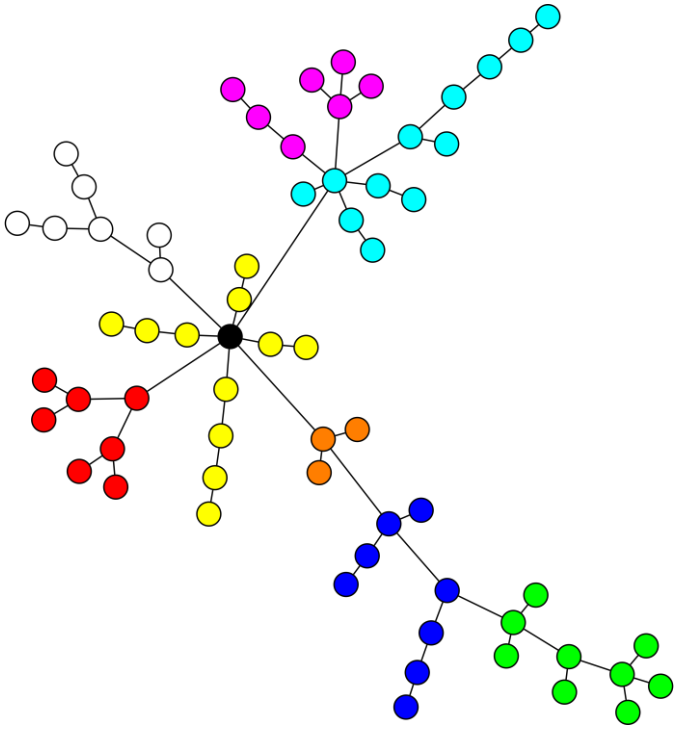


圖34. 圖 33 的繪製結果

(二) 類樹圖

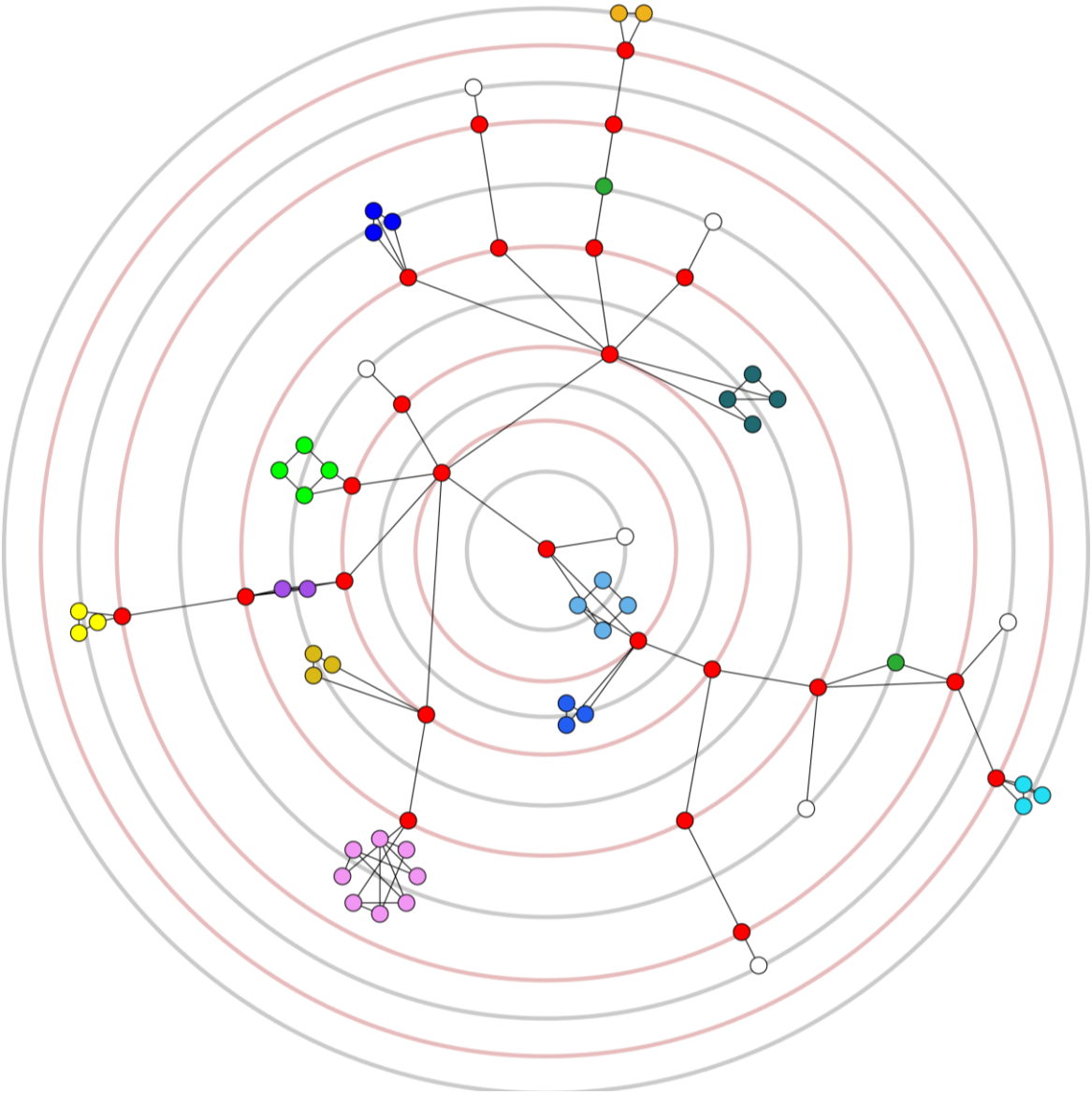


圖35. 類樹圖的初始節點配置

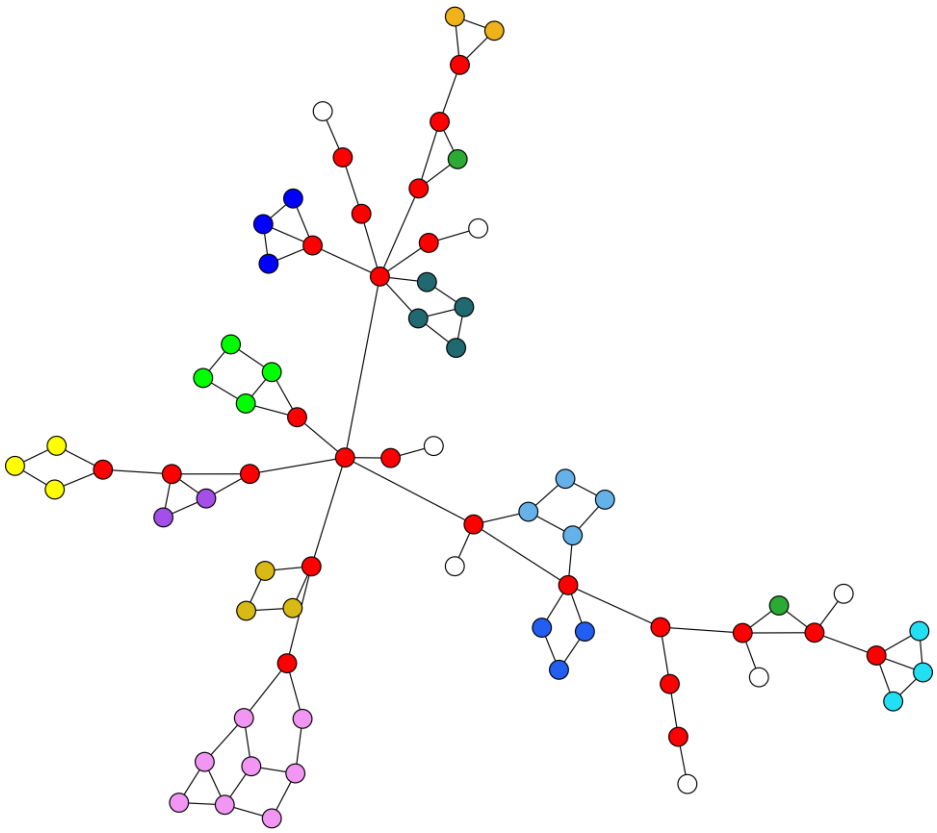


圖36. 圖 35 的繪製結果

(三) 複雜的樹(1025 個節點)

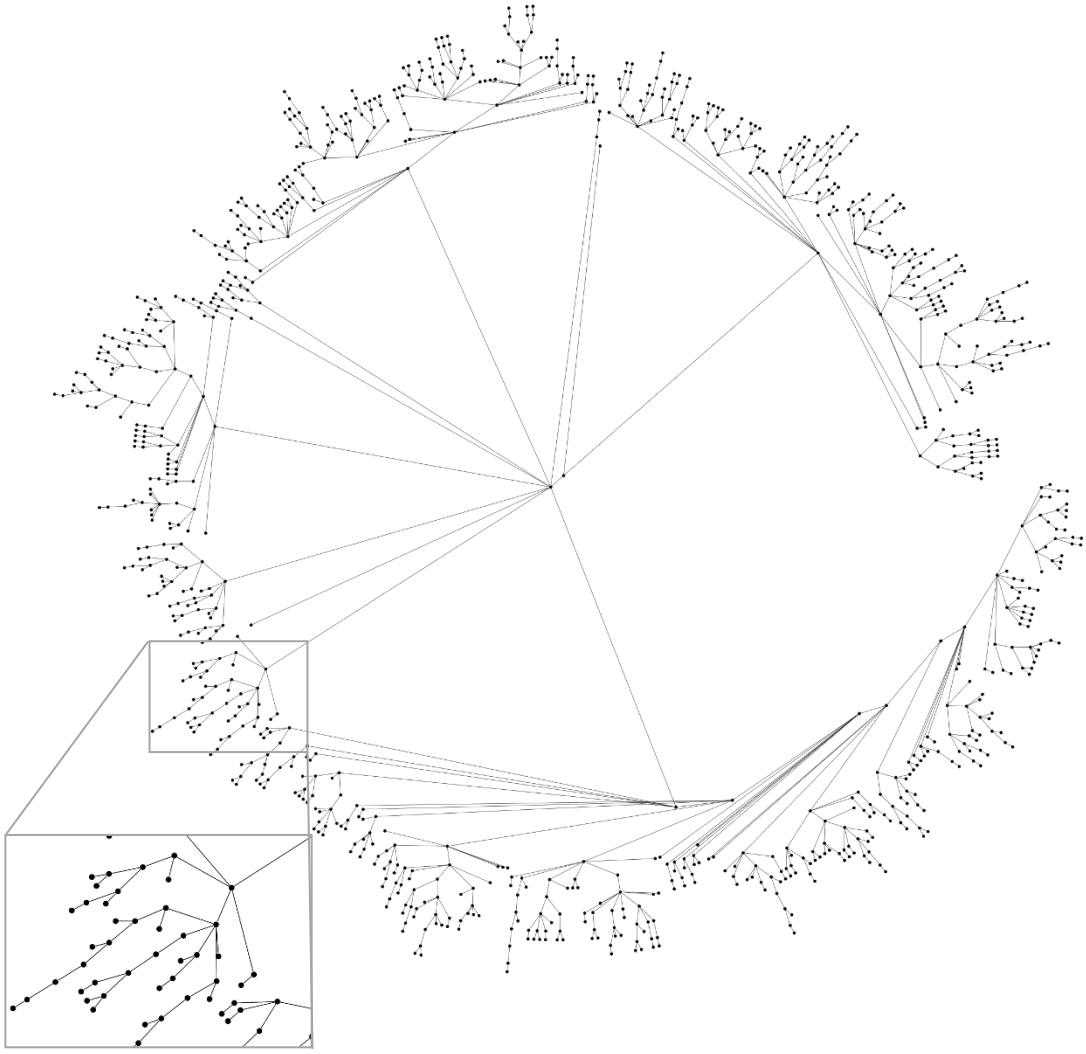


圖37. 複雜樹的初始節點配置

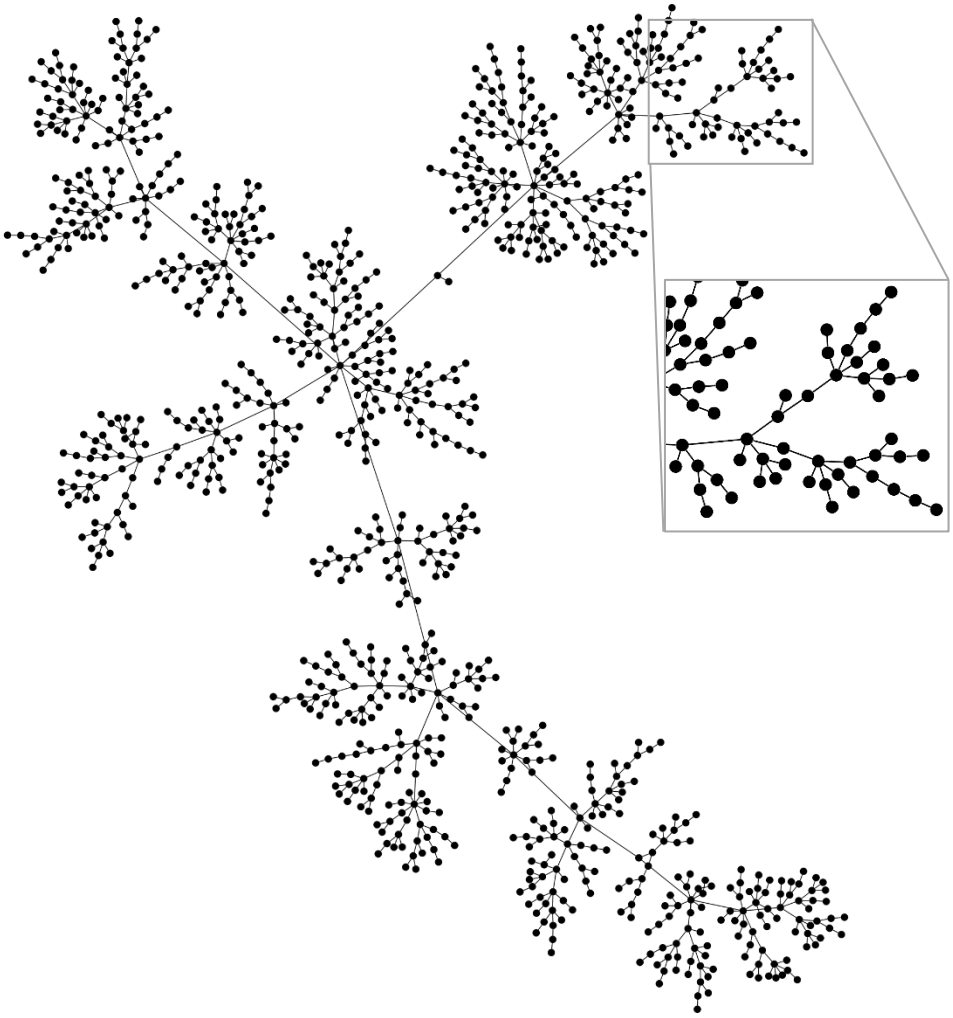


圖38. 圖 37 的繪製結果