國　立　交　通　大　學

工業工程與管理學系

碩士論文

斐氏圖軟體撰碼問題中自動規範測試系統的設計與實作

Design and Implementation of Automated Conformance Testing System in Petri-Net-Based Software Coding Problem

研究生：林潔妤

指導教授：梁高榮 博士

中華民國九十七年六月

斐氏圖軟體撰碼問題中自動規範測試系統的設計與實作

研究生：林潔妤　　指導教授：梁高榮　博士

國立交通大學工業工程與管理學系

摘要

對自動化製造系統而言，發展它的控制軟體過程可分為設計與實作兩階段。對設計階段而言，連續兩製造狀態的因果關係可用斐氏圖來建模，而所有可到達狀態的集合則稱為設計可達圖。相似地，對實作階段而言，對應的斐氏圖可透過程式語言來撰寫，並自然產生其實作可達圖。又測試設計可達圖與實作可達圖的等效性是必要的。換言之，設計可達圖可用來產生測試序列並用來測試實作可達圖的規範。

本研究中，透過設計行為的可達圖產生一條測試序列。假如設計可達圖是歐氏有向圖，則測試序列可直接透過弗勒希演算法來產生。假如設計可達圖不是歐氏有向圖，則建構運輸模式來增加最少的額外邊並形成新的歐氏有向圖，再透過 Fleury's 演算法來產生測試序列。可達圖中的生產循環也可藉由符號矩陣的對角線元素偵測之。又本文以爪哇語言實作出此規範測試系統來驗證這些構想的可行性。

關鍵詞：
自動化製造系統(Automated Manufacturing System)
斐氏圖(Petri Net)
規範測試(Conformance Testing)
運輸模式(Transportation Model)
修正 Fleury's 演算法(Modified Fleury's Algorithm)

Design and Implentation of Automated Conformance Testing System in Petri-Net-Based Software Coding Problem

Student：Chieh-Yu Lin　　Advisor：Dr. Gau-Rong Liang

Department of Industrial Engineering and Management
National Chiao-Tung University

Abstrat

For a given automated manufacturing system, the development of its control software usually involves a design phase and an implementation phase. In the design phase, the causal relation between two sequential manufacturing states can be modeled by a Petri net, and a directed network consisting of all the reachable states is named its designed reachability graph. Similarly, in the implementation phase, the corresponding Petri net is coded through a proper programming language, and its implemented reachability graph is generated naturally. Also it is necessary to test the equivalent conformance of both designed and implemented reachability graphs.

In this thesis, the designed reachability graph is used to generate a testing sequence for testing the conformance of the implemented reachability graph. If the designed reachability graph is an Eulerian digraph, then its testing sequence can be directly generated by Fleury's algorithm. If not, then a transportation model is constructed to add minimal extra edges for forming a new Eulerian digraph to which the Fleury's algorithm can be applied. Also the cycles on the reachability graph can be detected from the diagonal elements of its powered symbol matrix. In addition, a Java-based conformance testing system has been implemented for showing the feasibility of this approach.

Keywords:
Automated Manufacturing System, Petri Net, Conformance Testing, Transportation Model, Modif ied Fleury's Algorithm.

誌謝

本研究得以完成，首先要感謝在這兩年的研究生涯裡，對我細心指導及叮嚀的指導教授梁高榮博士，帶領我進入自動化製造的領域，並建置花卉戰情室，讓我獲得寶貴的求學態度及經驗。此外要感謝唐麗英老師與張永佳老師對本研究所提供之寶貴建議

其次要感謝我的父母，對我的悉心教養，對我人生方向的指引，使我明辨是非。以及兩位妹妹的陪伴，即使是分隔兩地，也能無時無刻的感受到家庭的溫暖，讓我在研究的過程中更加的堅定，更具備信心。

兩年的實驗室生活，幸好有一群志同道合的夥伴，一同研究學問的熱忱，一同玩樂的歡笑，一同比賽的汗水，這些都為我的研究生涯添加了許多色彩，你們也都是人生旅程上不可或缺的朋友。謝謝學長們大毛、阿端、阿牛、老王、BE 學長的經驗傳授，謝謝同儕昇晏、音帆、佳儒、馨儀的互相砥礪，謝謝學弟妹植宇、耿全，柏勳、瑋廷的支持與鼓勵。謝謝好友宏文的貼心陪伴。使得實驗室生活格外地溫馨與融洽。

最後感謝在交大研究生活裡所有給我指導的師長們，讓我得以成長，得以茁壯。回首過往，展望未來，我會永遠記得在交通大學的日子。

目錄

圖目錄

表目錄

第一章 緒論

本章內容分為六部份，分別為 1.1 節「研究動機」，1.2 節「問題界定」，1.3 節「研究目的」，1.4 節「研究方法」，1.5 節「論文架構」，及 1.6 節「論文進度」。

1.1 研究動機

近年來，政府積極以獎勵措施推動工廠自動化，自動化製造系統的設計包含加工、裝配與搬運三大方面互相協調合作行為。然而，雖然完成生產線自動化的規劃建置，卻無法透過遠端控制來防止或解決自動化生產線發生當機等異常情形。在投入無數的金錢與人力規劃自動化設置後，換來無法有效降低生產人力，對於需要二十四小時運作的生產工廠而言，需要三班制人員輪班看管機台，當遇到設備異常時，則須派遣人力修繕。即便完成自動化生產線，現場仍需人力監控機台，無法真正透徹自動化的涵義，達到管理者只需透過網路與監控程式即可完全自動化控制現場運作。因此對於自動化工廠而言，全面性的整體規劃十分重要。

爲了達到全面性整體規劃，自動化製造系統的設計與實作必須歷經五大步驟[3]：規格(Specification)、驗證(Validation)、撰碼(Coding)、測試(Testing)、偵查(Monitoring)。利用 IDEF0 軟體定義系統規格，斐氏圖(Petri Net)建立系統動態模型及驗證，爪哇(Java)程式語言撰寫程式，中國郵差理論(Chinese Postman Algorithm)作為規範測試，最後以擬陣理論(Matroid Theory)作系統監控法則。

自動化製造系統涉及多機台的互動行為，發展出多種不同的模擬工具來描述系統運作狀況，其中以斐氏圖模式因具有圖形化介面及數學分析，最常用來描述系統的互動行為。藉由分析斐氏圖性質驗證，了解系統的運作狀況。進而將斐氏圖模式撰碼為監控程式，以監控程式透過網路控制現場的機台運作。爲了確保監控程式的編譯符合機台的運作狀況，則需進行撰碼的規範測試。規範測試則以斐氏圖推導出狀態可達圖(Reachability Graph)檢測系統中每個狀態轉換，確認監控程式設計與現場實際運作情形的一致性，並檢驗實際運作狀態轉換間順暢，避免製造系統產生鎖死(Deadlock)現象。隨後透過擬陣理論找出製造系統中的監控法則，以斐氏圖模式觀念說明，斐氏圖模型中某些暫存點的浮標數總和存在著不變量性質，因而運用此不變量性質作為監控法則，當製造系統內這些狀態的浮標數總和不為定量時，則可馬上由監控程式偵察出系統的異常現象，立即停止系統作業，並且進行補救與修復的動作。

可達圖分析更可觀察到製造系統中具有多個等效的生產循環(Cycle)，整合機台的運作時間及搬運或等待時間。即可比較各個生產循環的績效表現，即可找出最佳的生產循環，設定為此製造系統的派工法則，將能在最少的時間完成產品製造。

針對小型的可達圖而言，可由直觀的窮舉法來測試每個節點至每個節點的路徑，然而當製造系統稍微複雜時，其可達圖會產生上百個上千個節點(Nodes)與弧(Arcs)，更無法透過人力一一測試各個狀態轉換。長久以來，如何產生複雜系統的可達圖一直困擾相關的研究領域，在此狀況下，規範測試的執行更顯見困難重重。近年來，由於斐氏圖加註語言的規範，許多斐氏圖套裝軟體被開發出來，描述複雜製造系統的狀態可達圖已非難事，複雜的製造系統可透過斐氏圖套裝軟體快速且自動化撰碼成監控程式與產生狀態可達圖，若能善加運用此自動化產生的可達圖，並提供使用者能測試完所有狀態的測試步驟，進行監控程式的規範測試，應能大幅縮減規範測試所需耗費的時間與人力。然而有關可達圖分析的相關研究，至今依然缺乏相關的系統整合。若能結合斐氏圖套裝軟體自動產生狀態可達圖的特點，進而自動化產生所有狀態轉換的最短規範測試路徑，即可快速透過測試路徑測試監控程式的準確性。

1.2 問題界定

圖 1.1 製造系統以斐氏圖模式來描述系統動態運作模型及驗證情形，透過網路監控現場生產線運作情形。經由斐氏圖加註語言的發展，已發展出套裝軟體由斐氏圖加註語言自動化產生爪哇程式語言的監控程式，以及轉出可程式邏輯控制器(Programmable Logic Controller, PLC)的控制程式，用以做為網路監控工具。監控程式需要驗證撰碼的正確性，由斐氏圖的狀態可達圖可進行撰碼的規範測試，也可由狀態可達圖分析系統的生產循環。

可達圖進行歐氏迴徑計算後推導出歐依勒路徑(Euler Path)，便可利用最短路徑測試系統所有的狀態轉換，同時檢測網路監控工具的轉換狀態與實際系統執行狀態是否一致。自動化製造系統完成後，首要透過規範測試後才能允許其互相連線操作，所謂全面測試法是指測試所有可達圖邊界時皆由起點出發，當製造系統越趨複雜時，狀態間轉換暴增，全面測試法相當耗費時間與人力。因此希能透過歐氏迴徑計算，找出最小測試路徑。規範測試基本上使用的是網路架構的可達圖，測試過程以最短路徑行經所有系統狀態轉換，確保每條狀態轉換路徑皆能至少經過一次，因此可套用中國郵差法則導出歐依勒路徑[19]。

生產循環分析則是運用圖論(Graph Theory)中的符號矩陣找出可達圖的所有循環，每個循環均為一個派工法則，藉由分析生產循環找到等效的派工法則，並由各循環中獲得時間函數，便可得到最佳的派工法則。

過去分析可達圖的困難點，在於對複雜的製造系統，無法產生系統狀態可達圖。然而，這問題已被新開發的斐氏圖套裝軟體迎刃而解。而以往系統規範測試中，必須針對各專案撰寫測試程式。如今，可由斐氏圖套裝軟體產生一致化的可達圖檔案，本研究期能提供通用程式，令所有製造系統能藉由此通用程式，完成可達圖分析自動化。

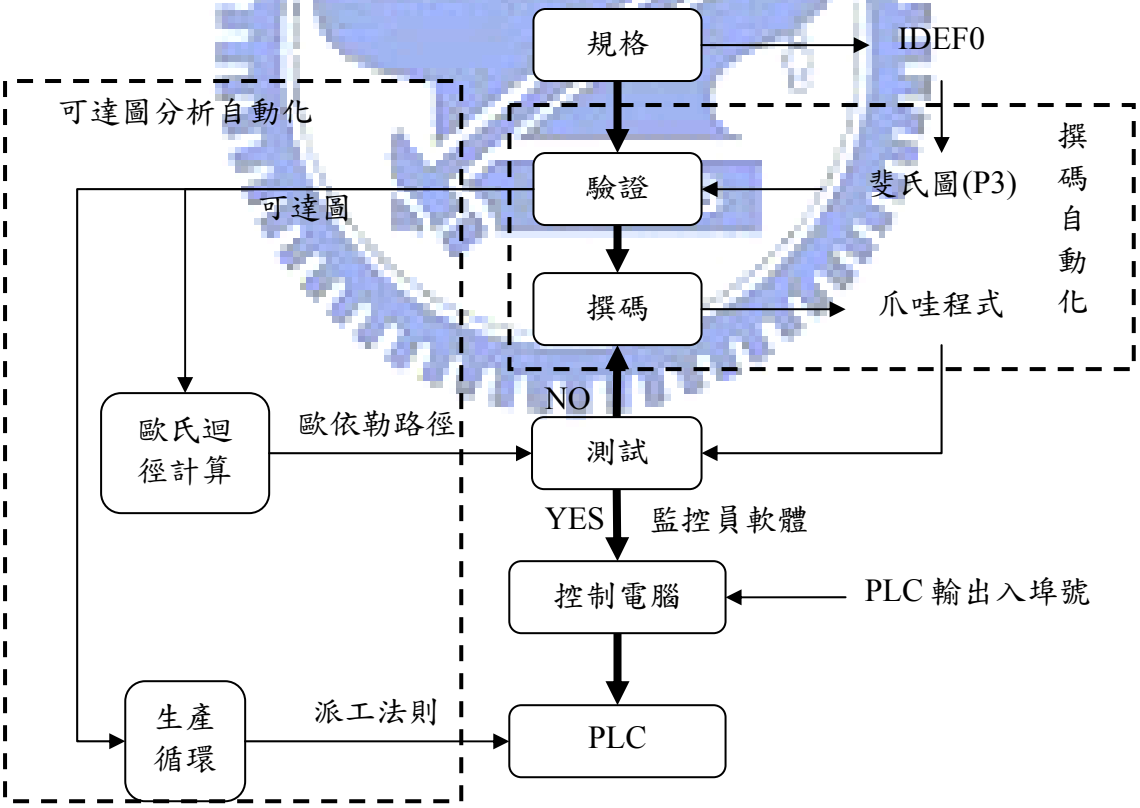


圖 1.1　自動化製造系統的設計與實作

1.3 研究目的

本研究目的在於提供迅速且通用於製造系統可達圖分析。包含兩大主題，其一運用運輸模式解法的中國郵差法則撰寫出自動化規範測試路徑通用程式，以達成規範測試自動化；其二找出可達圖中所有生產循環，可令管理者了解執行那種生產循環能擁有最佳的績效表現，建立系統的派工法則。

本研究運用爪哇程式語言讀取可達圖檔案，進而分析並產生測試路徑及生產循環的自動化程式。希望能透過此通用程式，快速縮減規範測試路徑時間，以及生產循環分析，並能套用於所有系統的可達圖分析，使用者不需依專案改變重新計算其規範測試路徑及分析生產循環。使得自動化製造系統由設計至實作需手動歷經的五大步驟，在使用者完成斐氏圖建立系統驗證後，即可透過套裝軟體完成驗證、撰碼及測試三大步驟自動化。

凡是具有流程觀念的資料，皆可用斐氏圖模式來呈現。凡是用斐氏圖模式呈現的資料，即可套用本研究所提出的規範測試軟體進行其規範測試，包含：有限狀態機(Finite-State Machine)、平行活動(Parallel Programs)、資料流計算(Dataflow Computation)、通訊協定(Communication Protocols)、同步控制(Synchronization Control)、有優先權的生產消費系統(Produces-Consumers System with Priority)、正規語言(Formal Languages)、多處理器系統(Multiprocessor Systems)、效能評估(Performance Evaluation)、彈性製造的控制系統(Flexible Manufacturing/Industrial Control Systems)、錯誤容忍系統(Fault-Tolerant Systems)、程式邏輯與積體電路 (Programmable Logic and VLSI Arrays)、分散事件系統 (Discrete Event Systems)、決策模型 (Decision Models)、辦公室的資訊流 (Office-Information Systems)等[28]。

1.4 研究方法

本研究的研究方法架構分為以下五個步驟分析探討。

步驟一：確認研究目的。

了解目前針對自動化製造系統的規範測試與生產循環領域，市面上並無提供相關的套裝軟體與系統整合。

步驟二：基本理論應用與知識研究。

確認研究方向後，針對所需要運用的知識與技術進行文獻回顧與實作，包含斐氏圖模式、可展加註語言、生產循環、運輸模式、中國郵差法則與爪哇程式語言的學習與應用。

步驟三：系統程式架構分析。

以爪哇程式語言撰碼通用程式，為了能與現行流通的套裝軟體整合，讀取現有斐氏圖套裝軟體的可達圖輸出檔，經由文獻回顧了解程式的演算法概念，進行程式撰寫，完成規範測試通用程式。

步驟四：使用者介面設定。

為了令規範測試通用程式能夠套用在不同的專案案例，程式撰寫需要設計使用者操作介面，使得不同的案例，都能透過此系統完成規範測試的步驟。

步驟五：系統實作驗證。

為了證明本研究所設定的系統可以達成預定的研究目的，套用兩專案進行系統實作驗證。

1.5 論文架構

本研究的內容編排如下，圖 1.2 為本論文之架構圖。

第一章：緒論。說明本論文研究動機、問題界定、研究目的及研究方法。

第二章：文獻回顧。首先了解本研究基礎模式的斐氏圖性質、可達圖性質與生產循環分析的文獻回顧，其次了解規範測試程式架構設計與需讀取的可達圖檔案架構與基本理論，包含斐氏圖加註語言來源與檔案規格、中國郵差問題的運輸模式解法與 Fleury 演算法。

第三章：規範測試通用程式架構設計。說明如何由第二章文獻回顧中，藉由爪哇程式語言讀取可達圖檔案，並以運輸模式解法與 Fleury 演算法為概念基礎，修正 Fleury 演算法藉以縮短程式執行速度，並以爪哇程式語言撰寫，產生規範測試路徑。

第四章：生產循環通用程式架構設計。說明如何由爪哇程式語言將可達圖檔案分析出生產循環，並且完成使用者互動通用程式系統介面。

第五章：系統實作與驗證。套用兩專案自動化導引車輛(Automated Guided Vechicle, AGV)與彈性製造系統範例執行此通用程式，包含第三章規範測試路徑及第四章生產循環分析。

第六章：結論與未來研究方向。

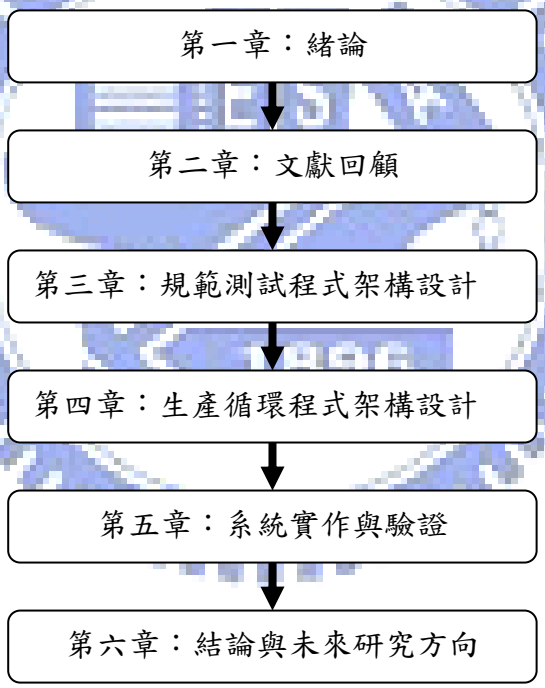


圖 1.2　論文架構圖

第二章 文獻回顧

本章主要在回顧本研究基礎架構斐氏圖模式、可達圖性質、生產循環分析(Cycle Analysis)。以及程式設計中所需運用到的規格、理論及演算法包含斐氏圖加註語言[14]、中國郵差問題，Fleury 演算法[19]及 de Bruijn 序列[19, 22]則等相關文獻，內容共分為四小節。第 2.1 節由斐氏圖模式了解彈性製造系統中不變量狀態方程式表達，經由撰碼工作將斐氏圖轉為系統的監控程式，監控程式需經規範測試後用以確保撰碼無誤，因此需透斐氏圖產生的狀態可達圖求得其規範測試路線，可達圖會受系統複雜度影響而過於龐大，故可由簡化法則簡化斐氏圖模式。第 2.2 節斐氏圖加註語言表達法，程式執行時讀取的可達圖檔案是經由斐氏圖加註語言轉換而成，故在此節說明加註語言標準格式的產生與演化。第 2.3 節說明規範測試的緣由以及了解程式設計時，如何由可達圖檔案求得規範測試路徑的演算法。第 2.4 節針對生產循環分析做文獻回顧。

2.1 斐氏圖模式

斐氏圖(Petri Net, Place/Transition Net or P/T Net)最先由德國 Carl Adam Petri 在 1962 年所發表[14]。斐氏圖是一種具有方向性的圖形化數學表達工具，用來形容動態系統之離散事件。只要具有流程觀念的資料，皆可以藉由斐氏圖來呈現。

斐氏圖包含暫存點 P(Place Nodes)、轉移點 T(Transition Nodes)、具方向性的弧 F(Directed Arc)、弧的權重 W(Weight)、初始狀態(Mo)。斐氏圖定義如表 2.1 下所示[28]：

表 2.1 斐氏圖定義

斐氏圖五元素，PN = (P, T, F, W, Mo)
P = {p1,p2,…, pm}，為暫存點的有限集合
T = {t1, t2, …, tm}，為轉移點的有限集合
$F \subseteq (P \times T) \cup (T \times P)$ ，為弧的組合(流程關聯)
$W: F \rightarrow \{1, 2, 3,...\}$，為弧的權重函數
$M_o: P \rightarrow \{0, 1, 2, 3,...\}$，暫存點的初始狀態
沒有特定初始狀態 Mo 的斐氏圖架構表示為 N=(P , T , F , W)
擁有初始狀態的斐氏圖表示成(N, Mo)

斐氏圖圖形介紹，由以下四個圖形化元素組成：

暫存點(○)(Place)：表示系統狀態。

轉移點(▬)(Transition)：代表系統的事件觸發。

方向弧(⟶)(Arc)：系統狀態改變的方向性。

浮標(◉)(Token)：代表系統正處於何種狀態。

斐氏圖的基本理論架構，藉由事件和狀態的觀念來描述系統模型，可將離散系統狀態的變化視為一個事件的發生。其中，暫存點表示狀態，轉移點表示事件，一個轉移點可以包含許多的輸入暫存點與輸出暫存點代表事件發生前的狀態與事件發生後的狀態。方向弧用來連結暫存點與轉移點，方向弧若由暫存點指向轉移點，稱此暫存點為輸入暫存點(Input Place)；反之，方向弧若由轉移點指向暫存點，則稱此暫存點為輸出暫存點(Output Place)。暫存點內可以同時擁有很多個浮標(Token)，一個暫存點內若存有 K 個浮標，代表有 K 筆項目或資源可供利用。斐氏圖中浮標在暫存點的分布狀況稱之狀態(Marking)，存有浮標的

暫存點代表系統正處於此暫存點的狀態;反之，沒有浮標的暫存點代表系統正並未處於此暫存點的狀態。

動態系統模擬方面，斐氏圖藉由轉移點的激發(Firing)，改變各個暫存點內浮標的分布狀態。下列為斐氏圖狀態轉移激發法則(Firing Rule)：

當一轉移點的所有輸入暫存點都只少存有一個浮標，則稱此轉移點為可被激發(Enable)。可被激發的轉移點(Enable Transition)可以選擇要亦或不要激發，端看系統事件是否發生。可被激發的轉移點激發後，此轉移點的輸入暫存點內的浮標數減少，而輸出暫存點內的浮標數增加。

圖 2.1 以水的化學反應方程式 $2H_2 + O_2 \rightarrow 2H_2O$ 來解釋轉移點激發法則[28]。圖 2.1(a)顯示各有兩個浮標位於輸入暫存點代表有兩單位的氫和兩單位的氧可供使用，而轉移點 t 可被激發。轉移點 t 經過激發後，系統狀態改變如圖 2.1 (b)所示，代表氫的輸入暫存點減少兩個浮標，代表氧的輸入暫存點減少一個浮標，輸入暫存點內沒有足夠的浮標數，因此轉移點 t 已經不能再被激發，而輸出暫存點產生兩個浮標，表示轉移點 t 的激發消耗兩單位的氫和一單位的氧，進而產生兩單位的水:

(a) 轉移點 t 激發前狀態，燃燒前　　(b) 轉移點 t 激發後狀態，燃燒後

圖 2.1 轉移點激發法則[28]

2.1.1 斐氏圖基本性質

斐氏圖應用於實際系統模式建構。分析斐氏圖，觀察所建構之模式是否符合實際系統的所需性質。若不符合，再對系統模式進行修正以達到系統所期望的性質。

斐氏圖基本性質又可依初始狀態(Initial Marking)而分成兩大類：行為性質(Behavioral Properties)與結構性質(Structural Properties)[28]。行為性質指斐氏圖的初始狀態有關，例如可達性(Reachability)、活性 (Liveness)等性質的探討。結構性質是指與初始狀態無關的性質，用來探討斐氏圖的結構包含安全性 (Safeness)、限制性 (Boundedness)、浮標不滅性(Conservativeness)、可逆性 (Reversibility) 與一貫性 (Consistent)等。

2.1.2 狀態方程式表達法

斐氏圖可以用來描述系統的動態行為，更可用代數方程式來表達，定義產率矩陣(Incidence Matrix)及狀態方程式(State Equation)更可將系統的動態行為描述出來。

產率矩陣定義：假設一個斐氏圖 N 具有 n 個轉移點及 m 個暫存點，則此定義產率矩陣為 $A = [a_{ij}]$ 為 $n \times m$ 的整數矩陣及 $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$，其中，$a_{ij}^{+}$ 表示方向弧權重值由轉移點 i 到輸出暫存點 j；a_{ij}^{-} 表示方向弧權重值由輸入暫存點(j)到轉移點(i)。

狀態方程式定義：$M_K = M_{K-1} + A^T u_k$, k=1,2,…，令狀態 M_K 為 $m \times 1$ 的向量，M_K 為某個激發順序中的第 k 次激發的浮標分佈狀態；uk 為 $n \times 1$ 的向量，uk 為由狀態 M_{K-1} 要到狀態 MK 第 k 次激發的控制向量，因此狀態方程式又可以初始狀態表達為 $M_d = M_0 + A^T \sum_{k=1}^{d} u_k$ 。

圖 2.2 為例[28]，斐氏圖用來描述水的化學反應方程式 $2H_2 + O_2 \rightarrow 2H_2O$，定義產率矩陣如圖 2.2 (a)所示；狀態方程式如圖 2.2 (b)所示。

$$\begin{matrix} H_2 \\ O_2 \\ H_2O \end{matrix}\begin{pmatrix} -2 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} - \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 0 \end{pmatrix} + \begin{pmatrix} -2 \\ -1 \\ 2 \end{pmatrix}^T \quad (1)$$

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \qquad M_1 = M_0 + A^T u_1$$

(a) 定義產率矩陣　　(b) 狀態方程式

圖 2.2　狀態方程式表達法

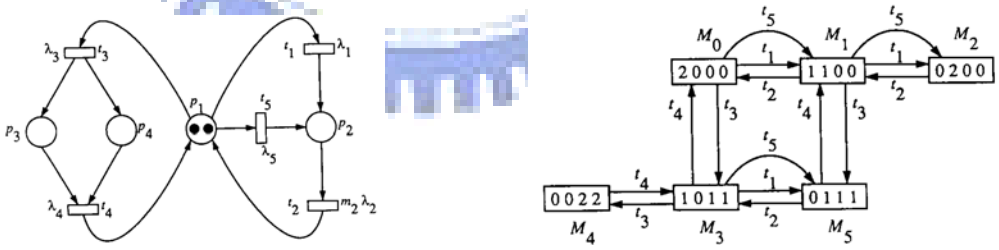
2.1.3　斐氏圖模式撰碼

自動化製造系統的設計與實作歷經五大步驟[3]。由於製造系統執行涉及多個機台與資訊的互動情形，因此利用 IDEF0 軟體定義系統規格，斐氏圖建立系統動態模型及驗證，通過驗證的斐氏圖模型則可進行監控程式撰寫，即將斐氏圖模式以程式語言轉碼為製造系統的監控程式，管理者可透過監控程式了解系統運作情形。

然而早期的撰碼工程十分費時，撰碼者需針對個別專案的斐氏圖模式進行編碼，例如監控程式的斐氏圖畫面，需經過多次編譯後，才能符合理想的畫面呈現[3]，花費太多程式設計時間於調整人機圖形介面上的斐氏圖元件。仰賴程式語言的發展，可運用爪哇豆技術[8]及斐氏圖加註語言[6]來縮短撰寫監控程式的時間。撰碼自動化能大幅縮減程式設計時間，更能通用於各個專案的斐氏圖模型。

2.1.4　可達圖表達法

狀態的可達性可用可達圖來表達。可達圖為有向圖，可將設計的斐氏圖中所有浮標可能的分佈情況表示出來；這裡點表示狀態，而弧表示兩狀態間需要激發的轉移點。可達圖的繪製首先需選定系統的初始狀態，找出所有可能從初始狀態激發的轉移點而到達的狀態，接著再找出所有可能從這些狀態到達的新的狀態。圖 2.3(a)為所設計的斐氏圖；圖 2.3(b)為圖 2.3(a)的可達圖。以圖 2.3(a)說明，當浮標分佈的初始狀態M0＝<2，0，0，0>，可選擇激發轉移點 t1、t5 和 t3。若選擇激發轉移點 t1 或 t5，則浮標分佈狀態會由M0 轉變為M1＝<1，1，0，0>;或若選擇激發轉移點 t3，則浮標分佈狀態會由M0 轉變為M3＝<1，0，1，1>。同理反覆繪製。



(a)　斐氏圖設計　　(b)　可達圖

圖 2.3　斐氏圖轉換可達圖範例[28]

可達圖的運用範圍廣泛，可用來進行系統的鎖死分析(Deadlock Analysis)[10]，生產週期分析與規範測試分析[11]。然而，可達圖繪製過程中，浮標的分佈狀態隨著暫存點與轉移點的增加成指數性爆炸的成長[34]。因此當系統越複雜時，可達圖的繪製變得窒礙難行，第 2.1.2 節所提到描述系統的產率矩陣與狀態方程式也更加龐大難以計算。因此第 2.1.5 節將介紹斐氏圖的簡化法則(Reduction Method)；利用簡化法則，縮小斐氏圖設計的複雜度，但不會影響系統特性。

2.1.5 簡化法則

簡化法則[28]的使用是爲了將複雜的系統模型變得簡單化，但仍保留系統原先的特性。如圖 2.4 列出六個簡化法保留了斐氏圖原先的特性，活性、安全性、限制性。a.合併連續的暫存點、b.合併連續的轉移點、c.合併平行的暫存點、d.合併平行的轉移點、e.消除自循環的暫存點、f.消除自循環的轉移點。

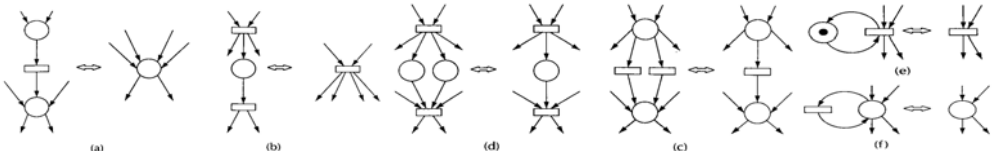


圖 2.4　簡化法則

2.2 斐氏圖加註語言表達法

在現今全球運籌的環境下，有效的控管物流資訊，能大量降低營運成本。加上商務資訊流電子化(e-commerce)的發展，不僅能讓商品加快運送的腳步，也能更精確的掌握資訊、使管理更為精確。電子商務的熱潮隨著網際網路的普及蓬勃發展，然而每家企業使用的資訊系統不一定相同，反而造成資料格式轉換成本增加。電子資料交換（Electronic Data Interchange, EDI）標準因運而生於 1960 年代，定義為：一台電腦的應用系統，運用協定的標準與資料格式，經過電子化傳遞方式，將資料傳送到另一台電腦的應用系統，讓電腦能夠自動『了解』、『處理』和『回應』，使電腦能自動處理作業進而解決資料交換的問題。台灣大概於 1990 年代開始推動 EDI 在產業間的應用，然而 EDI 雖實現了資料的一致性，卻嚴重地缺乏可延伸性。早期的 EDI 屬於專屬的封閉系統，要求所有合作企業都必須使用且唯一的解決方案，使得建置成本高，還要聘請專業的顧問，租用專屬網路，並且僅能改善和處理片段的作業流程。如果有一家廠商想增加或減少一些內容，則整體的 EDI 系統便重新進行修正。隨著電子交換的內容越來越複雜，美國 W3C 組織於 1996 年正式公佈 XML (eXtensible Markup Language)語法標準，修正 EDI 的缺點，提供商務資訊流電子化應用上更多維的延伸性[4]。

2.2.1 何謂可延伸性加註語言

1996 年，可延伸性加註語言(eXtensible Markup Language, XML)由全球資訊網發展協會(World Wide Web Consortium, W3C)所發展而成。它是一種中介標籤語言(meta-markup language)，用於標示具有結構性資訊之電子文件的標示語言，有助於文件內容的宣告並符合跨平台的搜尋[4]。

所謂標籤語言，指由一些特殊字碼(code)及標籤(tag)所組成。它單獨存在時並不代表任何意義，需透過特殊的軟體經一定規則解讀後，輸出至螢幕或是印表機等輸出設備上。標籤語言的使用，可使文件具有結構化以便於管理、解讀及運用。它又可分為「特定標籤語言」及「一般化標籤語言」兩大類。特定標籤語言是針對特定軟體或是特殊應用所設計的，具可攜性，如 HTML (HyperText Markup Language, 超文字加註語言)標籤語言是爲了特定運用在 WEB 上所設計的。一般化標籤語言只是描述文字的內容與結構，並不受限於特定的用途及軟體，如通過國際標準(International Organization for Standardization, ISO)認可的「標準通用標籤語言」(Standard Generalized Markup Language, SGML)與「通用加註語言」GML(Generalized Markup Language)，都是所謂的通用標籤語言。

XML 的開發者來自一群 SGML 的設計者與應用者，SGML 最初設計的一大目標是用來提供文件 50 年以上的壽命。SGML 功能十分強大並有相當多的機制，具高穩定性及可攜性，以便於提供各種語法來解讀、編輯與運用內容龐大且需互相連結的文件，但只有在大型企業或政府機關使用。如此複雜的系統卻無法實際地發揮出它的功用，因此，XML 擷取了 SGML 結構中的核心部份，並彌補 HTML 的不足，用以擴充在網路上應用。HTML 也是由 SGML 發展而來。HTML 是 1989 年由 Tim Berners-Lee 在 CERN 的時候所創造。HTML 著重在格局外觀的呈現；HTML 語法簡單易學，使得 HTML 在 W3(World Wide Web, W3)上大受歡迎。然而 HTML 存在些缺點包含標籤固定，延伸性較差且缺乏對內容文件資料涵意的表達。

XML 為整合 SGML 及 HTML 的優點所推演出來的標籤語言，著重在以 SGML 文件資料結構化的特色為基礎，精簡化 SGML 的複雜度。XML 對於應用程式來說，具有自我定義的特性，不需預先設定特殊的格式或架構，並能廣泛地相容於各種應用軟體中，讓 XML 標籤語言具有完整性、可攜性，相容性。因而 XML 推出的主要目的在於能直接應用於網際網路並在全球資訊網的環境中流通傳輸，因為 XML 的標籤元素與通訊協定互相獨立，

XML 可以專門應用於電子商務資料交換，藉此推動企業與企業電子商務的里程碑，圖 2.5 所示。

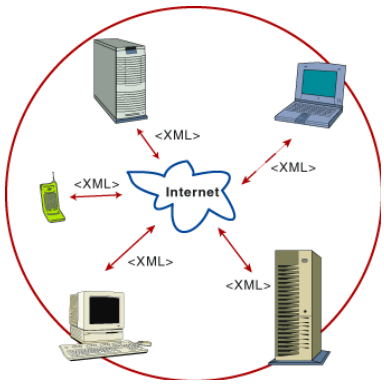
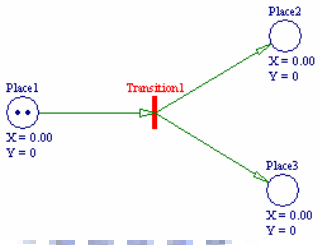


圖 2.5　XML 與網路互動運用情形[4]

2.2.2　斐氏圖加註語言

斐氏圖加註語言(Petri Net Markup Language, PNML)[14]是一種基於 XML 的特性，用來記錄多種斐氏圖架構的交換格式。利用斐氏圖類型定義(Petri Net Type Definition, PNTD)[14]來定義不同類型的斐氏圖特定屬性。

圖 2.6 為斐氏圖的範例，圖 2.6(b)是將圖 2.6(a)的斐氏圖轉換為斐氏圖加註語言。斐氏圖類型定義可用來定義斐氏圖屬性，對應的標籤用來說明暫存點、轉移點、初始狀態與方向弧分別為<place>、<transition>、<initialMarking>、<arc>，而每個標籤內又針對斐氏圖的每個種類做更詳細的屬性描述，這些屬性都被定義在一個 PNTD 內。由此運用斐氏圖加註語言標準化斐氏圖格式，便可進一步將斐氏圖加註語言套用或開發新的斐氏圖套裝軟體，進行更深入的斐氏圖分析[12]。



(a)　斐氏圖

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
<net id="n1" type="PTNet">
<place id="p1">
        <marking>
        <graphics>
        <offset page="1" x="0" y="0"/>
        </graphics>
        <value>2</value>
        </marking>
        <name>
        <graphics>
        <offset page="1" x="75" y="0"/>
        </graphics>
        <value>Place1</value>
        </name>
        <initialMarking>
        <graphics>
        <offset page="1" x="0" y="-10"/>
        </graphics>
        <value>2</value>
        </initialMarking>
        <graphics>
        <position page="1" x="105" y="163"/>
        </graphics>
        </place>
```

```
<transition id="t1">
        <name>
        <graphics>
        <offset page="1" x="45" y="-3"/>
        </graphics>
        <value>Transition1</value>
        </name>
        <graphics>
        <position page="1" x="209" y="163"/>
        </graphics>
        </transition>
<arc id="a0" source="p1" target="t1">
        <inscription>
        <graphics>
        <offset page="1" x="0" y="7"/>
        </graphics>
        <value>1</value>
        </inscription>
        <graphics>
        <position page="1" x="157" y="163"/>
        </graphics>
        </arc>
```

(b)　斐氏圖加註語言

圖 2.6　斐氏圖加註語言表達法

2.3 規範測試

可達圖為有向圖，用以描述製造系統內所有可達的狀態，且狀態間的轉換都是可控制的。因此，可藉由旅途法(Transition Tours)得到可達圖的測試序列。若可達圖具備有歐氏有向圖特性，則可藉由 Fleury's 演算法，或是將節點及弧轉換為 2 進位世界的 de Bruijn 序列，求得此可達圖的一筆畫路徑，研究採用修正 Fleury's 演算法求得可達圖的最短測試路徑。然而，無法確保製造系統的可達圖皆具有歐氏有向圖特性，因此藉由中國郵差問題，藉由增加額外的重複弧線讓可達圖變為歐氏有向圖。基本上，如何增加最少額外弧線以找到最小的歐氏有向圖也是有向性中國郵差問題。此問題可用線性規劃(Linear Programming)技術建模，更可簡化為運輸模式以降低計算複雜度。在本節中，將於第 2.3.1 節說明歐氏圖特性；第 2.3.1.1 節介紹 Fleury's 演算法；第 2.3.1.2 節介紹 de Bruijn 序列演算法；第 2.3.2 節說明中國郵差問題定義；第 2.3.2.1 節說明線性規劃解法；第 2.3.2.2 節說明運輸模式解法。

2.3.1 規範測試觀念

規範測試的概念源自於 1976 年 CCITT(International Telegraph and Telephone Consultative Committee)資料通信協定 X 系列第 25 號建議案時，發展 X.25 計畫時的痛苦經驗[26]。在此計畫中引發了三大問題：機器互相連接、難以生產標準品、互相操作。因此，發展出 ISO IS 9646 來做為通訊系統的規範測試標準，規範測試協定是用來檢測系統是否符合標準實施的一種方法，確保系統在不同的製造商之間保證成功的相互聯繫和互操作性的一項重要的技術，而通過此規範測試的設備才可允許其互相連線操作[26]。規範測試的觀念運用在許多領域，如軟體開發、通訊工業、封裝測試、自動化製造系統、網頁設計等，以確保每步驟都能正確執行。

表 2.2　ISO IS 9646

Part	Title
1	General concepts
2	Abstract test suite specification
3	Tree and tabular combined notation
4	Test realization
5	Requirements on test laboratories and clients for the conformance assessment process
6	Protocol profile test specification
7	Implementation conformance statements

測試序列可由下列四種方法產生[18, 23]：

1. 旅途法：只能使用目前已知可觀測的狀態來產生序列，測試序列可由中國郵差法則產生，特色簡單、可求得最佳解。

2. 區別型序列法 (Distinguishing Sequence)： 可以得知系統目前的狀態為何，但實際運用時，區別型序列法不一定會存在，且區別型序列長度一般非常的長，不易求解。

3. 特徵型序列法 (Characterizing Sequence)：針對無法找到區別性序列法的系統，所使用的方法。

4. 特定輸出入序列法 (Unique Input/Output(UIO) Sequence)：對於有限狀態機的每個系統狀態 Si，要尋找其最小成本輸入順序(UIO_i)，使得在輸入 UIO_i 後所產生的輸出結果對 Si 而言是惟一的。但 UIO_i 順序只能確認目前的狀態是否為 Si，不能得知目前狀態為何。

2.3.2 歐依勒旅途

從前普魯士(Prussia)的康尼斯堡(Konigsberg)(現今為俄國的加里寧格德市(Kaliningrad))。普雷格爾(Pregel)河水分隔成四個部份；如圖 2.7 所示，並透過七座橋將四個部份連接起來。當地的民眾希望能設計一趟旅行，可以行經七座橋，但每座橋都只能走過一次[19]。

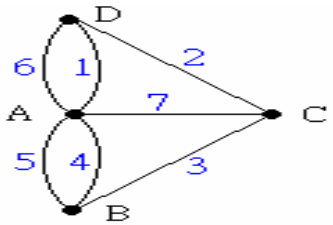


圖 2.7　七橋問題簡化圖

1736 年，歐依勒(Leonhard Paul Euler)訪問此地發現這有趣的問題，並將七橋問題以數學語言來描述，如圖 2.7，點代表被水分隔的四塊陸地，弧代表連接各陸地的橋。推論出七橋問題是無法實現的：是因為進入由一座橋進入一塊陸地時，也須由另一座橋離開此點。因此，每一個陸地與其他陸地連接的橋數必為偶數，換句話說，每一點與其他點連結的弧必為偶數，起點與終點除外。由七橋問題所繪出的圖 2.7 可看出，每個點所含的弧皆為奇數，故無法設計一趟旅行遊經七座橋且不重複。

圖中可以從一點出發行經所有邊後，到達另一點的路徑稱之為歐依勒路徑，若一圖中存在歐依勒路徑也稱之為「一筆畫問題」。圖中若存有歐依勒路徑，那麼除了起點和終點外，每點的邊界也都是偶數，而且圖中可能出現奇數邊界的點就是起點與終點。

2.3.2.1 Fleury's 演算法

在圖形中，能行經每個邊都恰好一次路徑稱此途徑為歐依勒途徑。與點相連的邊稱為邊界，如圖 2.8 所示，A 點的邊界為 3，B 點的邊界為 1。若一圖形中，每個點的邊界皆為偶數，則必定存在一條路徑，可以行經每一個邊不重複。圖 2.8 中，A 點與 B 點存在有奇數的邊界，若選擇由 A 點或 B 點作為起點或終點，則此圖可找到一筆畫路徑。若不以 A 點或 B 點作為起點，則此圖無法找到一條途徑走完所有的邊線。

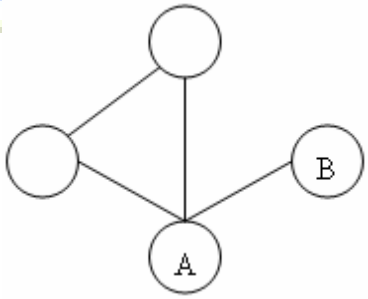


圖 2.8　連通圖範例

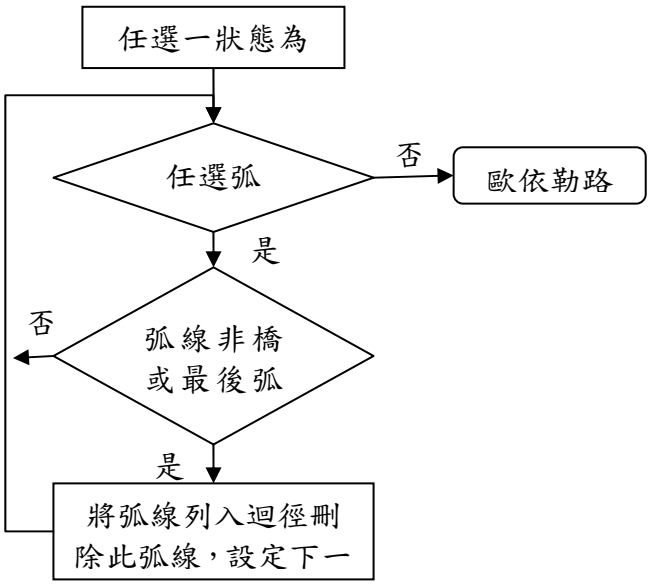


圖 2.9　無向圖之 Fleury's 演算法流程圖

依照 2.3.2 節歐依勒旅途的觀念，若圖形中每個點的邊均有偶數邊界，則可運用 Fleury's 演算法[19]找出一條歐依勒旅程。Fleury's 演算法流程如圖 2.9 所示，適用於無方向性的網路圖：可任選一狀態出發，任選一條由此狀態出發的邊界，確認此邊界是否為橋(Bridge Testing)[13]。若非橋，記錄此行經路徑並將此邊界刪除，重複選擇下一出發邊界；若此邊界為橋，則選擇此狀態的其他出發邊線，若此狀態已無其他連接邊線，則可行經橋並將此狀態刪除。

「橋」即為圖中的一個邊線，若刪除此邊線會導致圖形不連接，則稱此邊界為橋。如圖 2.10(a)所示，線段 AB 若刪除，會導致圖形分成如圖 2.10(b)所示 ACD 與 BEF 兩區域，造成圖形的不連接，因此稱此線段 AB 為橋。

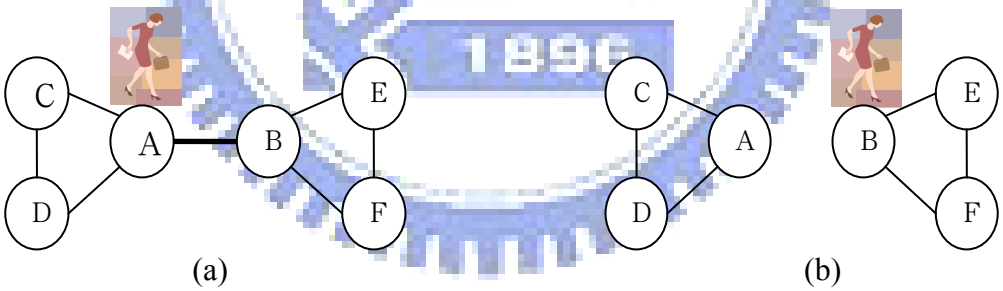


圖 2.10　橋-連通圖檢驗

2.3.2.2 de Bruijn 序列

de Bruijn 序列為一個循環數列$(a_1, a_2, \ldots, a_{2^n})$稱之為(2, n)- deBruijn 數列，滿足下列兩個條件：(1)$a_i \in \{0, 1\}$，$i = 1, 2, \ldots, 2^n$；且(2)$(a_j, a_{j+1}, \ldots, a_{j+n-1}), j = 1, 2, \ldots, 2^n, (\mathrm{mod}\, 2^n)$，為相異的 2 個維向量。例如：n = 1,2,3,5；的(2, n)-de Bruijn 數列分別為 01, 0110, 01110100, 0000100110101111。對於建構一個(2, n)-de Bruijn 數列至所有的 n 並不困難。荷蘭數學家 N. de Bruijn 用來尋找此數列的有向圖。(2, n)-de Bruijn 有向圖，$D_{2,n}$ 為一加權有向圖，它滿足下列兩條件：(1) $V(D_{2,n}) = (Z_2)^{n-1}$ 及；(2)$^{(a_1, a_2, \ldots, a_{n-1})}$ 連到 a 並且在這個弧上給予加權 $(a_1, a_2, \ldots a_n)$。如圖 2.11 所示。更由於有向圖 $D_{2,4}$ 為強連通圖而且每一點的出發的邊線與進入的邊線均為 2，可證明(2, n)-de Bruijn 有向圖為歐氏有向圖。

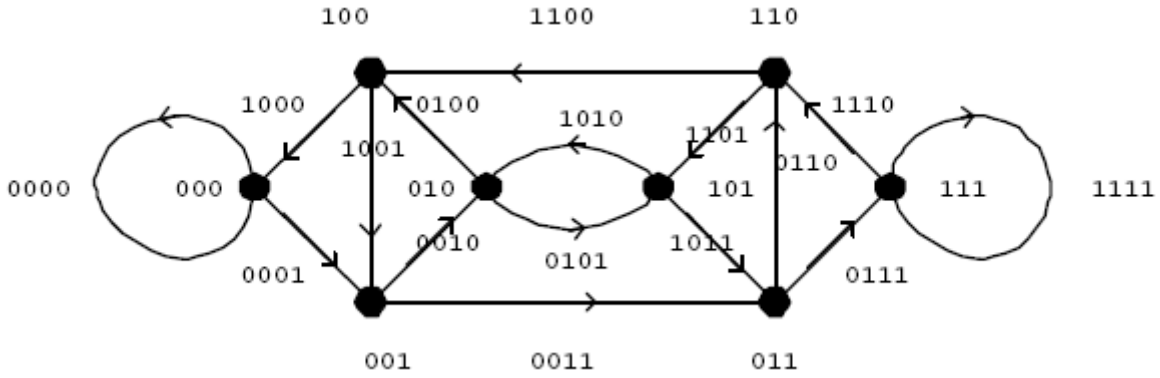


圖 2.11　de Bruijn 有向圖($D_{2,4}$)

一個(2, n)-de Bruijn 有向圖的存在一定存在一條歐氏迴徑，令此迴路經過的邊為 $e_1, e_2, \ldots, e_{2n}$；同時對於所有的 i 令 $l(e_i) = a_i$ 為 e_i 邊上加權的最左邊那個數字，於是我們得一個數列 $(a_1, a_2, \ldots a_{2^n})$，此數列就是一個(2, n)-de Bruijn 數列。對於圖 2.11 有向圖 $D_{2,4}$ 的一筆畫路徑為圖 2.12。

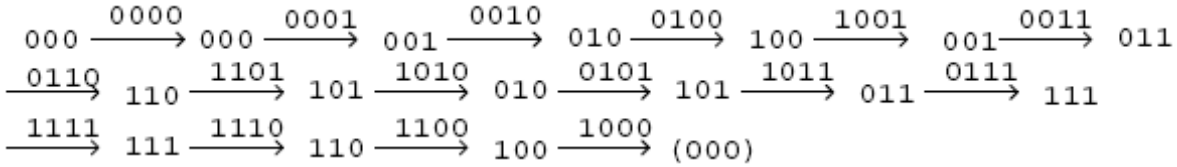


圖 2.12　de Bruijn 有向圖($D_{2,4}$)之歐氏迴徑

de Bruijn 數列最著名的應用在於旋轉鼓(Rotating Drum)。它可利用連續的位置來判斷不同的機械輸入原理。如

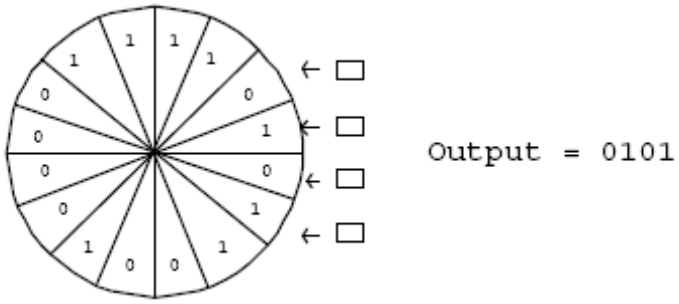


圖 2.13　旋轉鼓

2.3.3　中國郵差法則

中國郵差問題由華裔數學家管梅谷[19]於 1962 年所提出，為了尋找一條最短或是成本最少的路徑，使得每條路徑至少經過一次，最後回到原點。如同郵差必須沿著送信區域內的每一條街道送信，每條街一定且至少要經過一次，最後回到郵局也就是原點。有些街道一定要經過超過一次以上，為了節省送信時間，如何將重複走的街道長度越短越好，故如何找出此郵差最短的送信路徑被稱為“中國郵差問題”。

中國郵差定理至今的應用十分廣泛，並推演出許多的問題模式。如圖 2.14 可由街道的方向性分為無向中國郵差問題(Undirected Postman Problem)、有向中國郵差問題(Directed Postman Problem)及混向中國郵差問題(Mixed Postman Problem)三種[19]。或是考慮容量限制、路徑長短、時間限制等因素所拓展出的問題模式，例如容量限制中國郵差問題

(Capacitated Postman Problem)[31]、市郊中國郵差問題(Rural Postman Problem)[30]、風向中國郵差問題(Windy Postman Problem)[29]、最大利益中國郵差問題(Maximal-Benefit Postman Problem)[32]、時間限制中國郵差問題(Time-constraint Postman Problem)。

解決中國郵差問題的方法即將圖形歐依勒化（Eulerizing）。所謂歐依勒化就是在圖中加上一些邊，加上的邊是指需要重覆經過的路徑，而這些邊必須是這兩點間原本就存在的邊才行，代表兩地原本就有道路可以通行。由第 2.3.2 節歐依勒旅途解法可衍生至中國郵差問題。由歐依勒定理得知，每節點的邊界需均為偶數，也就是需將網路中每節點對稱化，所謂對稱化即是每節點進入的邊數等於出發於此節點的邊數。

斐氏圖所繪製出的可達圖屬於有向的連通網絡，故本研究只探討有向中國郵差問題。圖 2.15 有向中國郵差問題解法可分為線性規劃模式與運輸模式兩種，對於運輸模式初始解又有三種選定方法，將在第 2.3.3.2 節討論。兩種方法的理論基礎都是將節點對稱化，使其成為具有對稱性的網路。首先來探討線性規劃模式[19]。

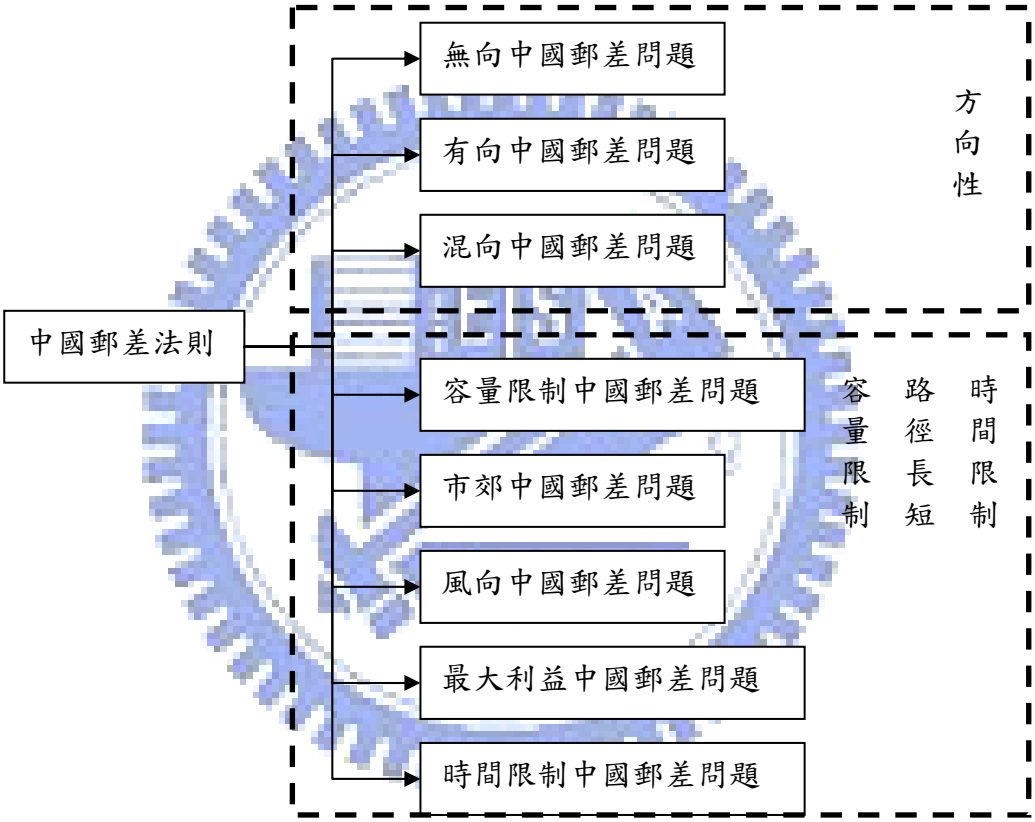


圖 2.14　中國郵差問題模式

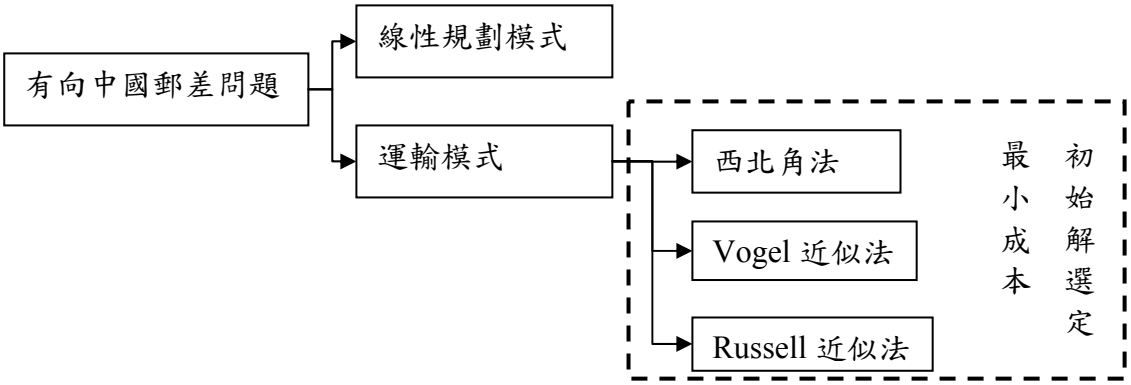


圖 2.15　有向中國郵差問題解法

2.3.3.1 線性規劃模式解法

線性規劃模式(Linear Programming Model)[19]首要關鍵是將網路中的每個節點對稱化，並使得所複製的邊界長度總合最小化，由此觀念可建立出線性規劃模式如下：

目標式 $\text{Min } z = \sum c_e x_e$

s.t $x_e \geq 0$， x_e為整數對$e \in E$

$\sum\{x_e: e\text{為從進入節點n的邊界}\}$

$-\sum\{x_e: e\text{為從節點n出發的邊界}\} = b_n, \forall n \in N$

其中 c_e 為行經邊界 e 所需要花費的成本，x_e 為邊界 e 所複製並加入網路中的邊界數，x_e 邊界數必定為大於零的正整數，b_n 為進入節點 n 的邊界減去從節點 n 出發的邊界。此模式的限制會對所有 $n \in N$ 而言，使得從每個節點出發的邊界等於進入此節點的邊界。線性規劃模式可利用 Lindo 或 Lingo[15]等數學軟體求解。

2.3.3.2 運輸模式解法

運輸模式(Transportation Model)[27]是因其應用在規劃最佳運送貨物而得名，為了以最小成本的方式，處理從任何供應中心運送商品至接收中心的問題，一個運輸問題會有可行解的特性，亦即所有源點的總供給量等於所有終點的總需求量。將運輸模式套用至本研究，首先必須將網路中節點分為供應點與需求點，供應節點即為進入此節點的邊界數大於由此節點出發的邊界數，表示此節點還可以提供一條或多條邊界數與其他節點連接，由線性規劃模式來解釋，即為 $b_n > 0$，此時 b_n 為此節點供應量；此外，需求節點即為由此節點出發的邊界數大於進入此節點的邊界數，表示此節點還需要一條或多條邊界數與其節點相連，亦為 $b_n < 0$，此時 b_n 為此節點需求量。由以上所述，還必須建立一張由需求點至供應點的成本需求表，即可應用作業研究中運輸問題演算法，求得需求點至供應點的最小路徑總和。表 2.3 為運輸問題表示法。

表 2.3 運輸問題表示法-成本需求表

供應點\需求點	需求點(n 個)	供應量($b_n > 0$)
供應點(m 個)	c_{11} …… ……c_{ij}…… ……c_{mn}	$s_i = b_n > 0$
需求量($b_n < 0$)	$d_j = \lvert b_n < 0 \rvert$	

運輸模式演算法包含三大步驟，即(1)建立運輸表(Transportation Tableau)，(2)找到一組初始解，及(3)利用凸多面體法(Simplex Method)重複改善初始解的值，直到產生最低運輸成本為止。由供應需求狀態的路徑成本表中，可藉由初始解演算法找出運輸問題初始解，然而初始解不一定為問題最佳解，因而初始解需進行最佳解檢測。若目前解不為最佳解，則進行凸多面體法；若目前解為最佳解，則停止。運輸問題選擇初始解有三種標準方法：西北角法(Northwest corner rule)、Vogel 近似法(Vogel's approximation method)[23]，Russell 近似法(Russell's approximation method)[23]。

西北角法如表 2.4 所示，首先選擇(x_{11})即從運輸單行表的西北角開始。其後，假設(x_{ij})為前一個選定的基變數，若原點還有剩餘供給量，則選擇(x_{ij+1})(亦即向右移動一欄)，否則選擇(x_{i+1j})(亦即向下移動一列)，因此表 2.4 中(x_{11})為第一基變數，供應狀態 1 無剩餘供應量，則下一基變數向下移動至(x_{21})，以此類推找出所有基變數。

Vogel 近似法對於尚在考慮的各欄各列，計算其差額。差額定義為各欄或各列需考慮的成本中，最小與次小的成本差。由差額最大的欄列中，再選出其成本最小的變數。以表 2.5 所示，選擇列差與欄差最大值 x_{23} 為基變數分配量為 1，刪除列 2。再次反覆計算列差與欄差最大值，以選擇下一個基變數。

表 2.4　西北角法

成本(c_{ij})				
供\需	4	5	6	供應量
1	2 1	3	1	1
2	2 0	4 1	4	1
3	3	4 0	3 1	1
需求量	1	1	1	

表 2.5　Vogel 近似法

成本(c_{ij})					
供\需	4	5	6	供應量	列差
1	2	3	1	1	1
2	2	4	4 1	1	2
3	3	4	3	1	1
需求量	1	1	1	選擇 $x_{23}=1$	
欄差	1	1	2	刪除列 2	

表 2.6　Russell 近似法

$\triangle_{ij}=c_{ij}-u_i-v_j$					
供\需	4	5	6	供應量	u_i
1	2 -4	3 -1	1 -6	1	3
2	2 -5	4 -4	4 -4	1	4
3	3 -4	4 -4	3 -5	1	4
需求量	1	1	1	選擇 $x_{13}=1$	
v_j	3	4	4	刪除列 1	

Russell 近似法則是對於尚在考慮的每一個源點列中，找出留在該列中最大的單位成本，令其為 u_i。對於尚在考慮的每一個源點欄中，找出留在該欄中最大的單位成本，令其為 v_i。對於尚未選定的 x_{ij}，計算其$\triangle_{ij}=c_{ij}-u_i-v_j$，選擇具有最大負值$\triangle_{ij}$的變數。以表 2.6 為例，找出各行列的最大值 u_i 及 v_i，計算其$\triangle_{ij}=c_{ij}-u_i-v_j$，選擇最大負值$\triangle_{ij}$為 x_{13} 為基變數分配量為 1，刪除列 1。再次反覆找出列與欄最大值，以選擇下一個基變數。

三種初始解的標準方法中，西北角法主要優點為快速容易，然而未考慮到路徑成本的因素，因此西北角法求得的初始解會離最佳解很遠。Vogel 近似法易於手算，因為差額代

表未能分配至最小成本的列或欄所需額外付出的最小成本，因此 Vogel 近似法以有效考慮成本因素。Russell 近似法在電腦的執行速度很快，因 Russell 近似法在 u_i 和 v_i 的定義方式與可估計 $c_{ij} - u_i - v_j$ 的值，可以簡化電腦程式，也有考慮到最小成本的因素。由以上分析 Russell 近似法與 Vogel 近似法皆能得到較好的解。

對於兩種解法複雜度而言，線性規劃模式複雜度為 $O(n^3)$[27]，n 為變數個數亦為網路中的邊界數；運輸模式複雜度為 $O(m \times n^2)$[27]，m 與 n 分別為網路中供應點和需求點個數。本研究針對斐氏圖所產生的狀態可達圖實作中國郵差問題解法，其中狀態可達圖的邊界數必定會多於節點數，以計算時間為考量，因此本研究選擇以運輸模式求解。運輸模式問題模式有三種標準方法選擇初始解，由於本研究運用爪哇程式語言撰寫，Russell 近似法與最佳解檢測有相同的定義方式，可簡化程式撰碼，故採用 Russell 近似法求得運輸的初始解。

2.4 生產循環

可達圖可利用圖論之符號矩陣(Symbol Matrix)來找出所有的循環(Cycle)[17]。圖 2.16 之可達圖可用 8 乘 8 矩陣來表達，如圖 2.17。其中的數值是表示可達圖中的某一個狀態可達另一個狀態，例如 12 表示狀態 1 可達狀態 2。計算 S 的次冪，並對照對角線元素值便可得循環數[1]，如圖 2.18。

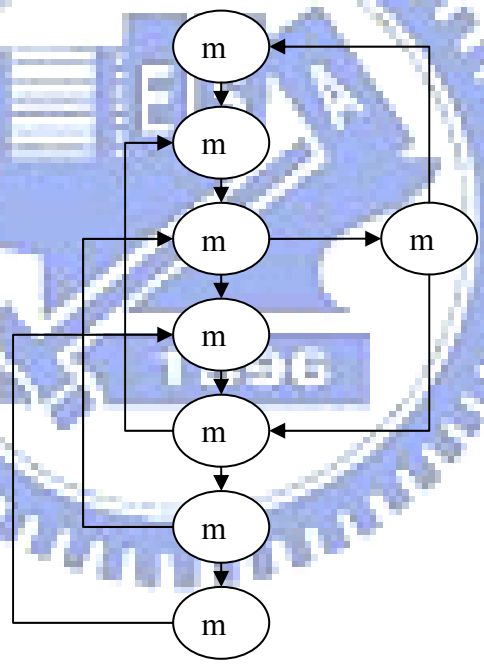


圖 2.16　可達圖

$$S = \begin{bmatrix} 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 23 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 34 & 0 & 0 & 0 & 38 \\ 0 & 0 & 0 & 0 & 45 & 0 & 0 & 0 \\ 0 & 52 & 0 & 0 & 0 & 56 & 0 & 0 \\ 0 & 0 & 63 & 0 & 0 & 0 & 67 & 0 \\ 0 & 0 & 0 & 74 & 0 & 0 & 0 & 0 \\ 81 & 0 & 0 & 0 & 85 & 0 & 0 & 0 \end{bmatrix}$$

圖 2.17　符號矩陣 S

$$S^4 = \begin{bmatrix} 12381 & 0 & 0 & 0 & 12345+12385 & 0 & 0 & 0 \\ 0 & 23452+23812+23852 & 0 & 0 & 0 & 23456+23856 & 0 & 0 \\ 0 & 0 & 38123+34523+34563+38523+38563 & 0 & 0 & 0 & 34567+38567 & 0 \\ 0 & 0 & 0 & 45234+45634+45674 & 0 & 0 & 0 & 45238+45638 \\ 52381+56381 & 0 & 0 & 0 & 52345+52385+56345+56385+56745 & 0 & 0 & 0 \\ 0 & 63452+67452+63812+63852 & 0 & 0 & 0 & 63456+67456+63856 & 0 & 0 \\ 0 & 0 & 74523+74563 & 0 & 0 & 0 & 74567 & 0 \\ 0 & 0 & 0 & 81234+85234+85634+85674 & 0 & 0 & 0 & 81238+85238+85638 \end{bmatrix}$$

圖 2.18　符號矩陣 S^4

由圖 2.18 所示，由 S^4 矩陣的對角線，可以求出六個循環：C1，C2，C3，C4，C5，C6。其狀態轉移模式如下：C1：[1][2][3][8][1]，C2：[4][5][6][3][4]，C3：[5][6][3][8][5]，C4：[2][3][8][5][2]，C5：[2][3][4][5][2]，C6：[4][5][6][7][4]。進而發現 S^5，S^6 的對角線已無循環，最後符號矩陣為 0。因此知道此可達圖無其他的循環

第三章 規範測試程式架構設計

本章將說明如何利用斐氏圖套裝軟體所產生的狀態可達圖，更進一步推導出最小成本的歐依勒一筆畫路徑。3.1 節說明自動化規範測試路徑應用流程，3.2 節說明規範測試程式基本架構圖，第 3.3 節說明如何利用可達圖資料分析鎖死狀態，3.4 節說明如何運用無鎖死狀態的可達圖，透過運輸模式的計算找出最少加邊路徑，3.5 節說明當加邊路徑確定後，運用修正 Fleury's 演算法導出歐依勒路徑。

3.1 自動化規範測試路徑應用流程

製造執行系統在開始量產製造產品前，必須先經過規範測試；特別是製造系統自動化後，製造加工步驟是透過電腦系統的決策系統自動控制現場機台，如圖 3.1 所示。規範測試可透過電腦系統自動控制現場的機台運作，同時透過網際網路監控現場實際運作狀況與電腦決策系統模擬的一致性，用以確保執行系統中所有狀態轉換皆為順暢可行，並且不會發生任何搶資源而造成的鎖死問題。通過規範測試的自動化製造執行系統才能開始實際運作。

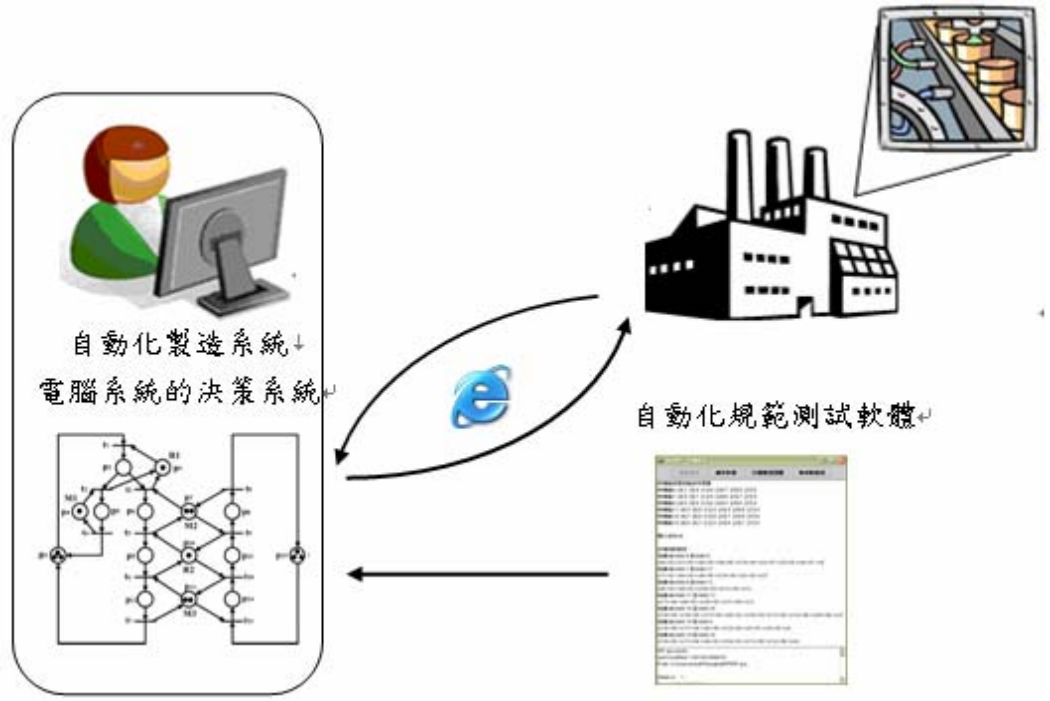


圖 3.1 自動規範測試軟體應用示意圖

本論文是採用爪哇程式語言撰寫而成，而其流程如圖 3.2 所示。自動化規範測試系統設計包含四個模組，分別為活可達圖(Live Reachability Graph)模組、歐氏有向圖模組、測試序列模組、循環分析模組。仰賴斐氏圖套裝軟體的發展，可自動化產生設計階段的狀態可達圖檔案。自動化規範測試路徑系統讀取並分析此可達圖檔案後，首先進行活可達圖模組分析，刪除可達圖內所有鎖死狀態，確保此可達圖具有活性。緊接著，判斷具有活性的可達圖是否為歐氏有向圖，若無，則進行歐氏有向圖模組分析，藉由中國郵差理論中運輸模式演算法，將可達圖藉由加邊方式轉換為歐氏有向圖。若此活的可達圖已是歐氏有向圖，則可進行測試序列模組分析，藉由廣益 Fleury's 演算法，自動產生一條最短測試序列路徑，以便於檢測所有系統狀態。此外，藉由活可達圖模組分析後具有活性的可達圖，可進行循環分析模組的分析，透過符號矩陣的對角線元素，找出具有相同製程效果的生產循

環。本論文能大幅縮短測試每個系統狀態的測試時間，並能套用在所有斐氏圖模型的可達圖檔案。本論文的規範測試路徑可透過自動化監控軟體進行狀態轉換測試，利用自動化監控軟體控制製造現場，同時可檢驗模擬軟體是否製造現場的狀態變化同步。

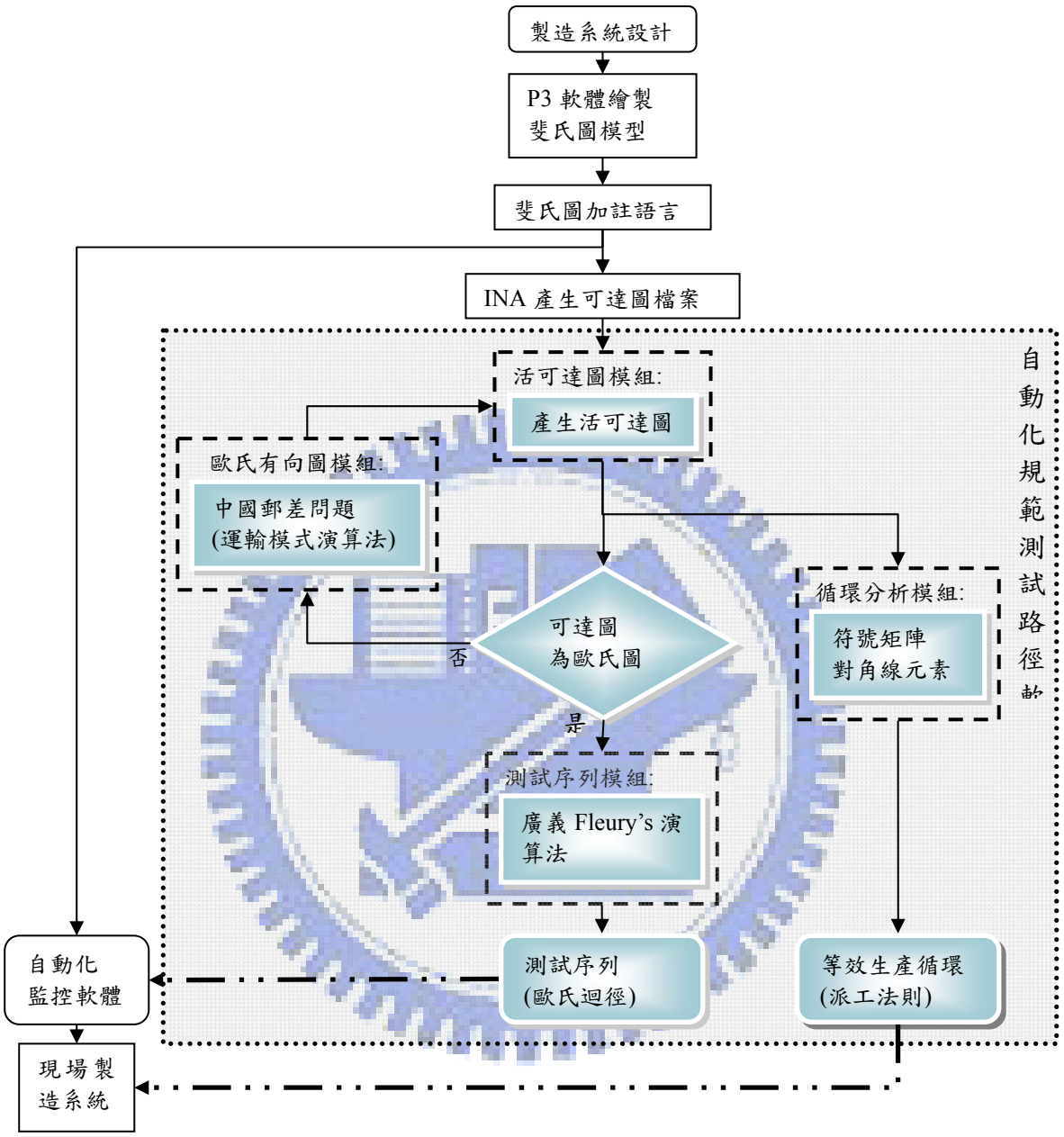


圖 3.2　規範測試路徑軟體分析應用流程圖

規範測試路徑軟體設計細節以 IDEF0 表示之如圖 3.3。IDEF0 於 1985 年由 Ross,D.T 教授所提出的結構化分析與設計技術(Structure Analysis and Design Techniques, SADT)[33]，1993 年定為美國國家標準與技術局(National Institute of Standards and Technology, NIST)第 183 號標準，具有功能性的系統架構工具，用於分析系統流程。IDEF0 表達圖中間的方框代表處理作業，左邊箭號代表輸入項目，右邊箭號代表輸出項目。圖 3.3 規範測試路徑軟體之需要輸入可達圖加註語言檔案，經過規範測試路徑處理作業後，可輸出規範測試路徑。主流程規範測試路徑處理作業其子流程為四項作業處理於第 3.2 節詳細介紹。

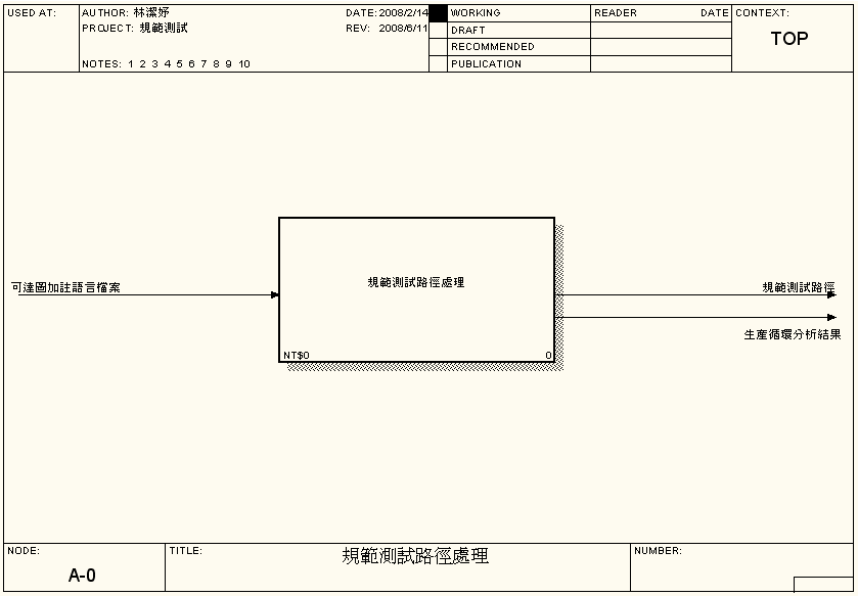


圖 3.3　規範測試路徑軟體之 IDEF0 表達圖

3.2 規範測試路徑處理作業程式架構

圖 3.4 為規範測試程式設計的 UML[16]圖，使用者可藉由操作介面觸發軟體中各個步驟的程式類別。整個使用者操作介面包含三大工具列，其一為主功能表，包含開啟檔案，清除檔案，以及系統資訊三個部分：開啟檔案按鍵，系統可讀取副檔名為.gra 的可達圖檔案，並將檔案內容顯示於螢幕上；清除檔案按鍵，系統可清除前一份可達圖專案資料，以便於進行另一專案的操作；系統資訊按鍵，為提供使用者了解此軟體功能。其次，為了區別規範測試與生產循環兩大主題，分別用兩個 Tabbed Panel 區別兩主題的操作按鈕。規範測試分析作業中，包含活可達圖模組，歐氏有向圖模組，測試序列模組三個按鍵，分別觸發不同的程式類別。生產循環分析作業中，包含循環分析模組按鍵，進行等效的生產循環分析。本論文將在第三章 3.3 節之後，針對主功能表的讀取檔案處理作業及規範測試處理作業做詳細的介紹。並於第四章，針對生產循環分析部分做詳細的介紹。

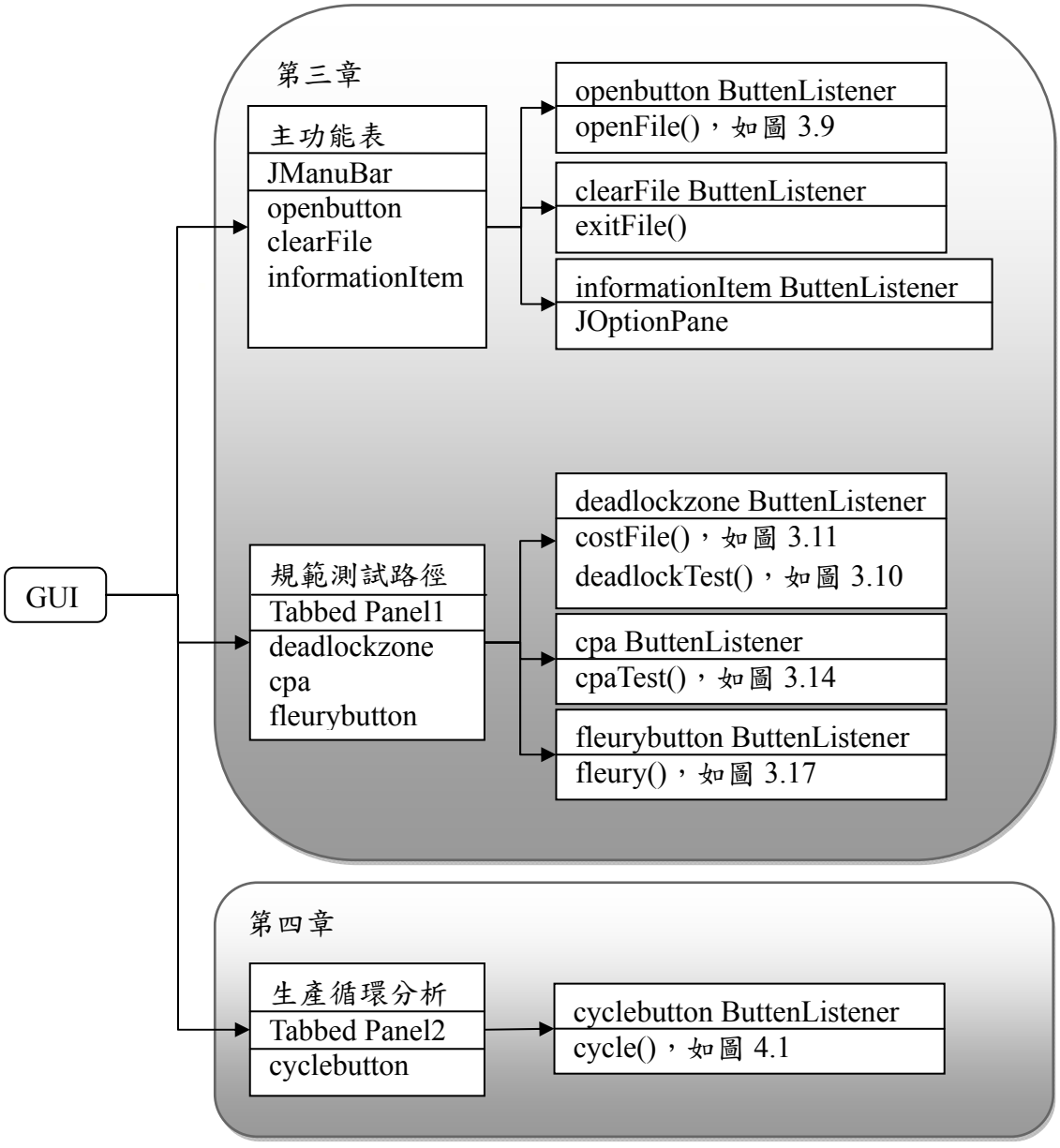


圖 3.4　規範測試程式設計 UML 圖

推導出規範測試路徑軟體的主程式流程如圖 3.5 所示，包含四個主要作業處理，分別為讀取檔案、鎖死分析、運輸模式加邊計算與歐依勒路徑計算。首先讀取欲分析的可達圖檔案，由檔案裡分析出狀態間轉移點資訊、路徑成本資訊與狀態間路徑資訊，其次進行鎖死分析。鎖死分析是爲確保所有狀態都可回到初始狀態，根據路徑成本資訊計算出鎖死狀態，找出會進入鎖死狀態的鎖死區域，刪除所有位於鎖死區域的狀態，使其斐氏圖具有活性，代表所有的狀態都能互相轉換，隨後進行運輸模式分析。運輸模式分析依據狀態路徑資訊、路徑成本資訊與鎖死區域資訊，找出需要重複行經的路徑，並最小化重複路徑，輸出狀態加邊資訊。最後歐依勒路徑計算接收狀態加邊資訊與狀態路徑資訊，透過 Fleury 演算法，算出歐依勒路徑，並整合狀態轉移點資訊輸出一條規範測試路徑。

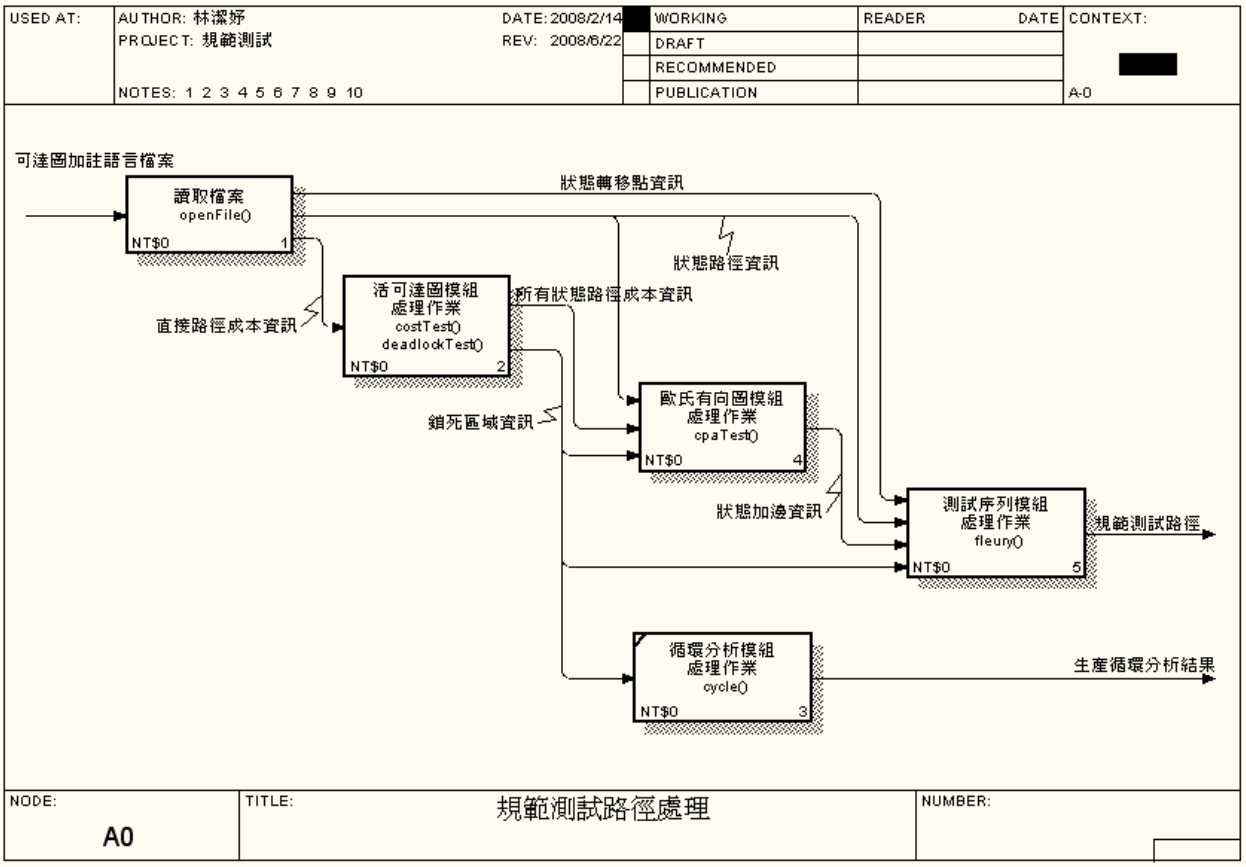


圖 3.5　規範測試路徑軟體程式流程架構

表 3.1 為程式設計中，各類別的變數設定及變數意義，將在第 3.3 節中介紹，由 openbutton 觸發的 openFile()類別程式執行；第 3.4 節鎖死分析處理作業介紹，由 deadlockzone 按鍵觸發的 costFile()及 deadlockTest()類別程式的執行方式；第 3.5 節運輸模式處理作業介紹，由 cpa 按鍵觸發的 cpaTest()類別程式的執行方式；第 3.6 節歐依勒路徑處理作業介紹，由 fleurybutton 按鍵觸發的 fleury()類別程式的執行方式。第四章將介紹生產循環分析處理作業，由 cyclebutton 按鍵觸發的 cycle()類別程式的執行方式。

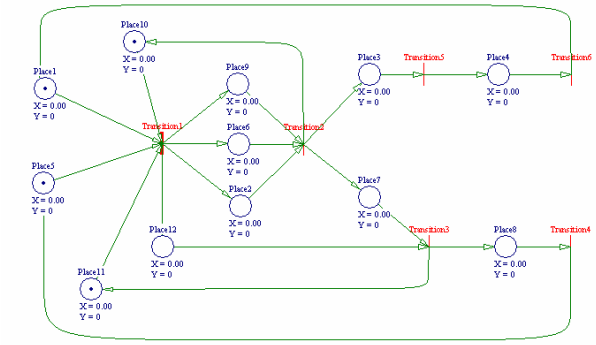
表 3.1　程式各類別變數設定及意義

類別	openFile()	讀取檔案	
變數名稱		變數類型	變數意義
-cost[][]		int	成本變數儲存
-transition[][]		int	轉移點變數儲存
-euler1[][]		int	有幾個路徑
count		int	計算有幾個狀態
類別	costFile()	計算所有路徑成本	
變數名稱		變數類型	變數意義
sum		int	紀錄是否替換 cost
-cost[][]		int	成本變數儲存
-map[][]		String	記錄需要加邊的路徑
類別	deadlockTest()	刪除連接到鎖死狀態的路徑	
變數名稱		變數類型	變數意義
-inputarc[]		int	進去此狀態次數
-outputarc[]		int	由狀態出去次數
類別	cpaTest()	歐氏有向圖模組	
變數名稱		變數類型	變數意義
supply[]		int	紀錄供應量
demend[]		int	紀錄需求量
supplypoint[]		int	紀錄供應狀態
demendpoint[]		int	紀錄需求狀態
costcal[][]		int	供需成本表
road[][]		int	供需成本表計算後節點
rowresidual[]		int	最佳化測試列剩餘值
columnresidual[]		int	最佳化測試行剩餘值
類別	fleury()	測試序列模組	
變數名稱		變數類型	變數意義
euler[][]		int	計算用的路徑
costF[][]		int	重新紀錄的成本矩陣
costG[][]		int	計算用的成本矩陣
not		int	紀錄不能走的路 下一個迴圈的初始點
state		int	紀錄總路徑數
passpath		int	錄已走過幾條路
類別	cycle()	循環分析模組	
變數名稱		變數類型	變數意義
availablestate		int	可用狀態數
cyclepoint		int	第幾次循環擁有等效生產週期現象
manufacturCycle[]		int	儲存生產循環
cyclestring		String	暫存初始狀態轉換資訊為字串
cycle[]		String	展開樹陣列
cycleshow[]		String	回到原始狀態的展開樹字串存至此
countlist[]		int	狀態轉移次數
countlist2[]		int	回到原始狀態的狀態轉移次數
cyclenumber[]		int	具有相同轉移次數的狀態數
exitcycle		boolean	檢驗有無 Cycle
startnumber		String	紀錄讀取的點
last		String	紀錄讀取的點
bag		Set	儲存狀態轉移的狀態組合
bag2		Set	儲存狀態轉移的狀態組合 2
manucycle[]		String	將有等效的生產循環存在 manucycle

3.3 讀取檔案處理作業

以往系統狀態可達圖是由人工繪製，隨著系統複雜度與日俱增，系統變化狀態也成指數型爆炸。至今，透過斐氏圖加註語言標準化斐氏圖格式，運用斐氏圖加註語言發展而成的斐氏圖套裝軟體逐漸被開發，系統狀態可達圖的繪製已非遙不可及的夢想。

本節主要描述如何由斐氏圖模型產生可達圖加註語言檔案[5, 6]，並由規範測試路徑軟體讀取檔案資料。圖 3.6(a)為利用斐氏圖繪製軟體 P3，所描述液體加熱系統斐氏圖模型[2]。圖 3.6(b)為圖 3.6(a)斐氏圖模型所轉換而成的部份斐氏圖加註語言 pnml 檔案，斐氏圖加註語言將斐氏圖以圖形的表現方式轉換為文字輸出，由檔案中的資料可讀取出斐氏圖產生的狀態總數，狀態可經由哪些觸發到下一個狀態，每個狀態中暫存點的浮標分佈。圖 3.7 為斐氏圖套裝軟體 INA，斐氏圖模型轉換為斐氏圖加註語言後，可利用 INA 進行斐氏圖特性分析，包含不變量分析、屬性分析、可達圖分析等。圖 3.8(a)為圖 3.6(a)斐氏圖模型的狀態可達圖，而圖 3.8(b)可達圖檔案則是利用圖 3.7 套裝軟體 INA 將圖 3.6(b)的斐氏圖加註語言進行分析後，以文字方式輸出的可達圖資訊。本研究則是運用圖 3.8(b)的可達圖文字資訊，作進一步的分析推導。




```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
net id="n1" type="PTNet">
<place id="p1">
        <marking>
                <graphics>
                        <offset page="1" x="0" y="0"/>
                </graphics>
        <value>1</value>
        </marking>
        <name>
                <graphics>
                        <offset page="1" x="75" y="0"/>
                </graphics>
        <value>Place1</value>
        </name>
        <initialMarking>
                <graphics>
                        <offset page="1" x="0" y="-10"/>
                </graphics>
        <value>1</value>
        </initialMarking>
        <graphics>
                <position page="1" x="50" y="185"/>
        </graphics>
</place>
```

(a) 液體加熱系統斐氏圖模型　　(b) 斐氏圖模型轉換而成斐氏圖加註語言

圖 3.6 斐氏圖模型轉換斐氏圖加註語言

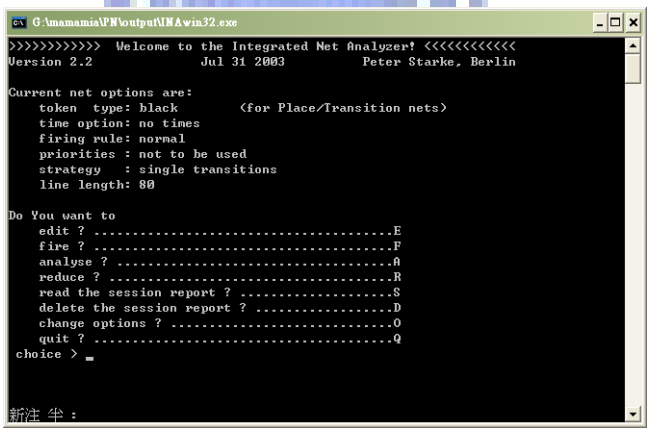
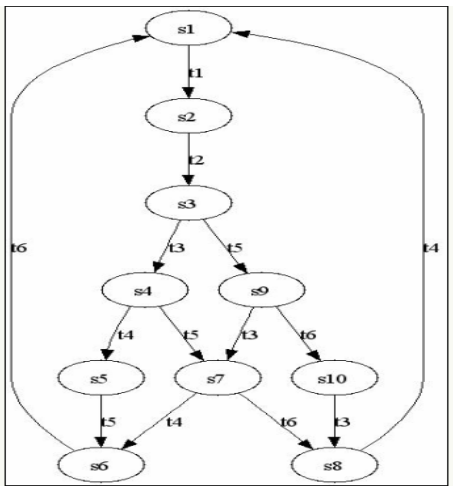


圖 3.7 斐氏圖套裝軟體 INA



(a) 可達圖

```
State nr.    1
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  1  0  0  0  1  0  0  0  0  1  1  0
==t1=> s2
State nr.    2
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  0  1  0  0  0  1  0  0  1  0  0  1
==t2=> s3
State nr.    3
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  0  0  1  0  0  0  1  0  0  1  0  1
==t3=> s4
==t5=> s9
State nr.    4
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  0  0  1  0  0  0  0  1  0  1  1  0
==t4=> s5
==t5=> s7
State nr.    5
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  0  0  1  0  1  0  0  0  0  1  1  0
==t5=> s6
State nr.    6
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  0  0  0  1  1  0  0  0  0  1  1  0
==t6=> s1
State nr.    7
P.nr:  1  2  3  4  5  6  7  8  9 10 11 12
toks:  0  0  0  1  0  0  0  1  0  1  1  0
==t4=> s6
==t6=> s8
```

狀態轉移資訊

狀態浮標資訊

(b) 可達圖輸出格式

圖 3.8 可達圖輸出檔案格式

圖 3.8(b)可達圖輸出檔案包含狀態浮標資訊與狀態轉移資訊，然而規範測驗路徑軟體只需讀取狀態轉移資訊，並將狀態轉移資訊分別儲存為狀態間轉移點資訊、路徑成本資訊與狀態間路徑資訊，如圖 3.9 所示。表 3.2 為讀取可達圖檔案後輸出的三大資訊與對應程式陣列及意義。

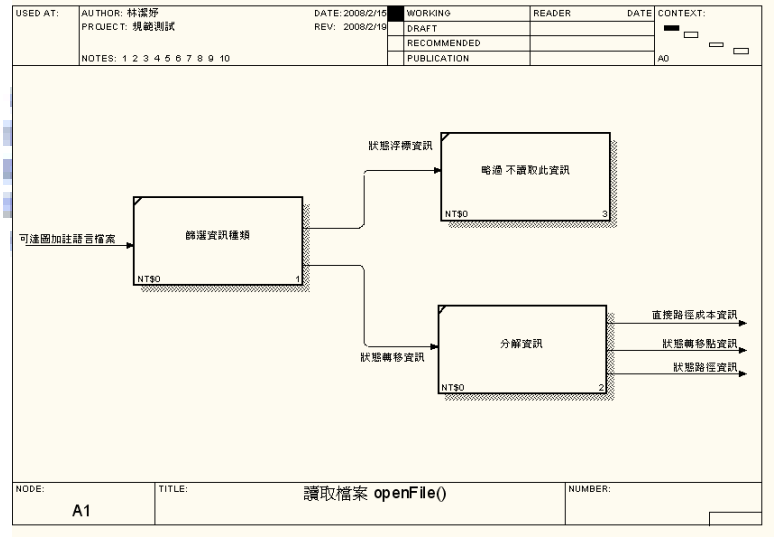


圖 3.9 讀取資料處理作業子流程 IDEF0 表示圖

表 3.2 狀態間轉移點資訊、路徑成本資訊與狀態間路徑資訊陣列儲存意義

	陣列名稱	意義
狀態轉移資訊	euler1[i][j]	由狀態 i 至狀態 j 有路徑：euler1[i][j]=1 無路徑則 euler1[i][j]=無限大
狀態間轉移點資訊	transition[i][j]＝轉移點	由狀態 i 至狀態 j 觸發的轉移點
路徑成本資訊	cost[i][j]	由狀態 i 至狀態 j 有路徑：cost[i][j]=1 無路徑則 cost[i][j]=無限大

3.4 活可達圖模組分析處理作業

活可達圖模組分析是爲了確保系統在進行狀態轉移時，不會落入鎖死區域，用以保證每個狀態都能經過一連串的觸發後回到原始狀態[10, 35]。因此程式在讀入可達圖資料後，須先將狀態分類，找出鎖死區域的狀態，並給予刪除之。因為下一階段運輸模式分析中，需要分類出系統狀態的供應點與需求點。對鎖死狀態而言，具有進入鎖死狀態的邊界，卻沒有由鎖死狀態出發的邊界，若執行運輸模式分析前未進行鎖死分析，則鎖死狀態也會被歸類為供應點($b_n > 0$)，然而鎖死狀態並無邊界能觸發到其他狀態，將導致程式無法得出歐依勒路徑。

活可達圖模組分析處理作業如圖 3.10 所示。首先由讀取檔案處理作業中輸入直接路徑成本資訊，進行路徑成本計算處理作業，由於有些狀態間並無直接轉移點轉換，而是需要經由多個轉移點轉換連結，狀態轉移路徑成本分析的演算法將於第 3.4.1 節詳細描述。由路徑成本計算處理作業輸出所有狀態路徑成本資訊，進行鎖死狀態篩選處理作業，篩選出路徑成本為無限大的路徑，標註為鎖死區域狀態，並於螢幕中顯示出來。

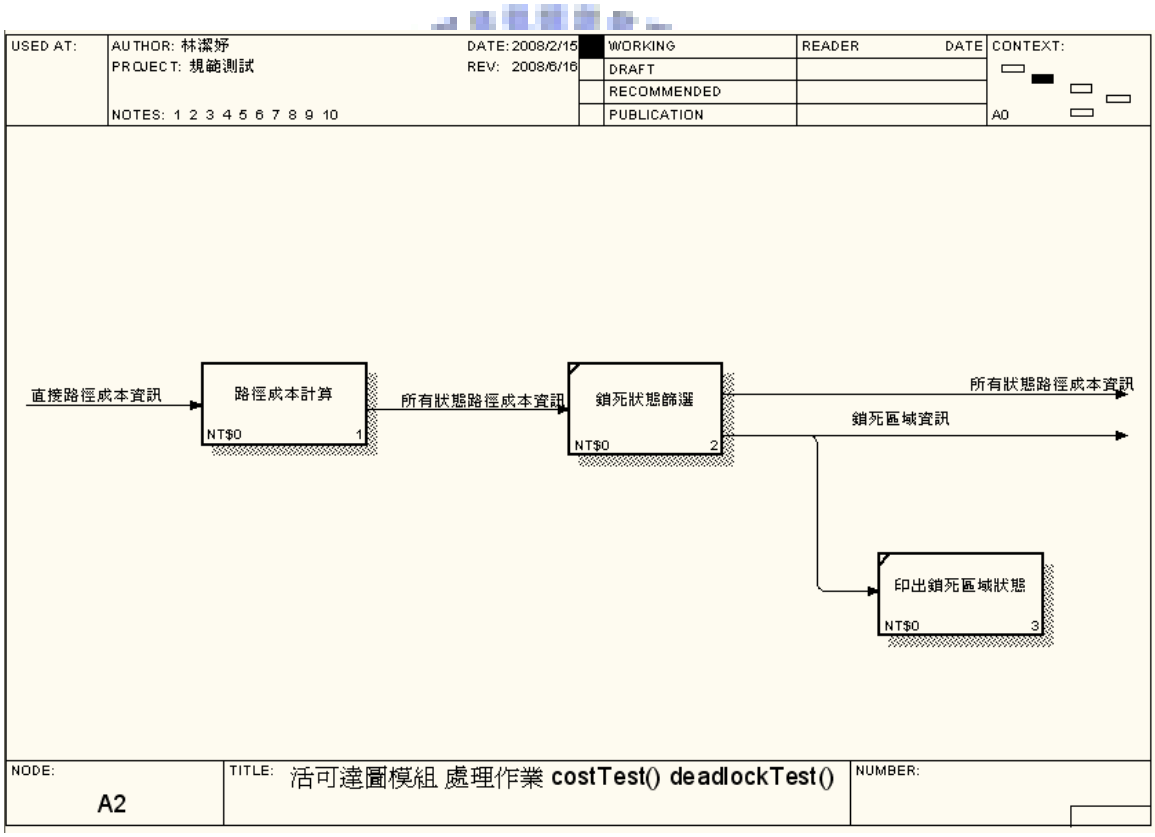


圖 3.10　活可達圖模組分析處理作業子流程 IDEF0 表達圖

3.4.1　路徑成本計算處理作業

本研究進行狀態轉移路徑成本分析的演算法如下所示：

1. 狀態若能經過一個觸發至另一狀態，則將連接兩狀態的邊界路徑成本設為 1。
2. 兩狀態間若無法透過一個觸發轉移，則先將連接兩狀態路徑成本設為無限大。
3. 若狀態(i)至狀態(j)路徑成本為 1，找尋可由狀態(j)觸發到的新狀態，將狀態(i)至(j)的路徑成本與狀態(j)至新狀態的路徑成本相加。若狀態(i)至新狀態的路徑成本低於舊的路徑成本，則取代之。圖 3.11 為程式設計路徑成本計算流程圖。
4. 重複步驟三，直到路徑成本不在變動為止。
5. 若兩狀態路徑成本依舊為無限大，則可判定這兩狀態的出發狀態為鎖死狀態。

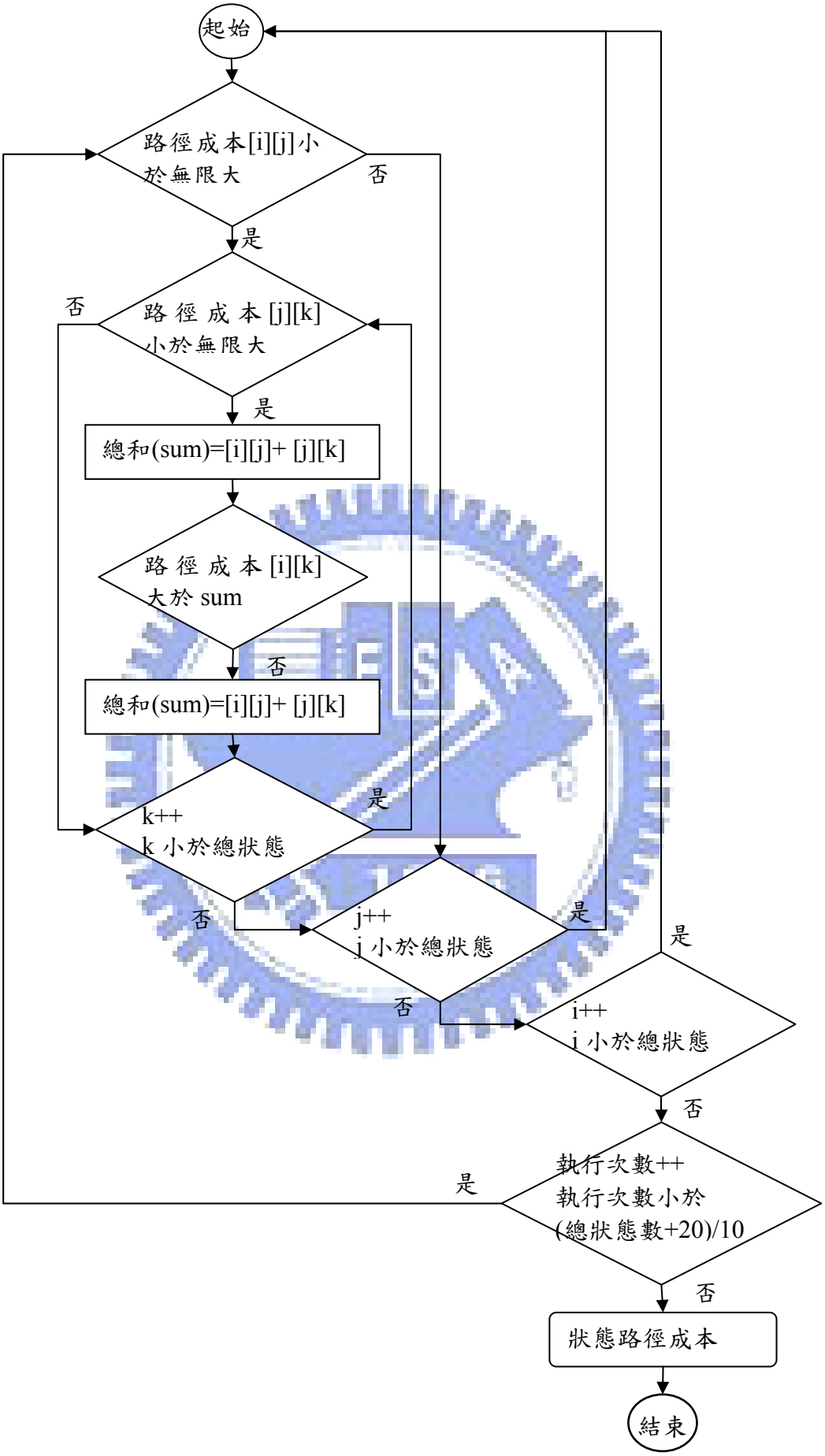


圖 3.11　路徑成本計算流程圖

鎖死狀態演算法進行方式如下所示。由圖 3.6(a)液體加熱系統斐氏圖模型的狀態轉移可達圖如第 3.3 節圖 3.8 所示，狀態以大寫 S 表示，由圖上可直接觀測出 S1 至 S2 有直接觸發路徑，因此將 S12 路徑成本設為 1，以此類推，S2S3、S3S4、S3S9、S4S5、S4S7、S9S7、S9S10、S5S6、S7S6、S7S8、S10S8、S6S1、S8S1 路徑成本初始狀態皆設為 1，其餘的狀態轉移路徑成本預設為無限大，將各狀態轉換的初始預設路徑成本表示如表 3.3，在此無限大的路徑成本設為 500。

表 3.3　液體加熱系統狀態轉移預設路徑成本

狀態轉換 成本路徑表	1	2	3	4	5	6	7	8	9	10
1	500	1	500	500	500	500	500	500	500	500
2	500	500	1	500	500	500	500	500	500	500
3	500	500	500	1	500	500	500	500	1	500
4	500	500	500	500	1	500	1	500	500	500
5	500	500	500	500	500	1	500	500	500	500
6	1	500	500	500	500	500	500	500	500	500
7	500	500	500	500	500	1	500	1	500	500
8	1	500	500	500	500	500	500	500	500	500
9	500	500	500	500	500	500	1	500	500	1
10	500	500	500	500	500	500	500	1	500	500

表 3.4　第一次路徑成本推導

狀態轉換 成本路徑表	1	2	3	4	5	6	7	8	9	10
1	500	1	2	500	500	500	500	500	500	500
2	500	500	1	2	500	500	500	500	2	500
3	500	500	500	1	2	500	2	500	1	2
4	500	500	500	500	1	2	1	2	500	500
5	2	500	500	500	500	1	500	500	500	500
6	1	2	3	500	500	500	500	500	500	500
7	2	3	4	500	500	1	500	1	500	500
8	1	2	3	500	500	500	500	500	500	500
9	3	4	5	500	500	2	1	2	500	1
10	2	3	4	500	500	500	500	1	500	500

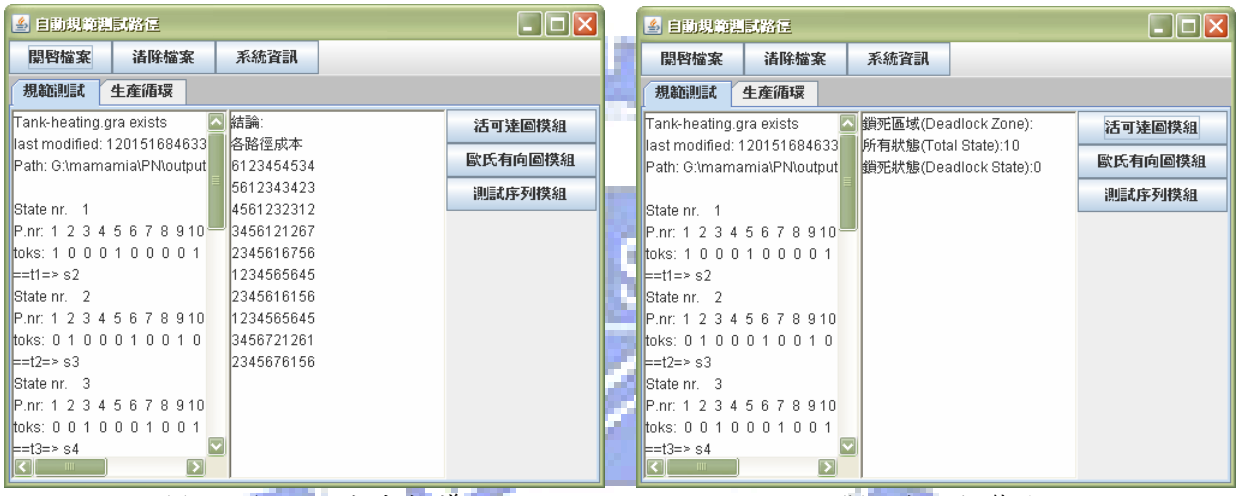
表 3.5　重複執行演算法步驟三得出最後路徑成本

狀態轉換 成本路徑表	1	2	3	4	5	6	7	8	9	10
1	6	1	2	3	4	5	4	5	3	4
2	5	6	1	2	3	4	3	4	2	3
3	4	5	6	1	2	3	2	3	1	2
4	3	4	5	6	1	2	1	2	6	7
5	2	3	4	5	6	1	6	7	5	6
6	1	2	3	4	5	6	5	6	4	5
7	2	3	4	5	6	1	6	1	5	6
8	1	2	3	4	5	6	5	6	4	5
9	3	4	5	6	7	2	1	2	6	1
10	2	3	4	5	6	7	6	1	5	6

執行步驟三，先找出路徑成本不為無限大 S1S2 = 1，再找到 S2S3 = 1，可推出新的路徑成本 S1S3 = S1S2 + S2S1 = 2，小於原先路徑成本 S1S3 = 1，取代原本的路徑成本。往下推演，S2S3 = 1，由 S3 可觸發的路徑為 S3S4 = 1，由此可推出新的路徑成本 S2S4 = S2S3 + S3S4 = 2，由 S2 至 S4 的新的路徑成本 2 低於原本預設路徑成本 500，因此 S2S4 路徑成本由 2 取代之。同理可推得其他狀態轉移路徑成本，表 3.4 為第一次執行演算法步驟三所推得的狀態轉移路徑成本。重複執行演算法步驟 4，直到所有的成本路徑表不在改變為止，可得到所有狀態轉移路徑成本如表 3.5 所顯示。

由表 3.5 路徑成本表可得到所有狀態轉移須觸發的最短成本路徑，如 S1 需要經過最少 6 次觸發才能再次回到 S1，S1 需要經過 2 次觸發後才能到 S3 的狀態。沒有狀態路徑成本為 500 的路徑，代表此液體加熱系統無鎖死狀態，任何狀態都能經一定的轉移至別的狀態。

液體加熱系統可達圖由程式算出路徑成本如圖 3.12(a)所示，所有成本皆小於 500，沒有成本無限大的路徑，因此液體加熱系統範例無鎖死狀態，更可由圖 3.12(b)可看到程式顯示，此範例共有 10 個狀態，鎖死狀態為 0 個。



(a) 各路徑成本推導　　(b) 鎖死狀態顯示

圖 3.12 通用程式路徑成本分析

3.5 歐氏有向圖模組處理作業

透過路徑成本演算分析後，確保可達圖內每個狀態轉移皆具有活性，無任何鎖死狀態存在。隨後，本節即要套用中國郵差問題將可達圖轉換為具有一筆畫性質的圖形，理論精神關鍵在於將每個狀態對稱化，也就是對於每個狀態而言，進入此狀態的邊界等於由此狀態出發的邊界，找出需要重複經過的路徑，並且最小化必須重複經過的路徑。本節以運輸模式的觀念找出最小重複路徑，再透過 3.6 節由修正 Fleury's 演算法找出最少成本一筆畫行經所有邊界的路徑。

規範測試在於確保狀態間轉移均能順利進行，亦即需要測試所有狀態與狀態間的轉移路徑，此概念與中國郵差問題相似，郵差送信需行經負責區域內的所有街道，勢必有些街道必須重複經過，為了找出能以最短路徑行經所有街道，首要步驟即將圖形內所有節點對稱化。對稱化問題解法有線性規劃模式與運輸模式兩種模式，針對狀態可達圖圖形屬性分析，運輸模式較線性規劃模式複雜度低並可節省計算時間，因此本節將討論如何利用運輸模式使可達圖圖形對稱化。

運輸問題為一種特殊的線性規劃問題[23]，建立由供應狀態至需求狀態的最短路徑成本，最後利用運輸模式演算法找出需要重複的最短路徑。運輸模式處理作業子流程如圖 3.13 所表示，首先輸入鎖死狀態資訊篩選出可達圖內供應狀態與需求狀態，其中，供應狀態為進入此狀態的邊界大於由此狀態出發的邊界；需求狀態為進入此狀態的邊界少於由此狀態出發的邊界。其次整合供應需求狀態資訊與所有狀態路徑成本資訊，建立其供需成本表，計算出最短加邊路徑資訊。最後將最小加邊路徑整合入狀態路徑資訊，使其最後輸出的狀態路徑資訊代表著狀態間可以重複行經的次數。

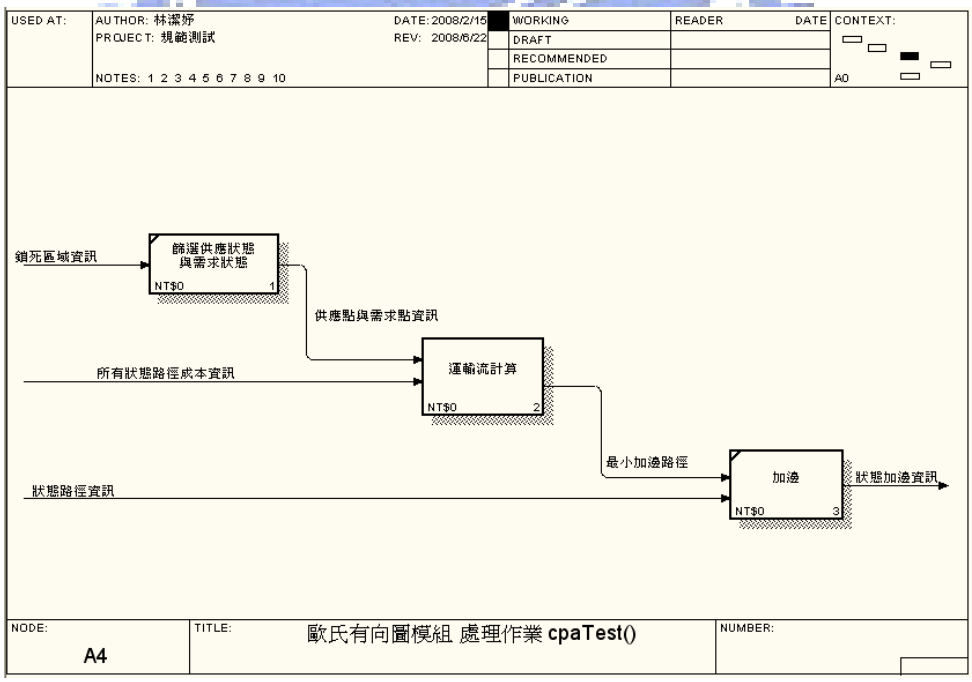


圖 3.13　歐氏有向圖模組處理作業子流程 IDEF0 表示圖

運用輸入的鎖死資訊刪除連結到鎖死狀態的路徑，以第 3.3 節圖 3.8(a)液體加熱系統可達圖中為例，並假設每一條街道的長度都一樣長，可以計算出進入各狀態進入的邊界數以表 3.6 所示，狀態 1 的有 2 條進入邊線 1 條出發邊線，進入邊線大於出發邊線，因此狀態 1 為供應狀態，並求得其需求供應量為 2 – 1 = 1。同理可找出需求點為狀態 3、狀態 4、狀態 9，供應點為狀態 1、狀態 6、狀態 8，以及各個供應需求量。

表 3.6　定義供應點與需求點

狀態	進入節點的邊線	由節點出發的邊線	標註	供應需求量
1	2	1	供應點	1
2	1	1		0
3	1	2	需求點	1
4	1	2	需求點	1
5	1	1		0
6	2	1	供應點	1
7	2	2		0
8	2	1	供應點	1
9	1	2	需求點	1
10	1	1		0

3.5.1　運輸模式計算處理作業

圖 3.14 顯示進行運輸模式計算作業流程，輸入所有狀態路徑成本資訊，將供應點至需求點所需的路徑成本繪製成表，完成供需點路徑成本後，即可利用運輸模式演算法找出最短路徑成本。運輸模式演算法包含三大步驟，即初始解選定、最佳性測試與反覆。由供需求狀態的路徑成本表中，可藉由初始解演算法找出運輸問題初始解，然而初始解不一定為問題最佳解，因而初始解需進行最佳解檢測，若目前解不為最佳解，則進行反覆；若目前解為最佳解，則停止。運輸問題的初始解選定，依第 2.3.3.2 節運輸模式問題解法的分析，本研究以 Russell 近似法求得運輸問題初始解。

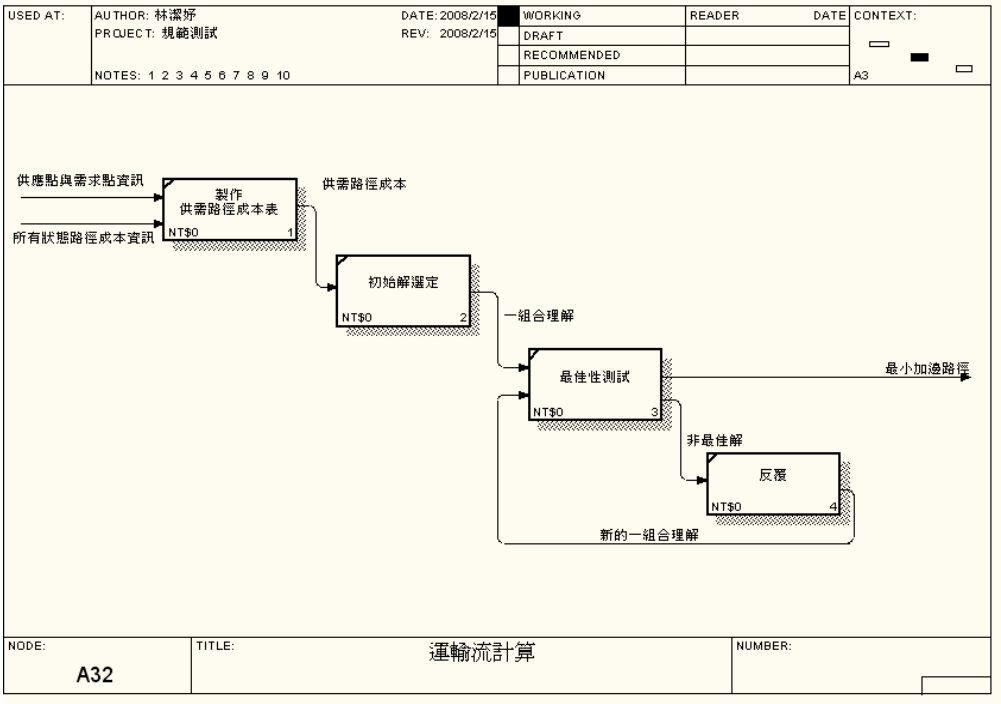


圖 3.14　運輸模式計算處理作業子流程 IDEF0 表示圖

由第 3.5 節找出的供應需求點資訊整合所有狀態路徑成本資訊，繪製出如表 3.7 供應點與需求點路徑成本與需求表，表中資訊意義為由供應狀態 1 至需求狀態 3 需要花費路徑成本為 2，由供應狀態 6 至需求狀態 3 需要花費路徑成本為 3，而供應狀態 1、6、8 的供應量各為 1，需求狀態 3、4、9 的需求量各為 1。

表 3.7　供應點與需求點路徑成本與需求表

供\需	3	4	9	供應量
1	2	3	3	1
6	3	4	4	1
8	3	4	4	1
需求量	1	1	1	

表 3.8　初始成本路徑計算

(a)　初始路徑成本及供應量

成本(c_{ij})				
供\需	3	4	9	供應量
1	2	3	3	1
6	3	4	4	1
8	3	4	4	1
需求量	1	1	1	

(b)　初始成本扣除各行列的最大值

$c_{ij} - u_i - v_j$				
供\需	3	4	9	u_i
1	-4	-4	-4	3
6	-4	-4	-4	4
8	-4	-4	-4	4
v_j	3	4	4	

表 3.9　第二次成本路徑計算

(a)　第二次路徑成本及供應量

成本(c_{ij})				
供\需	3	4	9	供應量
1	ˇ			
6		4	4	1
8		4	4	1
需求量		1	1	

(b)　第二次成本扣除各行列的最大值

$c_{ij} - u_i - v_j$				
供\需	3	4	9	u_i
1				
6		-4	-4	4
8		-4	-4	4
v_j		4	4	

以第 3.3 節圖 3.8(a)液體加熱系統可達圖中為例，由第 3.5 節求出其供需狀態與供需量如表 3.8(a)，以 Russell 近似法計算，找出每一行列最大成本值設為的 u_i 及 v_j，並且計算各行列路徑成本扣除其行列的最大值以求出 $c_{ij} - u_i - v_j$，找出其最小 $c_{ij} - u_i - v_j$。如表 3.8(b)所示，其各行列的 $c_{ij} - u_i - v_j$ 均為-4，因此選擇其路徑成本最少為行經路徑。選擇路徑成本

為 2，由供應狀態 1 至需求狀態 3 為第一個基變數，依照其供應量等於需求量為 1，設定由狀態 1 至狀態 3 需重覆經過一次。並且刪除列 1 與行 1 的所有成本值如表 3.9(a)，以便於進行下一個基變數選定。

進行第二基變數選定，依剩餘的行列找出各行列最大值設為 u_i 及 v_j，並計算 $c_{ij} - u_i - v_j$，計算值如表 3.9(b)，因其各欄位 $c_{ij} - u_i - v_j$ 值相同，且剩餘欄位的路徑成本也相同，因此隨意選擇一條為加邊路徑。在此選定第 2 欄第 2 列為其基變數，依供應量等於其需求量為 1，因此建立由狀態 6 至狀態 4 的重複行徑路徑。同理進行下一次反覆找出第三個基變數，如表 3.10。完成反覆後，可得到表 3.11 代表所有需要重複的路徑以及其重複的次數，表中顯示需重複路徑為由狀態 1 至狀態 3 重複一次、由狀態 6 至狀態 4 重複一次、由狀態 8 至狀態 9 重複一次。

表 3.10　第三次成本路徑計算

(a)　第三次路徑成本及供應量

成本(c_{ij})				
供\需	3	4	9	供應量
1	ˇ			
6		ˇ		
8			4	1
需求量			1	

(b)　第三次成本扣除各行列的最大值

$c_{ij} - u_i - v_j$				
供\需	3	4	9	u_i
1				
6				
8			-4	4
v_j			4	

表 3.11　重複行經的路徑

供\需	3	4	9	供應量
1	1			1
6		1		1
8			1	1
需求量	1	1	1	

3.5.2　運輸模式最佳解測試

最佳性檢測爲了查驗目前解是否為最佳解，發現目前解並非問題模式的最佳解，即要進行下一反覆的步驟，以找出較佳的可行解。以目前解進行最佳解測試，找到出現列數最多之 u_i，設此 $u_i = 0$，利用所有基變數 x_{ij} 之聯立方程式 $c_{ij} = u_i + v_i$，產生所有的值，若所有 (i,j)的非基本變數 x_{ij} 之 $c_{ij} - u_i - v_i \geqq 0$，則目前解為最佳解，故停止最佳性檢定，否則，則進行反覆。反覆：首要步驟為選定進入的基變數，選出具有最大負值的 $c_{ij} - u_i - v_i$ 之非基變數；第二步驟為選定退出的基變數，找出進入基變數增大時，保持可行性的連鎖反應，從捐贈格中選出分配量較小者；最後即可找出新的可行解，並再重複最佳解測試，確定目前解是否為最佳解。

表 3.12　重複行徑路徑最佳解測定

(a)　最佳解測定成本與路徑表

成本(c_{ij})				
供\需	3	4	9	供應量
1	2 1	3	3	1
6	3 0	4 1	4	1
8	3 0	4	4 1	1
需求量	1	1	1	

(b)　最佳解測定 $c_{ij} - u_i - v_i$

$c_{ij} - u_i - v_i$				
供\需	3	4	9	u_i
1	0	1	1	2
6	3	0	0	0
8	3	0	0	0
v_i	0	4	4	

進行第 3.5.1 所求得路徑最佳解測定如表 3.12(a)所示，各路徑成本列於各欄左上角，需要重複的路徑顯示於各欄右下角，利用所有基變數 x_{ij} 之聯立方程式 $c_{ij} = u_i + v_i$，求得各行列的 u_i 及 v_i 如表 3.12(b)，以及所有(i,j)的非基本變數 x_{ij} 之 $c_{ij} - u_i - v_i$。求得所有(i,j)的非基本變數 x_{ij} 之 $c_{ij} - u_i - v_i$ 皆大於等於零，故由第 3.5.1 節採用 Russell 近似法求得運輸的初始解為此範例的最佳解。

3.5.3　運輸模式處理作業驗證

計算運輸模式是特殊的線性規劃模型[23]。目標式為最小化成本乘於路徑的總和，由供應點(i)出發至所有需求點的供應量總和要等於此供應點的最大供應量，同理，由需求點(i)獲得所有供應點的供應量總和要等於此供應點的最大需求量。以表 3.6 供應點與需求點路徑成本與需求表為例，可描述為圖 3.15 左邊的線性方程式，利用線性規劃軟體 Lindo 計算出的答案如圖 3.15 右邊所示，最小成本為 10，其解為 x11 = 1、x22 = 1、x33 = 1，等同於增加下列三條路徑：由狀態 1 至狀態 3、由狀態 6 至狀態 4、由狀態 8 至狀態 9，其解與本研究撰寫的程式模型得到的運輸模式最佳解相同。

圖 3.16(a)為本研究撰寫的運輸模式計算，首先顯示供需點路徑成本表，需要加邊的最小成本計算，最小成本為 10 表示最小需要增加 10 條路徑，最後顯示需要增加的路徑，加邊路徑為狀態 1 經 t1 觸發至狀態 2，狀態 2 再經 t2 觸發至狀態 3。將所有需要加邊的路徑繪至此系統的可達圖如圖 3.16(b)，增加 10 條虛線路徑，可得到進入每個節點的路徑數等於由此節點出發的路徑數，轉化為具有對稱性的可達圖。

運輸模式解法可將原先具有中國郵差問題的可達圖對稱化，對稱化可達圖可用一筆畫行經所有路徑，換言之，可求得此可達圖歐依勒路徑。

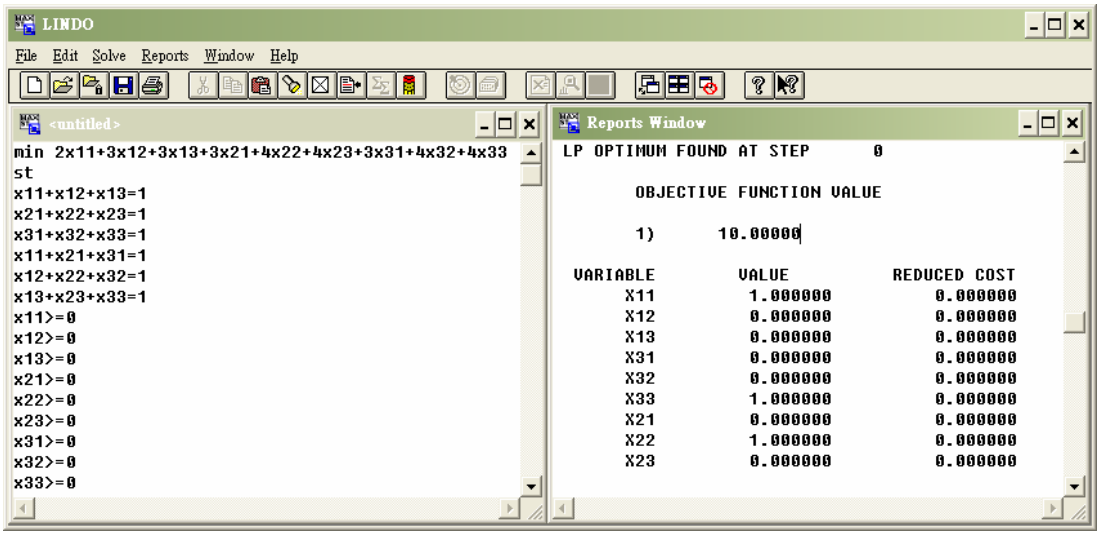
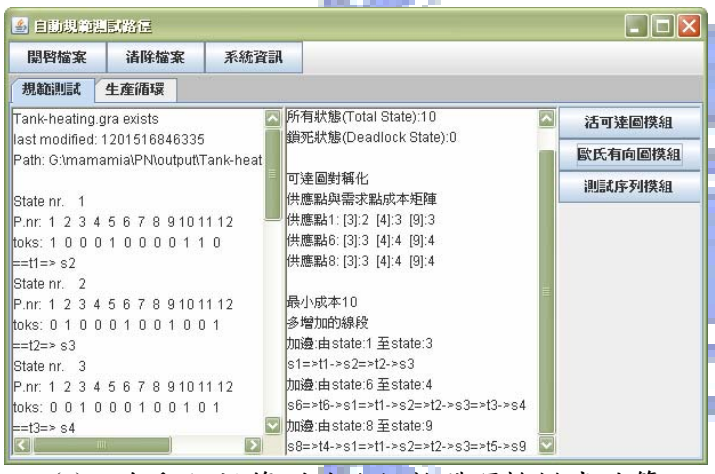
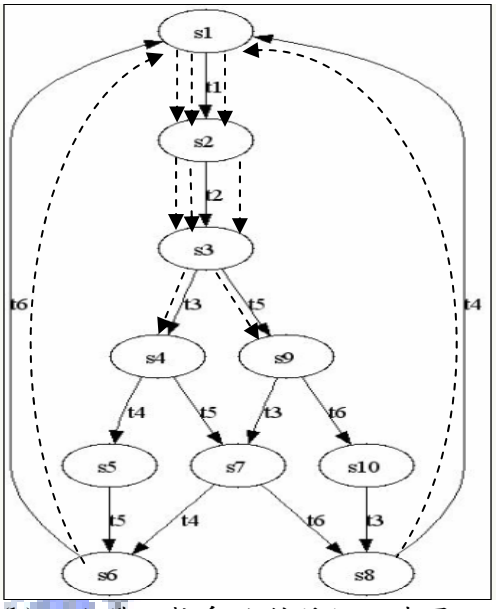


圖 3.15　Lindo 驗證液體加熱系統運輸模式解



(a)　自動化規範測試路徑軟體運輸模式計算



(b)　液體加熱系統對稱化可達圖

圖 3.16　通用程式運輸模式計算

3.6 測試路徑模組處理作業

一個圖形是否具有一條歐依勒路徑的充分必要條件為每一點均為偶數邊界。爲了使可達圖均具有歐依勒路徑，本研究運用第 3.5 節歐氏有向圖模組處理作業中，運輸模式以最小的成本增加必須重複行經的路徑，使得每個狀態都具有偶數邊界，本節將介紹運用第 2.3.2.1 節 Fleury's 演算法得到一條歐依勒路徑。Fleury's 演算法的觀念為可任選一狀態出發，任選一條由此狀態出發的邊界，確認此邊界是否為橋。若非橋，記錄此行經路徑並將此邊界刪除，重複選擇下一出發邊界；若此邊界為橋，則選擇此狀態的其他出發邊線，若此狀態已無其他連接邊線，則可行經橋並將此狀態刪除。

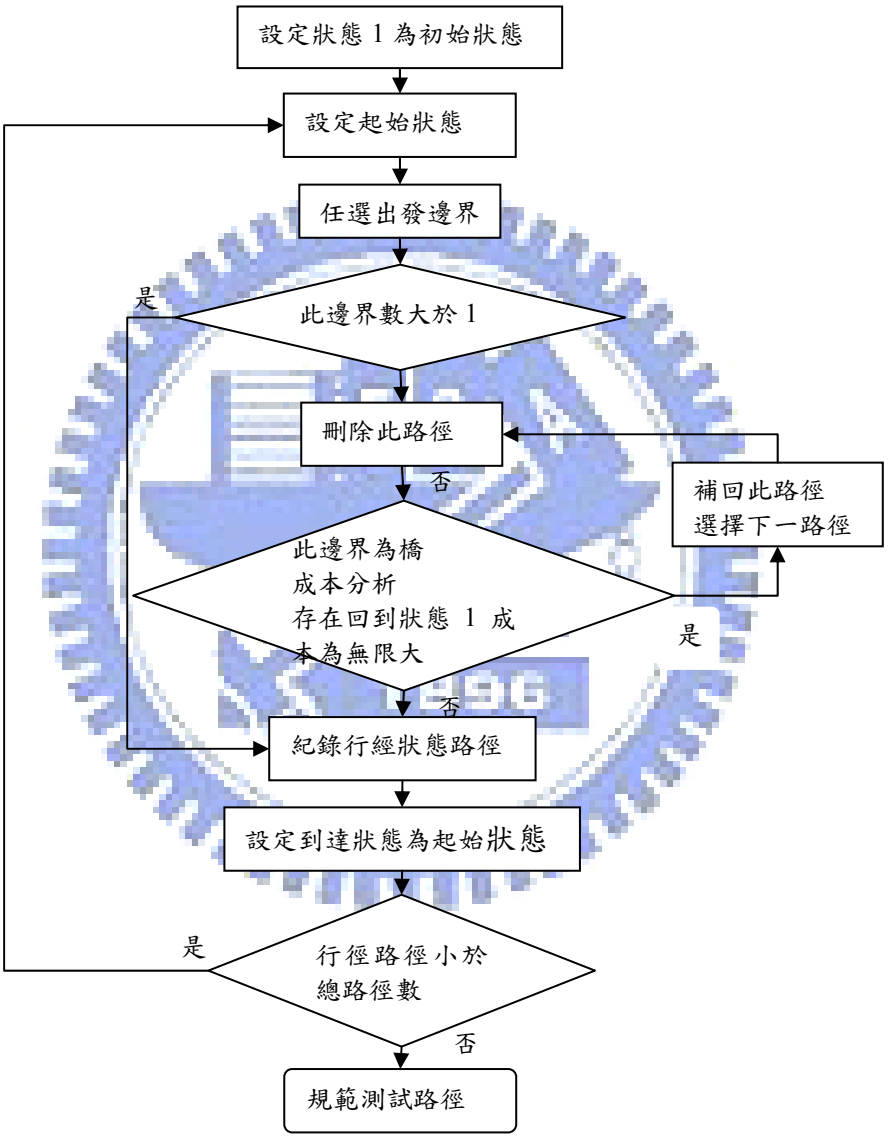
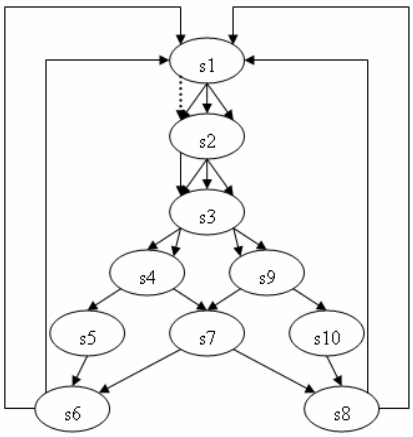


圖 3.17　歐氏迴徑計算處理作業流程圖

本研究測試路徑模組處理作業流程圖如圖 3.17，並針對可達圖特性，修正 Fleury's 演算法。將出發狀態設為初始狀態 S1，找尋一條由此狀態出發邊界。判別邊界是否為橋的則是計算所有狀態回到初始狀態的路徑成本，若有任一狀態回到初始狀態的路徑成本為無限大，表示刪除此路徑會造成圖形的不連續，代表此路徑為橋，因此重新選擇另一條由此狀態出發的邊線。若任一狀態皆可經一定觸發回到初始狀態，則記錄此路徑，並以此路徑為下一個出發狀態，重複尋找路徑作業。當行經的路徑數等於可達圖面所有可走的路徑數時，停止作業。輸出行經路徑，此路徑則為歐依勒路徑。

延續第 3.5 節圖 3.16 得出的對稱化可達圖進行測試路徑模組處理作業。以 S1 為初始狀態，出發路徑為 S1→S2，如圖 3.18(a)虛線所示，刪除此路徑後進行橋的分析如圖 3.18(b)，所有狀態均可藉由一定的轉換回到 S1，S1→S2 不為橋，記錄此路徑，以 S2 為初始狀態。同理可求得 S2→S3 不為橋，記錄此路徑 S1→S2→S3，以 S3 為初始狀態。

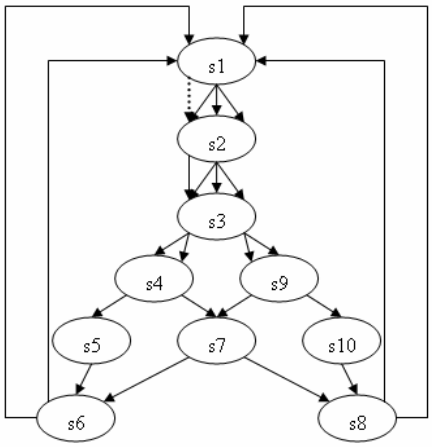


(a) 路徑選擇 1

狀態轉換 成本路徑表	S1
S1	6
S2	5
S3	4
S4	3
S5	2
S6	1
S7	2
S8	1
S9	3
S10	2

(b) 成本分析 1

圖 3.18 Fleury's 演算法 1

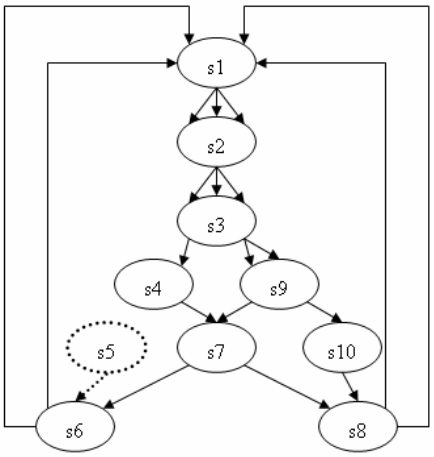


(a) 路徑選擇 2

狀態轉換 成本路徑表	S1
S1	6
S2	5
S3	4
S4	3
S5	2
S6	1
S7	2
S8	1
S9	3
S10	2

(b) 成本分析 2

圖 3.19 Fleury's 演算法 2



(a) 路徑選擇 3

狀態轉換 成本路徑表	S1
S1	6
S2	5
S3	4
S4	3
S5(刪除)	
S6	1
S7	2
S8	1
S9	3
S10	2

(b) 成本分析 3

圖 3.20 Fleury's 演算法 3

以 S3 為初始狀態，出發路徑有兩條為 S3→S4 及 S3→S9，可任選一條，本研究選擇狀態數字小為先即 S3→S4，如圖 3.19(a)虛線所示，刪除此路徑後進行橋的分析如圖 3.19(b)，所有狀態均可藉由一定的轉換回到 S1，S3→S4 不為橋，記錄此路徑 S1→S2→S3→S4，以 S4 為初始狀態。同理可求得 S4→S5 不為橋，記錄此路徑 S1→S2→S3→S4→S5，以 S5 為初始狀態。

以 S5 為初始狀態，出發路徑為 S5→S6，可任選一條，如圖 3.20(a)虛線所示，刪除此路徑後進行橋的分析如圖 3.20(b)，由於 S5 刪除路徑 S5→S6 後不存在任何路徑，因此刪除 S5，不考慮 S5 回到初始狀態的成本路徑，其餘狀態均可藉由一定的轉換回到 S1，S5→S6 不為橋，記錄此路徑 S1→S2→S3→S4→S5→S6，以 S6 為初始狀態。同理可求得所有路徑走法如圖 3.21，規範測試歐依勒路徑 S1→S2→S3→S4→S5→S6→S1→S2→S3→ S4→S7→S6→S1→S2→S3→S9→S7→S8→S1→S2→S3→S9→S10→S8→S1。

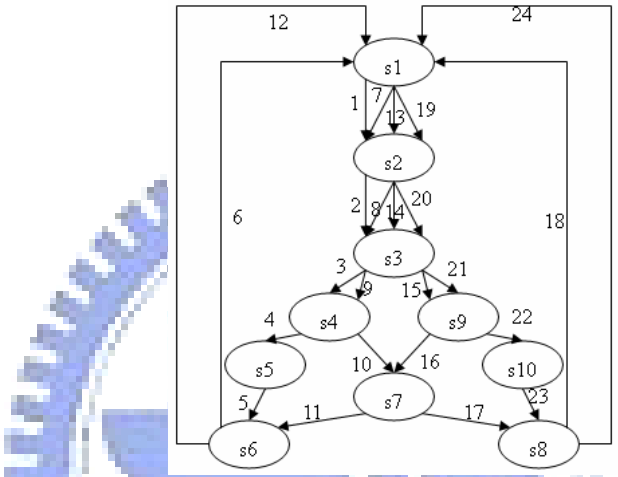


圖 3.21　液體加熱系統規範測試歐依勒路徑

圖 3.21 對應圖 3.22 由爪哇程式自動化規範測試路徑軟體的得出的歐依勒路徑相同，自動化測試路徑軟體同時顯示狀態間轉移需要觸發的轉移點，如 S1 經觸發點 t1 轉換至 S2，S2 經觸發點 t2 轉換至 S3，由此可以提醒管理者規範測試執行步驟，並且可以最短的路徑測試完所有狀態轉移。

圖 3.22　液體加熱系統自動化規範測試歐依勒路徑

第四章 生產循環分析

本章將說明讀取狀態可達圖資訊後，進一步藉由圖論的觀念，找出具有相同的生產循環。4.1 節說明生產循環分析應用流程，4.2 節說明生產循環分析處理作業程式架構，了解等效的生產循環後，找出最佳的派工法則。

4.1 生產循環分析應用流程

生產循環分析應用流程如圖 4.1 所示，首先透過第 3.4 節鎖死分析處理作業步驟，讀取具有活性的可達圖檔案，接著利用展開樹(Spanning Tree)原理，找出可達圖中所有可能的狀態轉移循環。並由文獻回顧第 2.4 節生產循環得知，利用圖論中的符號矩陣去找出可達圖的所有循環，檢查符號矩陣的次幕並對照對角線元素值便可的循環數。換句話說，由可達圖中任一狀態出發，都可經由同樣轉換次數回到原狀態的生產循環，才具有相同的生產製造效果。因此藉有展開樹找出可達圖中所有狀態轉移路徑後，必須檢驗所有狀態經由轉移回到原狀態，是否具有相同的轉移次數。若無，則此可達圖無相同的生產循環；若有，則可列出所有具等效的生產循環。

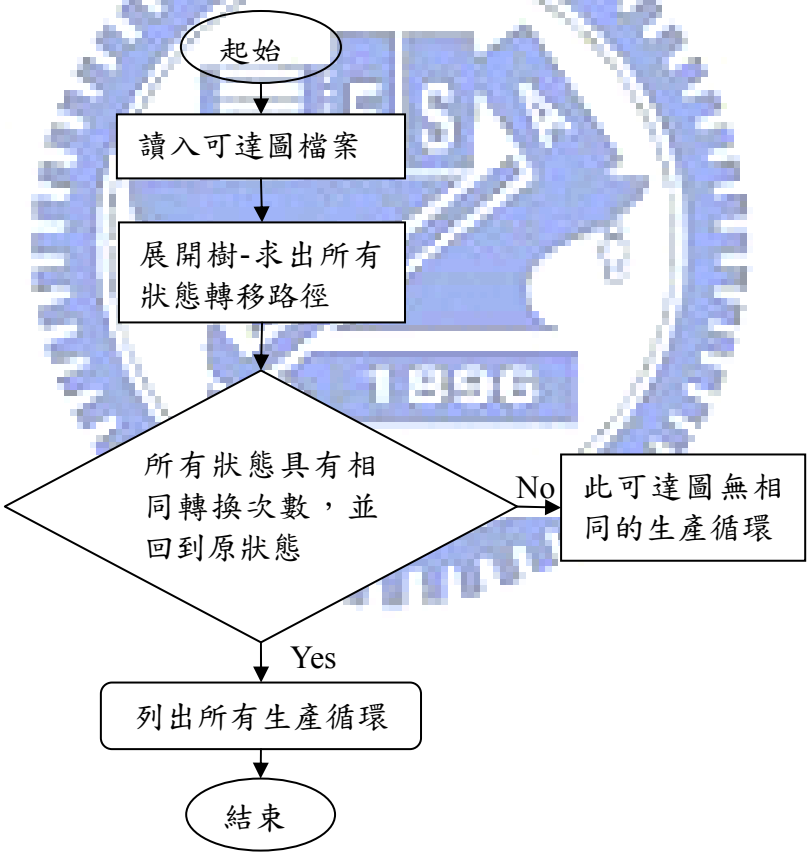


圖 4.1 生產循環流程圖

4.2 循環分析模組處理作業程式架構

首先，系統讀取可達圖檔案後，將透過規範測試路徑的處理作業中的活可達圖模組處理作業，刪除具有鎖死問題的狀態，留下具有活性的可達圖狀態。生產循環分析處理作業讀取此活可達圖檔案，為了達成圖論中符號矩陣所代表的意義，藉由展開樹的觀念，由任一狀態出發，並記錄所有狀態轉移路徑，當狀態轉移回到原狀態時，則停止搜尋。以圖 4.2

生產循環分析可達圖為例，可表示為圖 4.3 的符號矩陣，但因符號矩陣的觀念較難用程式語言表達，因此以展開樹的想法如圖 4.4 所示，由狀態一為初始狀態的展開樹，具有三條路徑。經由狀態多次轉換，可以找到路徑[1][2][3][4][1]，狀態一可經由五次轉換回到原狀態，因此加以記錄，而展開樹的另一條路徑[1][2][3][5][6][2]，則會呈現重複的循環，故不加以考慮。第三條路徑可經由更多次的轉換，但因為此路徑略長，在此省略討論。

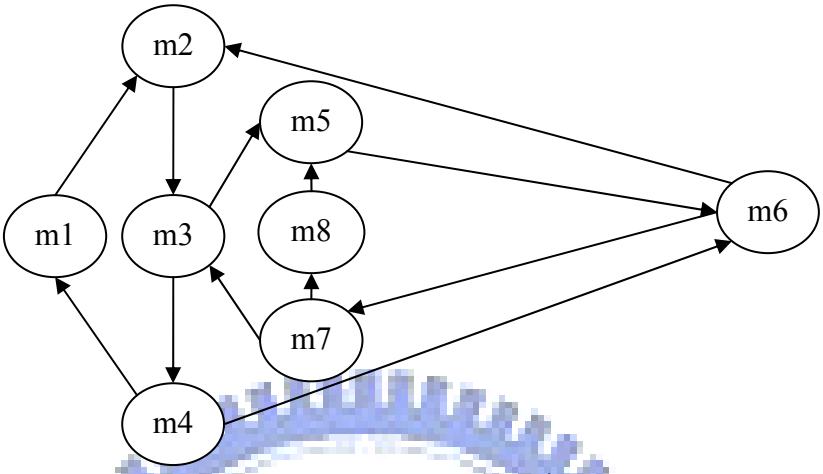


圖 4.2　生產循環可達圖範例

$$S = \begin{bmatrix} 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 23 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 34 & 35 & 0 & 0 & 0 \\ 41 & 0 & 0 & 0 & 0 & 46 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 56 & 0 & 0 \\ 0 & 62 & 0 & 0 & 0 & 0 & 67 & 0 \\ 0 & 0 & 73 & 0 & 0 & 0 & 0 & 78 \\ 0 & 0 & 0 & 0 & 85 & 0 & 0 & 0 \end{bmatrix}$$

圖 4.3　初始轉換-符號矩陣 S

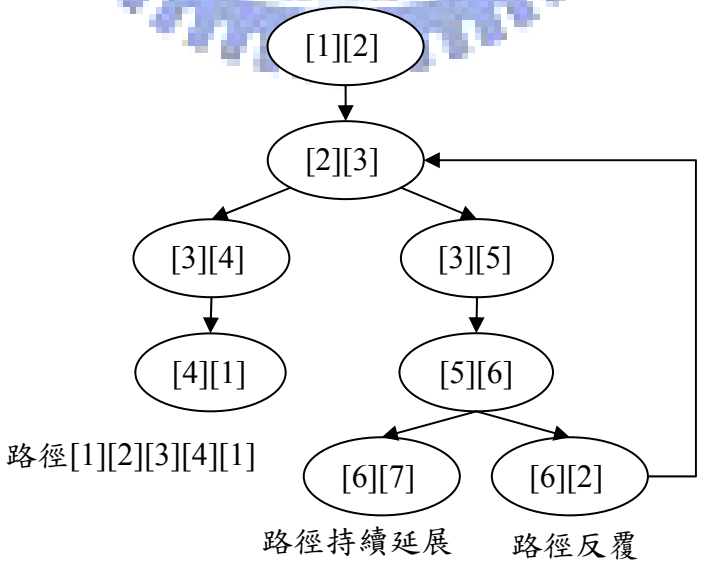


圖 4.4　展開樹範例

透過展開樹的方法，找出以所有狀態為起始狀態，並回到原狀態的路徑，發現可達圖以圖 4.4 為例，各狀態可經過五次轉換回到原狀態，轉換路徑如表表 4.1 所示，利用爪哇程式設計中的 Set 功能，刪除重複的組合，可得六條生產循環路徑，S1=[1][2][3][4]，S2=[3][5][6][7]，S3=[3][4][6][7]，S4=[2][3][4][6]，S5=[2][3][5][6]，S6=[5][6][7][8]。程式表示如，列出六條不重複的生產循環路徑。

表 4.1　所有狀態具有相同轉換次數的路徑

初始狀態	路徑	初始狀態	路徑
狀態一	[1][2][3][4][1]	狀態五	[5][6][7][3][5]
狀態二	[2][3][5][6][2]	狀態五	[5][6][2][3][5]
狀態二	[2][3][4][1][2]	狀態五	[5][6][7][8][5]
狀態二	[2][3][4][6][2]	狀態六	[6][7][3][5][6]
狀態三	[3][4][1][2][3]	狀態六	[6][7][3][4][6]
狀態三	[3][4][6][7][3]	狀態六	[6][2][3][4][6]
狀態三	[3][4][6][2][3]	狀態六	[6][2][3][5][6]
狀態三	[3][5][6][2][3]	狀態六	[6][7][8][5][6]
狀態三	[3][5][6][7][3]	狀態七	[7][8][5][6][7]
狀態四	[4][1][2][3][4]	狀態七	[7][3][5][6][7]
狀態四	[4][6][2][3][6]	狀態七	[7][3][4][6][7]
狀態四	[4][6][7][3][4]	狀態八	[8][5][6][7][8]

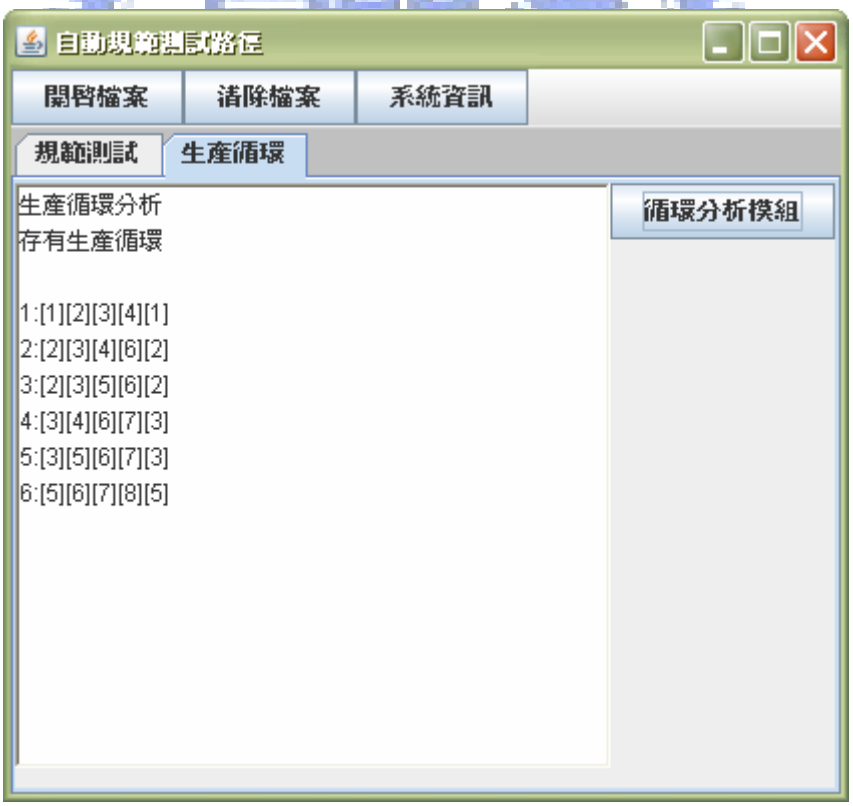


圖 4.5　程式執行圖

第五章　範例實際操作

本章將舉兩個自動化執行系統案例，依據第 3.1 節的流程作為本論文的實例操作說明。兩案例分別為自動化導引車輛與彈性製造系統產生其規範測試路徑：第 5.1 節將說明自動化導引車輛範例，其系統模型具有選擇性路線，並存在鎖死問題；第 5.2 節將說明彈性製造系統，其原始系統模型具有複雜的可達圖，同時存有鎖死問題，並討論系統模型未簡化前與簡化後規範測試路徑執行效果。

5.1 自動導引車輛範例

本節將討論自動導引車輛範例，研究其可達圖狀態。若存有鎖死問題，提供鎖死狀態區域。進行修正刪除其鎖死區域後，確保其可達圖無鎖死問題後，進而產生其規範測試一筆畫路徑。其基本性質探討與其轉移點不變量分析、暫存點不變量分析與可程式控制器連線可參考 2008 年陳氏所發表的論文[7]。具有鎖死問題的自動導引車範例取自 1990 年 Viswanadham 所發表的論文[35]。自動化系統中，因為資源共用會導致系統產生鎖死問題，本範例如圖 5.1 所示：上下貨區(L/U Station)，未加工元件與加工元件放置區；加工機(NC Machine)，一次加工一個元件；自動導引車輛(AGV)，一次搬運一個元件，可運送未加工元件由上下貨區至加工機，並裝載已加工元件由加工機至上下貨區。本範例斐氏圖運作模式如圖 5.2，鎖死狀況發生如下，(1)由自動導引車輛由上下貨區運送未加工元件至加工機，加工機開始加工此元件，稱之元件 1；(2)自動導引車輛空車回上下貨區，又載著另一未加工元件稱之元件 2 至加工機，此時加工機正在執行元件 1 等待卸貨至自動導引車輛，而元件 2 則等著進行加工，產生鎖死現象。圖 5.2 斐氏圖暫存點與轉移點如表 5.1 說明之。

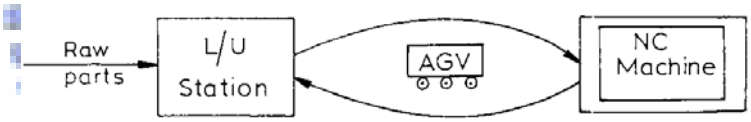


圖 5.1　自動導引車系統示意圖

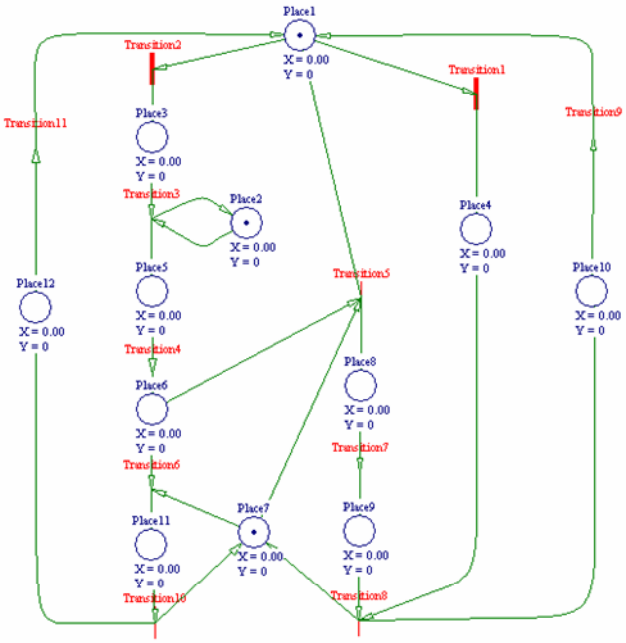


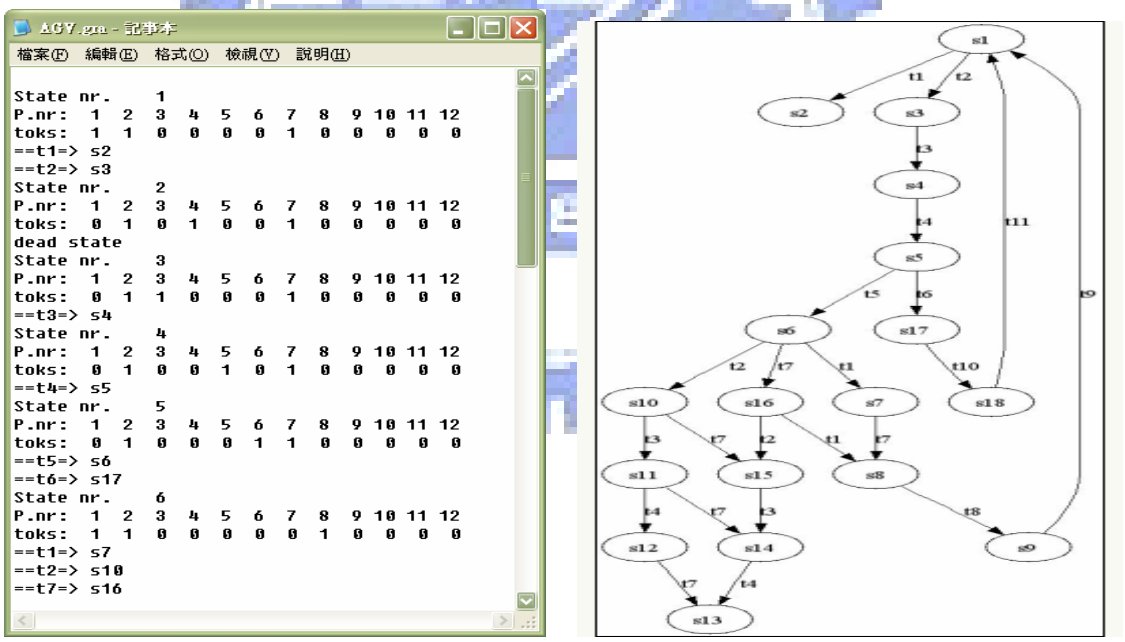
圖 5.2　自動導引車輛範例斐氏圖模型

表 5.1　自動車輛導引系統斐氏圖模型的暫存點與轉移點說明

暫存點意義		轉移點意義	
Place1	自動導引車輛(AGV)空閒	Transition1	指定 AGV 運送已加工機台
Place2	未加工工件已準備	Transition2	指定 AGV 運送未加工機台
Place3	AGV 運送未加工工件	Transition3	AGV 裝載未加工工件
Place4	AGV 運送已加工工件	Transition4	AGV 運送未加工工件至加工機
Place5	AGV 運送未加工工件至加工機途中	Transition5	加工機開始加工，AGV 離去
Place6	AGV 與加工機旁等待上裝未加工工件	Transition6	加工機開始加工，AGV 等待於加工機旁
Place7	加工機空閒	Transition7	加工機加工完成
Place8	加工機加工中，AGV 離去	Transition8	AGV 卸載已加工工件
Place9	加工機加工完成等待 AGV 到達	Transition9	AGV 運送已加工工件至上下貨區
Place10	AGV 運送已加工工件	Transition10	加工機加工完成，未離去之 AGV 卸載已加工工件
Place11	加工機加工中，AGV 於一旁等待	Transition11	未離去之 AGV 運送已加工工件至上下貨區
Place12	位離去之 AGV 運送已加工工件		

此自動導引車輛系統存有兩種派工策略，其一為車輛運送未加工元件至加工機後，等待其加工完成後，卸載加工完元件一起回到上下貨區；其二為車輛運送未加工元件至加工機後，先行離去，等待加工機加工完成後，再至加工機處卸載元件回上下貨區。

依照第 3.1 節專案分析流程，進行系統軟體實作分析。由軟體 P3 繪製而成圖 5.2 斐氏圖模型，經由 INA 軟體分析產生可達圖檔案如圖 5.3(a)，圖 5.3(b) 自動導引車輛系統之可達圖。



(a)　可達圖檔案　　(b)　可達圖

圖 5.3　自動導引車輛範例

圖 5.3 為自動導引車輛系統的可達圖檔案，點選開啟檔案按鍵，將可達圖 AGV.gra 檔案讀入開啟。開啟成功，檔案資料於程式左方顯示 AGV.gra 檔案存在與檔案內容。

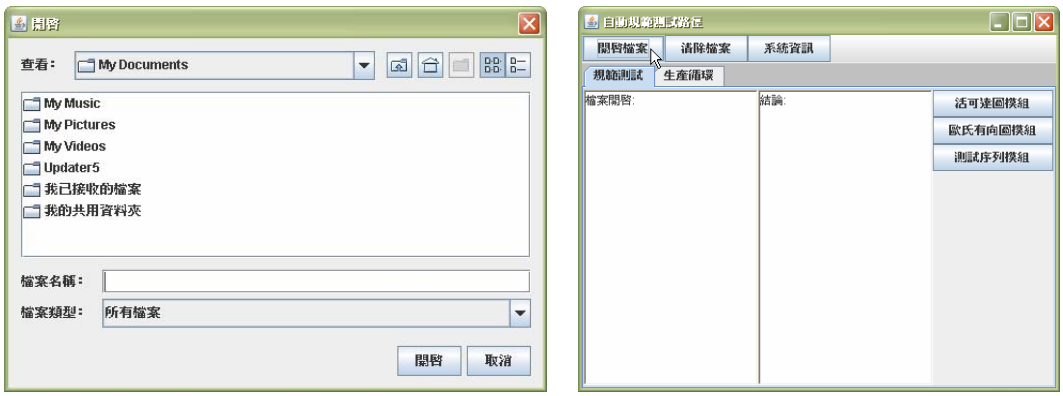


圖 5.4　載入自動導引車輛範例之可達圖檔案

開啟檔案成功，點選活可達圖模組分析按鈕如圖 5.5，系統列出自動導引車輛系統範例中共 18 個狀態，擁有 7 個鎖死狀態，其中為 S2、S10、S11、S12、S13、S14、S15。

圖 5.5　自動導引車輛範例之鎖死狀態分析

活可達圖模組分析完成後，系統會自動刪除所有鎖死狀態，點選歐氏有向圖模組按鈕如圖 5.6，進行運輸模式分析，找出所有需要加邊的最短路徑，使得自動導引車輛系統範例可達圖具有對稱性。系統列出自動導引車輛系統範例中供應點為 S1、S8，需求點為 S5、S6，路徑成本 S1S5=3、S1S6 = 4、S8S5 = 5、S8S6 = 6。求出最小加邊路徑成本為 9，需要多增加的線段為 S1S5 路徑為 S1→S3→S4→S5 以及 S8S6 路徑為 S8→S9→S1→S3→S4→S5→S6。

圖 5.6　自動導引車輛範例之中國郵差問題運輸模式分析

運用歐氏有向圖模組解決具有中國郵差問題的可達圖，使得自動導引車輛系統範例可達圖具有對稱性，即可求得此歐依勒路徑。如圖 5.7 按測試序列模組按鈕，系統即可顯示除自動導引車輛系統範例可行經所有邊界的歐依勒路徑以及狀態間轉移所需觸發的轉移點。自動導引車輛系統範例之歐依勒路徑為 S1→S3→S4→S5→S17→S18→S1→S3→S4→S5→S6→S16→S8→S9→S1→S3→S4→S5→S6→S7→S8→S9→S1。

圖 5.7　自動導引車輛範例之規範測試路徑產生

完成一筆畫路徑後，選取生產循環頁面，點選循環分析模組如圖 5.8 所示。在自動導引車輛範例的可達圖，沒有存在相同的生產循環，因此系統將會顯示如圖 5.8，沒有存在相同的生產循環。

圖 5.8　自動導引車輛範例之生產循環分析

若不使用自動化規範測試系統軟體，則自動導引車輛系統的測試法則為全測法，所謂全測法為測試每一條狀態轉換路徑。圖 5.2 首先要找出由初始狀態至各狀態轉換路徑的最短距離如圖 5.2，若可達圖漸趨複雜時，難以直接目視法找出由初始狀態至各狀態轉換的最短距離。此外假設執行規範測試狀態轉移一步需花費一秒，則共須花費 75 秒。自動化規範測試路徑系統軟體可直接由可達圖快速分析出歐依勒路徑，而路徑的產生只需花費一分鐘即可求得。運用歐依勒路徑進行自動導引車輛系統的規範測試，只需花費 22 秒，即可測試完成所有路徑。在此自動導引車輛小型可達圖案例中，規範測試路徑軟體在測試狀態轉移步驟方面大幅節省一半以上的時間((75-22)/75 ≒ 71%)。此外，在找尋最短測試路徑方面，自動化規範測試軟體更大幅的減輕人力搜尋路徑的困難，能快速的產生歐依勒路徑。

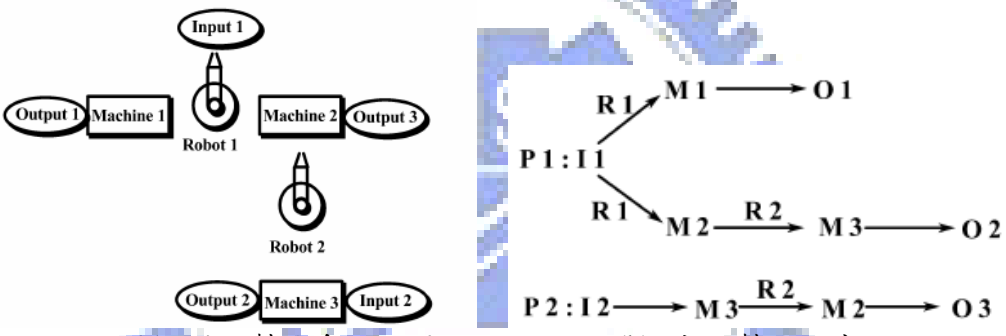
表 5.2　由初始狀態至各狀態轉換路徑

狀態轉換路徑	最短路徑	總路徑
狀態 1→狀態 3	1→3	2
狀態 3→狀態 4	1→3→4	3
狀態 4→狀態 5	1→3→4→5	4
狀態 5→狀態 6	1→3→4→5→6	5
狀態 5→狀態 17	1→3→4→5→17	5
狀態 6→狀態 7	1→3→4→5→6→7	6
狀態 6→狀態 16	1→3→4→5→6→16	6
狀態 7→狀態 8	1→3→4→5→6→7→8	7
狀態 8→狀態 9	1→3→4→5→6→7→8→9	8
狀態 9→狀態 1	1→3→4→5→6→7→8→9→1	9
狀態 16→狀態 8	1→3→4→5→6→16→8	7
狀態 17→狀態 18	1→3→4→5→17→18	6
狀態 18→狀態 1	1→3→4→5→17→18→1	7

自動導引車輛範例產生小型的可達圖，可用規範測試軟體立即找出測試路徑，第 5.2 節將討論在複雜製造系統中，規範測試軟體的應用情況。

5.2 彈性製造系統

第 5.1 節討論的自動導引車輛為小型的可達圖分析，包含未刪除的鎖死狀態一共 18 個狀態。因此本節以 2003 年 Uzam 所提出的論文為例[34]，討論同樣具有鎖死狀態的彈性製造系統，在未經簡化法則簡化前後，運用規範測試路徑軟體的產生路徑的時間。在彈性製造系統內，可以同時生產製造多種產品，因此需要分享系統內的機台、機器手臂、暫存區、設備等等資源。彈性製造系統如圖 5.9(a)，包含兩個機械手臂，三個機器工作檯，機台一一次加工一個元件，機台二、三可一次加工兩個元件，並且同時製造加工兩種產品。圖 5.9(b)為此系統可生產兩種產品的製造程序，產品一有兩種製造方式，其一由 1 號機械手臂從 1 號輸入區(Input 1)拿起原物件後，經由機台(Machine)一加工；其二為 1 號機械手臂由 1 號輸入區拿起原物件後，經由機台二加工，再由 2 號機械手臂將物件由機台二運送至機台三加工。產品二由 2 號輸入區進入機台三加工，再由 2 號機械手臂將物件由機台三運送至機台二加工。兩產品在製造程序中共用機台二、機台三與 2 號機械手臂，當產品一在機台二加工，等待 2 號機械手臂將物件由機台二運送至機台三，而此時產品二在機台三加工，也等待 2 號機械手臂將物件由機台三運送至機台二，此時就會發生兩方機台互相等待對方釋出資源以便加工，但又無法釋出機台資源，會有鎖死問題的產生。



(a) 彈性製造系統設置　　(b) 產品製造程序

圖 5.9　彈性製造系統運作情形

彈性製造系統設置繪成斐氏圖模型以圖 5.10 表示之，其斐氏圖中暫存點與轉移點說明以表 5.3 表示。

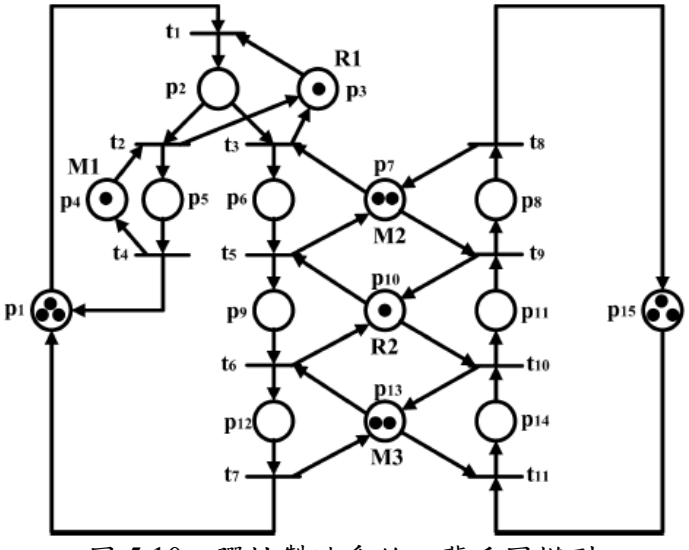


圖 5.10　彈性製造系統之斐氏圖模型

表 5.3　彈性製造系統斐氏圖模型之暫存點與轉移點說明

暫存點意義		轉移點意義	
Place1	1 號輸入輸出區	Transition1	1 號機械手臂夾取原物料 1
Place2	1 號機械手臂運送	Transition2	1 號機械手臂運送物料 1 至機台一，機械手臂空閒，機台一開始加工
Place3	1 號機械手臂空閒	Transition3	1 號機械手臂運送物料 1 至機台二，機械手臂空閒，機台二開始加工
Place4	機台一空閒	Transition4	物料 1 於機台一加工完成
Place5	機台一加工	Transition5	機台二加工完成，2 號機械手臂運送物料 1
Place6	機台二加工	Transition6	2 號機械手臂運送物料 2 至機台二，機台三開始加工
Place7	機台二空閒	Transition7	物料 1 於機台三加工完成
Place8	機台二加工	Transition8	物料 2 於機台二加工完成
Place9	2 號機械手臂運送至機台三	Transition9	2 號機械手臂運送物料 2 至機台二，機台二開始加工
Place10	2 號機械手臂空閒	Transition10	機台三加工完成，2 號機械手臂運送物料 2
Place11	2 號機械手臂運送至機台二	Transition11	機台三開始加工
Place12	機台三空閒		
Place13	機台三加工		
Place14	機台三空閒		
Place15	2 號輸入輸出區		

在斐氏圖基本性質分析中，可將複雜的斐氏圖模型簡化，簡化過後的斐氏圖模型依然保留原斐氏圖的基本性質。以此彈性製造系統為例，可達圖多達 261 個，並存在 29 個鎖死狀態，運用第 2.1.5 簡化法則簡化後的斐氏圖模型如圖 5.11，可大幅減少系統狀態數，縮減可達圖的複雜性。圖 5.12 為簡化後的彈性製造系統之可達圖，系統狀態 26 個，鎖死狀態縮減為 7 個。

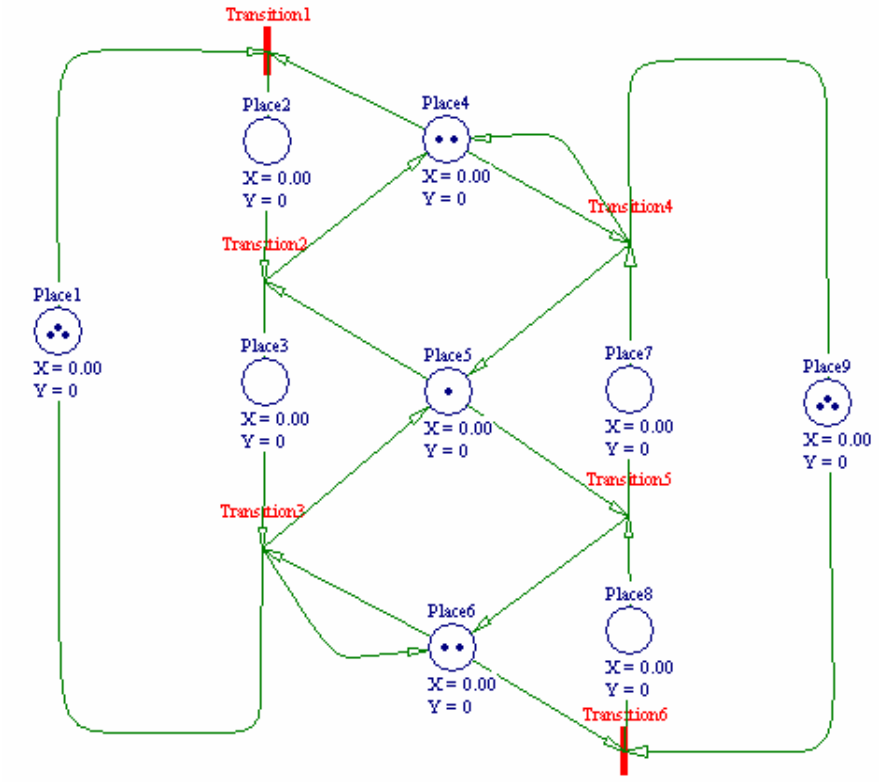


圖 5.11　彈性製造系統之斐氏圖簡化模型

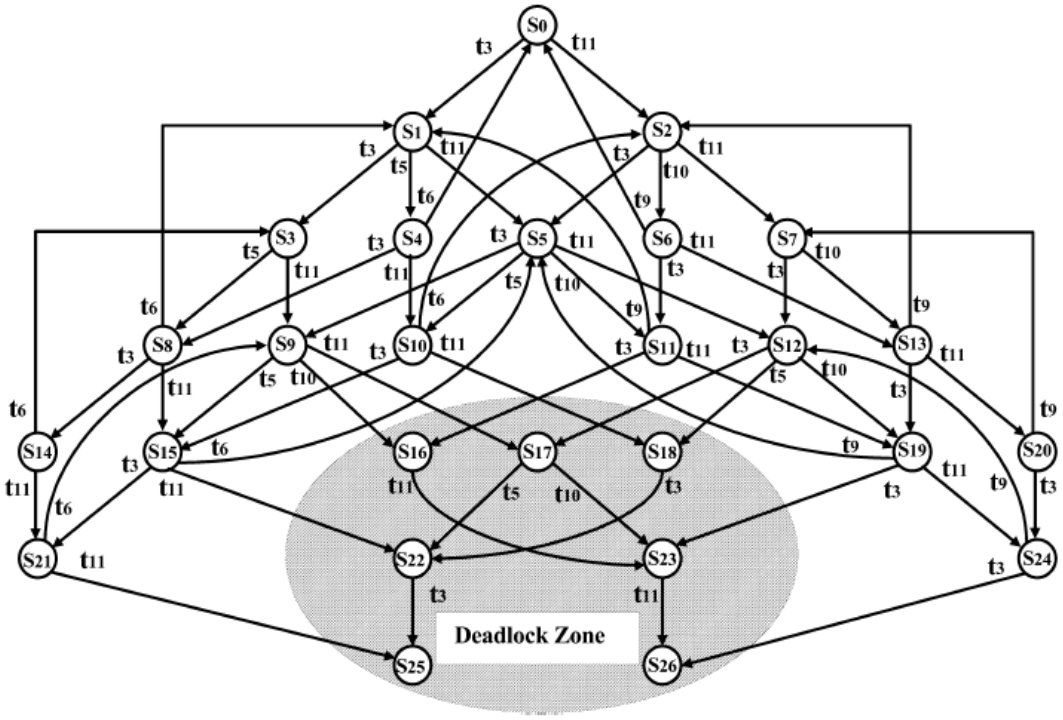


圖 5.12　彈性製造系統斐氏圖簡化模型之可達圖

圖 5.13 為彈性製造系統簡化模型將可達圖 RP.gra 檔案讀入開啟。開啟成功，檔案資料於程式左方顯示 RP.gra 檔案存在與檔案內容。

圖 5.13　載入彈性製造系統簡化模型範例之可達圖檔案

開啟檔案成功，點選活可達圖模組按鈕如圖 5.14，系統列出彈性製造系統簡化模型範例中共 27 個狀態，擁有 7 個鎖死狀態，其中為 S14、S15、S16、S20、S21、S22、S23。

圖 5.14　彈性製造系統簡化模型範例之鎖死狀態分析

活可達圖模組分析完成後，系統自動刪除所有鎖死狀態，點選歐氏有向圖模組按鈕如圖 5.15，進行運輸模式分析，找出所有需要加邊的路徑，使得彈性製造系統簡化模型範例可達圖具有對稱性。系統列出彈性製造系統簡化模型範例中六個供應點為 S6、S7、S8、S17、S18、S19，六個需求點為 S4、S5、S12、S24、S25、S27，以及其路徑成本矩陣。求出最小加邊路徑成本為 46，代表需要增加 46 條路徑，需要多增加的線段為 S6S5 路徑為 S6→S7→S8→S9→S13→S2→S3→S4→S5，以及 S7S27 路徑為 S7→S8→S9→S13→S2→S27，以及 S8S12、S17S12、S18S25、S19S4、S19S24 等總共七組路徑。

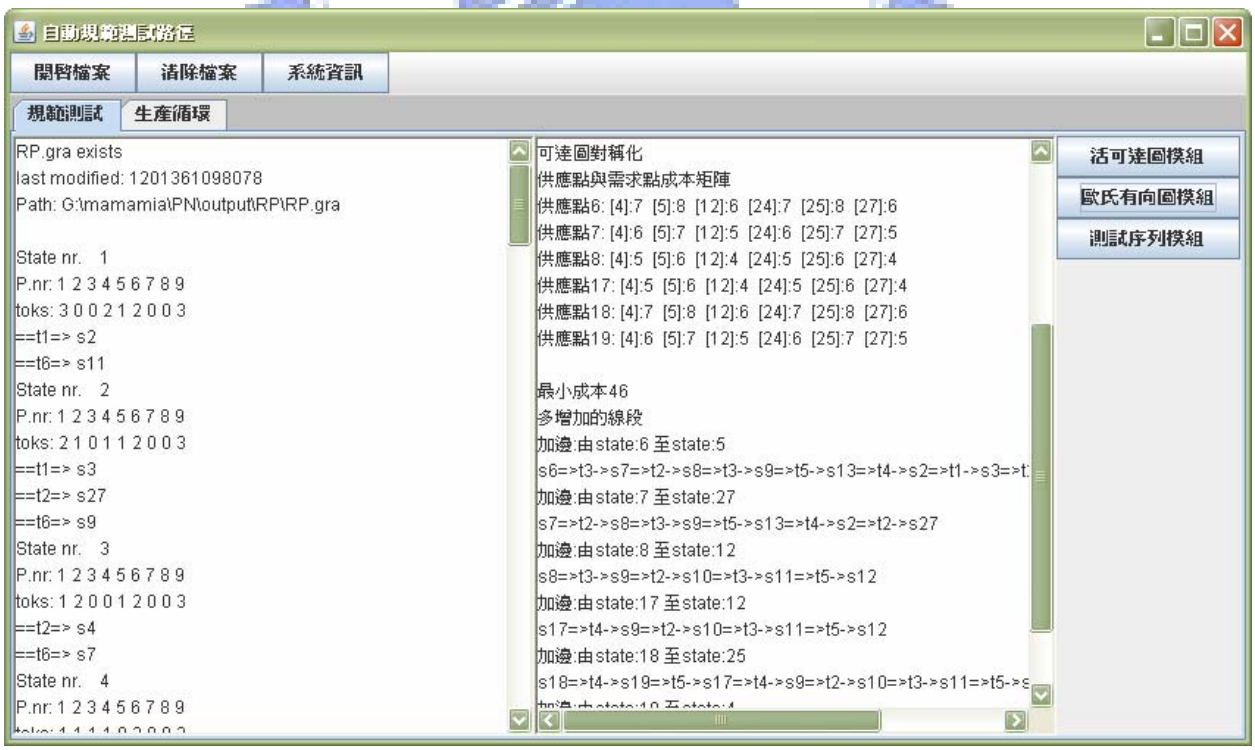


圖 5.15　彈性製造系統簡化模型範例之運輸模式分析

具有對稱性的彈性製造系統簡化模型範例可達圖，即可求得此歐依勒路徑。如圖 5.16 按下測試序列模組按鈕，系統即可顯示彈性製造系統簡化模型範例可行經所有邊界的歐依

勒路徑以及狀態間轉移所需觸發的轉移點。共需行經 90 條路徑。其規範測試歐依勒路徑顯示於圖 5.16。完成一筆畫路徑後，選取生產循環頁面，點選循環分析模組按鈕，如圖 5.17 所示。在彈性製造系統簡化模型範例的可達圖，存在 16 組具有相等效果的生產循環，因此系統將會顯示如圖 5.17，列出所有相同的生產循環。

圖 5.16　彈性製造系統簡化模型範例之規範測試路徑

圖 5.17　彈性製造系統件化模型範例之生產循環分析

第六章 結論與未來發展

6.1 結論

製造系統以斐氏圖模式來描述系統動態運作模型及驗證情形，透過網路監控現場生產線運作情形。爲了確實落實工廠自動化，全面性的整體規劃十分重要。爲了達到全面性整體規劃，自動化製造系統的設計與實作必須歷經五大步驟：規格、驗證、撰碼、測試、偵查。近年來，由於斐氏圖加註語言的規範，開發了許多有價值的斐氏圖套裝軟體，然而這些套裝軟體，只針對斐氏圖進行相關的分析。對於斐氏圖所產生的狀態可達圖而言，也具備著相當多有價值的資訊分析，然而對於可達圖領域的套裝軟體，卻依然缺乏。

在已建立控制軟體的斐氏圖模式及開發出斐氏圖軟體的前提下，本研究針對測試階段來設計與實作其自動化規範測試系統。更詳細地說，規範測試是比較設計行為的可達圖和實作行為的可達圖是否等效，如圖 6.1 所示。如果實作出來的可達圖有發揮設計上可達圖的效果，則稱撰碼階段裡開發出來的控制軟體符合驗證階段裡斐氏圖模式的設計要求。

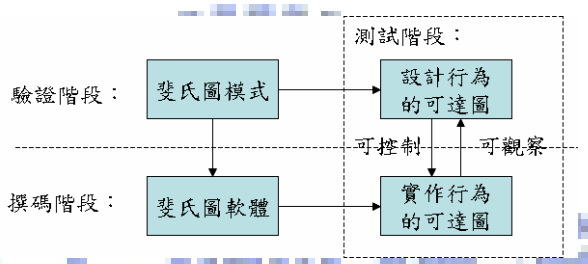


圖 6.1 規範測試

因此，本研究運用爪哇程式語言提供一個通用且快速的規範測試軟體，藉由結合斐氏圖套裝軟體自動產生狀態可達圖的特點，分析系統狀態可達圖。由於初始可達圖圖形並非具有歐依勒一筆畫路徑的特性，故採用中國郵差問題的運輸模式計算，得到最短需要重複行經的路徑，利用加邊方式將可達圖對稱化，以達到具備有歐依勒路徑的可達圖圖形。接著運用 Fleury 演算法，進而自動化產生所有狀態轉換的最短規範測試路徑，即可快速透過測試路徑測試監控程式的撰碼準確性。並且藉由最短路徑測試，快速縮減規範測試路徑時間，並能套用於所有系統產生的狀態可達圖分析，使用者不需因為專案的改變，而重新計算其規範測試路徑。更加提供生產循環分析，利用圖論的符號矩陣，找出等效的生產循環，管理者可根據具有相同製程能力效果的生產循環，找出耗費最少時間的加工方式，藉此得到最佳的派工法則。

例如第 5.1 節討論的自動導引車輛例子。若不使用規範測試路徑，則測試法則為全測法，所謂全測法為測試每一條由初始狀態至各狀態的最短路徑。需以目視法找出由初始狀態至各狀態的最短距離。使用目視法找到的自動導引車輛的全測步驟需要花費 75 步。此外，尚不包含使用人力目視法找到由初始狀態至各狀態的最短路徑。運用自動化規範測試軟體進行自動導引車輛系統的規範測試，可直接快速的分析出最短的歐依勒一筆畫路徑，產生出來的一筆畫路徑的測試步驟只有 22 步，即可測試完系統間所有的狀態轉換，在測試步驟方面，只需花費原本的 29%(22 / 75)的時間。並能立即分析出系統中是否含有相同的生產循環。自動化規範測試軟體更大幅的減輕人力搜尋路徑的困難，能快速的產生歐依勒一筆畫路徑，對於複雜系統而言，其效益更是難以估算。

本研究尚未考慮狀態間轉換耗費時間不同的因素，並採用獨立系統作業，完成自動化製造系統設計與實作五大步驟中，測試的自動化。但並未與撰碼及驗證的自動化系統進行整合。若能將三大步驟的自動化系統整合，將有助於使用者更方便快速的進行自動化製造系統設計與實作。

6.2 未來發展

未來研究方向可加入狀態間轉換需要耗費的時間，轉換時間包含機台加工時間，機台間搬運時間，組裝時間等。更進一步說明，藉由中國郵差問題的運輸模式將狀態可達圖圖形進行加邊時，時間參數等同於狀態間轉換所需要耗費的路徑成本，考慮時間參數進行最短加邊路徑計算時，將有助於找出最佳的重複行經路徑，更能有效的縮短規範測試最短路徑的搜尋。

其次，可針對五大步驟中的規格及偵察部分完成自動化軟體設計。對於規格部分，可將系統利用 IDEF0 定義規格後，直接轉成斐氏圖模式；對於偵查部分，則可套用擬陣理論自動化產生偵察法則的通用程式。往後，管理者可透過套裝軟體通用程式完成自動化製造系統的設計與實作中規格、驗證、撰碼、測試與監控的五大步驟自動化。藉由自動化規格軟體，定義製造系統的規格並直接匯出斐氏圖模式；接著透過斐氏圖軟體其斐氏圖模式進行系統驗證；連接自動化撰碼系統，將驗證過的斐氏圖模式直接撰碼為監控程式；透過本研究提出的自動化規範測試軟體，快速的測試監控程式與實際機台的狀態轉換；最後，藉由自動化偵查軟體提供的偵察法則，套入監控程式中，立即監控製造的運作情形，避免發生任何當機現象。假以時日，更能將五大步驟的自動化套裝軟體整合，使得製造系統的全面性整體規劃所需的五大步驟，只需透過一個套裝軟體，便能快速提升自動化製造系統的設計與實作的執行效率。

參考文獻

[1] 陳錫川、梁高榮，「機械手臂製造單元的遠端控制設計與實作」，中華民國工業工程學年會論文集第一冊。455-461 頁，1995。

[2] 梁高榮，「自動化製造系統內的即時故障察覺」，機械工業 278 期，170-179 頁，2006。

[3] 梁高榮，「虛體製造系統內的多緒架構」，機械工業。241-261 頁，2003。

[4] 勞虎，無廢話 XML，1991 年 10 月。

[5] 陳音帆、梁高榮，「透過斐氏圖加註語言分析自動化製造系統行為」，機械工業 282 期，170-181 頁，2006 年 9 月。

[6] 陳音帆、梁高榮，「自動化製造系統裡斐氏圖模式的撰碼自動化」，機械工業 285 期，170-180 頁，2006 年 12 月。

[7] 陳音帆，「斐氏圖導向控制器開發系統的設計與實作」，國立交通大學工業工程與管理研究所碩士論文，2008。

[8] 曹漢清，「爪哇豆技術建構虛體製造系統及其規範測試」，機械工業 243 期，246-255 頁，2003。

[9] 曹漢清，「利用爪哇訊息服務設計與實作供應鏈執行系統」，國立交通大學工業工程與管理研究所碩士論文，2002。

[10] 林辰戴、梁高榮，「自動化製造系統互動組件的非鎖死設計」，中華民國工業工程學年會論文集第一冊。455-461 頁，1995。

[11] 王中行、劉大銘、趙柏鴻，「斐氏網路圖為基礎之彈性製造系統建模與效率分析」，大業學報，10(2)，2001。

[12] http://xml.org.tw/。XML 台灣資訊網。2007 年 10 月 25 日。

[13] http://xserve.math.nctu.edu.tw/people/cpai/carnival/bridge/index.htm。七橋問題。2007 年 10 月 25 日。

[14] http://www.informatik.uni-hamburg.de/TGI/PetriNets/ Petri Nets World. October 25, 2007.

[15] http://www.lindo.com/. LINDO SYSTEMS INC. June 10, 2008.

[16] http://www.uml.org/. OMG. June 10, 2008.

[17] Boggey, T. B.,Graph Theory in Operations Research,1982.

[18] Bosik, B. S. and Uyer, M. U., “State Machine Based Formal Methods in Protocol Conference Testing: from Theory to Implementation,” Computer Networks and ISDN Systems, Vol. 22, pp. 7-33, 1991.

[19] Chartrand, G. and Oellermann, O. R., Applied and Algorithmic Graph Theory, McGraw-Hill, 1993.

[20] Eiselt, H. A., “Arc Routing Problems, Part I: The Chinese Postman Problem,” Operations Reasearch, Vol. 43, No. 2, pp. 231-242, 1995.

[21] Feng, L. J., “A Comparison of Matroid-Theoretical Monitor Design Methods,” Thesis, Department of Industrial Engineering and Management, National Chiao Tung University, 2001.

[22] Fleischner, H., Eulerian Graphs and Related Topics, Part1, Vol. 2, North-Holland, 1991.

[23] Hillier, F. S. and Lieberman, G. J., Introduction To Operation Research 7e, 1974.

[24] Hsueh, F. C., “Conformance Testing of Interactive Compact Disc Titles Using Chinese Postman Algorithm,” Thesis. Department of Industrial Engineering and Management, National Chiao Tung University, 1999.

[25] Lee, H. Y. and Lee, T. E., “Scheduling Single-Armed Cluster Tools With Reentrant Wafer Flows,” IEEE, Vol. 19, No. 2, pp. 226-240, 2006.

[26] Liang, G. R., “Network-integrated Production Lines,” Journal of the Mechatronic Industry, Vol. 172, pp. 207-220, 1997.

[27] Lin, Y. and Zhai, Y., “A New Algorithm for The Directed Chinese Postman Problem,”

Computer Operations Research, Vol. 15, No. 6, pp. 577-584, 1988.
[28] Murata, T., “Petri Nets: Properties, Analysis and Applications,” Proceedings of the IEEE, Vol. 77, No. 4, pp. 541-580, 1989.
[29] Pearn, W. L. and Li, M. L., “Algorithm for The Windy Postman Problem,” Computer Operations Research, Vol. 21, No. 6, pp. 641-651, 1994.
[30] Pearn, W. L. and Wu, T. C., “Algorithm for The Rural Postman Problem,” Computer Operations Research, Vol. 22, No. 8, pp. 819-828, 1995.
[31] Pearn, W. L., “Approximate Solutions For The Capacitated Arc Routing Problem,” Computer Operations Research, Vol. 16, No. 6, pp. 589-600, 1989.
[32] Pearn, W. L. and Wang, K. H., “On the Maximum Benefit Chinese Postman Problem,” Omega, Vol. 31, pp. 269-273, 2003.
[33] Ross, D. T., “Applications and Extensions of SADT,” Computer, pp. 25-34, 1985.
[34] Uzam, M., “The use of the Petri net reduction approach for an optimal deadlock prevention policy for flexible manufacturing systems,” International Journal of Advanced Manufacturing Technology, Vol. 23, pp. 204-219, 2004.
[35] Viswanadham, N., Narahari, Y., and Johnson, T. L., “Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models,” IEEE Trans. Robots Automat., Vol. 6, No. 6, 1990.

附註 1　規範測試通用程式碼

規範測試通用程式共分為七個子程式，分別為：1. 讀取可達圖檔案、2. 成本計算、3. 鎖死分析、4. 運輸模式計算、5. Fleury 演算法計算、6. GUI 介面設計、7. 生產循環分析計算。以下將分別說明。

1.　讀取可達圖檔案

```
private void openFile(){
//成本初始值
   for(int y=0; y < cost.length;y++){
          for(int z=0; z <cost.length;z++){
             cost[y][z]=1000;
          }
       }
 //開啟檔案
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        int result = fileChooser.showOpenDialog(this);

        if (result == JFileChooser.CANCEL_OPTION)
           return;

        File fileName = fileChooser.getSelectedFile();

        if (fileName.exists()){
        outputArea.setText(fileName.getName() + " exists" +
                    /*    (fileName.isFile()? "is a file\n" : "is not a file\n") +
                        (fileName.isAbsolute()? "is a path":"is not a path") +*/
                        "\nlast modified: " + fileName.lastModified() +
                        "\nPath: " + fileName.getPath());

              if(fileName.isFile()){
                //open File
                //append contents of file to outputArea
                  try {
                       BufferedReader input = new BufferedReader(new FileReader (fileName));
                       buffer =new StringBuffer();
                       String text;          //作為 input.readline 的存檔
                       outputArea.append("\n");

        //button display
                       openbutton.setEnabled(true);
                       fleurybutton.setEnabled(false);
                       deadlockzone.setEnabled(true);
                       cpa.setEnabled(false);

                      String a,g;
                      int c,t;       //count 紀錄 state 的狀態
                       while((text = input.readLine())!=null){
                        // output state read
                        if(text.indexOf("State nr.")!=-1){
                            count++;
                            outputarc[count]=0;
                        }
                        //input state read
                        if(text.indexOf("=>")!=-1){
                                  a = text.substring(text.indexOf(">")+3);
```

```
                    g = text.substring(3,text.indexOf(">")-1);    //transition

                    c = Integer.parseInt(a);              //State to Int
                    t = Integer.parseInt(g);              //transition to Int
                    transition[count][c] =    t;          //place transition  儲存
                    map[count][c]=("=>t"+g+"->s"+a);
                    cost[count][c]=1;
                    euler1[count][c]=1;
                }

                buffer.append(text + "\n");

            }
             outputArea.append(buffer.toString());
        }


          //process file processing problems
          catch(IOException ioException){
            JOptionPane.showMessageDialog(this,"FILEE    RROR","FILE    ERROR",    JOptionPane.
ERROR_MESSAGE);
          }
        }//end if
        //output directory listing

    } //end outer if
    //not file or directory,output error message

    //display error if invalid
   // if(filename ==null||filename.getName().equals(" "))
  else    {
    JOptionPane.showMessageDialog(this,"Invalid    File    Name","Invalid    File    Name",    JOptionPane.
ERROR_MESSAGE) ;
   }
  }
```

2. 成本計算

```
private void costTest(){
    System.out.println(count);
    int sum=0;                                          //紀錄是否替換 cost
    computeArea.setText("各路徑成本\n");

     //成本預設值 1000   計算各路徑成本
  for(int g=0;g <(count+20)/10;g++){
    for (int i = 1; i <= count; i++) {
       for (int j = 1; j <= count; j++) {
          if (cost[i][j] < 1000) {
             for (int k = 1; k <= count; k++) {
                if (cost[j][k] < 1000) {
                   sum = cost[i][j] + cost[j][k];
                   if (cost[i][k] > sum){
                      cost[i][k] = sum;
                      map[i][k]=(map[i][j]+map[j][k]);      //紀錄行經路徑                    }
                }
             }
          }
       }
    }
  }
    System.out.print("ok");
}
```

3. 鎖死分析

```
//display and analize deadlock
   private void deadlockTest(){


      int a=0,b=0;
      computeArea.setText("鎖死區域(Deadlock Zone):\n");

      for(int i=1;i<=count; i++){
         for(int j=1;j<=count;j++){
            if (cost[i][j]==1000) {
               b=1;          //deadlock state =1
             }
         }
         if(b!=0){
         computeArea.append("\nState: " +i);      //印出 Deadlock 區域
         a++;
         b=0;
            for(int k=1 ;k<=count;k++){                    //刪除 Deadlock 的狀態
               transition[i][k] = 0;
               transition[k][i] = 0;
                 }
         }
         }
   //紀錄各狀態進入與出發線段
      for(int i=1;i<=count;i++){
            for(int j=1;j<=count;j++){
               if(transition[i][j]!=0&&cost[i][j]!=1000){        //刪除鎖死路徑
                  inputarc[j]++;
                  outputarc[i]++;
               }
           }
          }
   //display deadlock zone
      computeArea.append("\n 所有狀態(Total State):"+count+"\n 鎖死狀態(Deadlock State):" +a);
   }
```

4. 運輸模式計算

```
private void cpaTest(){
    int supply[]=new int[count+1];                    //紀錄供應量
    int demend[]=new int[count+1];                    //紀錄需求量
    int supplypoint[]=new int[count+1];               //紀錄供應狀態
    int demendpoint[]=new int[count+1];               //紀錄需求狀態
    int costcal[][]=new int[count+1][count+1];        //供需成本表
    int road[][]=new int[count+1][count+1];           //供需成本表的節點
    int rowresidual[] =new int [count+1];             //最佳化列剩餘值
    int columnresidual[] =new int [count+1];          //最佳化列剩餘值
    int a=0,b=0;
    int seti=0,setj=0;

     computeArea.setText("供應點與需求點成本矩陣\n");
      //find out 供應點與需求點
        for(int i=1;i<=count;i++){
        if(inputarc[i]<outputarc[i]){
           a++;
           demendpoint[a]=i;
           demend[a] = outputarc[i]-inputarc[i];
        }
        else if(inputarc[i]>outputarc[i]){
                b++;
                supplypoint[b]=i;
                supply[b]=inputarc[i]-outputarc[i];
        }
      }
      //把 cost 把 cost 另存一個檔案
      for(int i=1;i<=b;i++){
         computeArea.append("供應點"+supplypoint[i]+": ");
         for(int j=1;j<=a;j++){
            costcal[i][j] = cost[supplypoint[i]][demendpoint[j]];
            computeArea.append("["+demendpoint[j]+"]:"+costcal[i][j]+"    ");
         }
         computeArea.append("\n");
      }
      int remaining[][]=new int[b+1][a+1];
      int u=count,costmin=count;
      int mincost=0;

//Russell 近似法
//找出單行或列最大值
          int maxrow[]=new int[b+1];     //列
          int maxcolumn[]=new int[a+1];//行
          int row=0,column=0;
//行最大值
for(int y=0;y<=a+b;y++){
       for(int i=1;i<=b;i++){
          for(int j=1;j<=a;j++){
                if(costcal[i][j]>row){
                   row=costcal[i][j];
                }
            }
          maxrow[i]=row;
          row=0;
        }

//列最大值
       for(int i=1;i<=a;i++){
```

```
    for(int j=1;j<=b;j++){
       if(costcal[j][i]>column){
          column=costcal[j][i];
       }
    }
    maxcolumn[i]=column;
     column=0;
}

  u=count;
 for(int i=1;i<=b;i++){
    for(int j=1;j<=a;j++){
     if(costcal[i][j]!=0){
      remaining[i][j]=costcal[i][j]-maxrow[i]-maxcolumn[j];

       if(remaining[i][j]<u){                  //判斷最小值&&costcal[i][j]<costmin
          u = remaining[i][j];
       //costmin = costcal[i][j];
       }
     }
    }
 }


 //choose road    and min cost
costmin=count;
 for(int i=1;i<=b;i++){
    for (int j=1; j<= a; j++) {
     if(costcal[i][j]!=0){
       if(remaining[i][j]==u&&costcal[i][j]<costmin){
                  costmin = costcal[i][j];
                  seti=i;
                  setj=j;
       }
    }
    }
 }

 //setting road
 if(demend[setj]>supply[seti]&&supply[seti]!=0){
    road[seti][setj]=supply[seti];                              //路=供應量
    columnresidual[setj]=0;
    rowresidual[seti]=costcal[seti][setj];
    demend[setj] = demend[setj]-supply[seti];
    mincost=mincost+(supply[seti]*costcal[seti][setj]);
    supply[seti]=0;
    for(int i=1;i<=a;i++){
       costcal[seti][i]=0;
    }
    }
 else if(demend[setj]<supply[seti]&&demend[setj]!=0){
    road[seti][setj] = demend[setj];
    columnresidual[setj] = 0;
    rowresidual[seti] = costcal[seti][setj];
    supply[seti]=supply[seti]-demend[setj];
    mincost=mincost+(demend[setj]*costcal[seti][setj]);
    demend[setj]=0;
    for(int i=1;i<=b;i++){
     costcal[i][setj]=0;
  }
 }
 else if(demend[setj]==supply[seti]&&demend[setj]!=0&&supply[seti]!=0){
    if(rowresidual[seti]>0||columnresidual[setj]>0){
```

```
          columnresidual[setj]=costcal[seti][setj]-rowresidual[seti];
          }
          else{
             rowresidual[seti]=costcal[seti][setj];
             columnresidual[setj]=costcal[seti][setj]-rowresidual[seti];
          }
          road[seti][setj] = demend[setj];
          mincost=mincost+(supply[seti]*costcal[seti][setj]);
          supply[seti]=0;
          demend[setj]=0;
          for(int i=1;i<=a;i++){
            costcal[seti][i]=0;
          }
          for(int i=1;i<=b;i++){
          costcal[i][setj]=0;
       }
       }
    }
//印出 road
  //哪條的值最多
    int reach=0,start;
    String state,bb;
    computeArea.append("\n 最小成本"+mincost+"\n");
    computeArea.append("\n 多增加的線段");

    for(int i=1;i<=b;i++){

       for(int j=1;j<=a;j++){
          if(road[i][j]!=0){
             bb=map[supplypoint[i]][demendpoint[j]].substring(map[supplypoint[i]][demendpoint[j]].indexOf("s"));
             start = supplypoint[i];
        for(int y=0 ; y < cost[supplypoint[i]][demendpoint[j]]-1;y++){    //
             state = bb.substring(bb.indexOf("s") + 1, bb.indexOf("="));
             reach = Integer.parseInt(state);
             euler1[start][reach]=euler1[start][reach]+road[i][j];
             start=reach;
          if(bb.length()>=7){
            bb = bb.substring(bb.indexOf("=")+1);
          }
       }
            euler1[reach][demendpoint[j]]=euler1[reach][demendpoint[j]]+road[i][j];

            computeArea.append("\n 加邊:由 state:"+supplypoint[i]+"  至 state:"+demendpoint[j]);
            computeArea.append("\ns"+supplypoint[i]+map[supplypoint[i]][demendpoint[j]]);
          }
          //System.out.print("\nroad["+x+"]["+v+"]:" + road[x][v]);
       }
       }
       int state1=0;
       for(int i=1;i<=count;i++){
     for(int j=1;j<=count;j++){
          if(euler1[i][j]>0&&cost[j][1]!=1000){          //若有路徑和成本小於 1000
             state1=state1+euler1[i][j];                 //計算 Total path
          }
     }
}
    System.out.print("\nstate1"+state1);

}
```

5. Fleury 演算法計算

```
private void fleury(){

  int euler[][]=new int [count+1][count+1];      //計算用的路徑
  int costF[][]=new int [count+1][count+1];      //重新紀錄的成本矩陣
  int costG[][]=new int[count+1][count+1];       //計算用的成本矩陣
  int not=0;              //紀錄不能走的路     下一個迴圈的初始點
  int state=0;            //紀錄 Total 路徑數
  int passpath=0;       //紀錄已走過幾條路

 computeArea.setText("路徑");
 for(int i=1;i<=count;i++){
     for(int j=1;j<=count;j++){
        costF[i][j]=1000;                          //預設成本為 1000
        euler[i][j]=0;                             //預設無路徑
        if(euler1[i][j]>0&&cost[j][1]!=1000){      //若有路徑和成本小於 1000
           costF[i][j]=1;                          //路徑成本設為 1
           euler[i][j]=euler1[i][j];               //路徑存再另外一條路
           state=state+euler[i][j];                //計算 Total path
        }
     }
}
  computeArea.append(":"+state+"\n");

  int pointi=1,pointj=0,sumF,jumploop=0;
  String path=("1");
for(int x=0;x<state+passpath;x++){                 //重複次數=總路線加上不能走的次數
     for(int j=not+1;j<=count;j++){
      if(euler[pointi][j]>0&&jumploop==0){
        pointj=j;          //未來狀態設為 pointj
        jumploop=1;
     }
  }
  jumploop=0;
  not=0;                                           //不能走的路歸零
        euler[pointi][pointj]--;                   //刪除此路徑
        if(euler[pointi][pointj]==0){              //此路徑若無路 則成本設為 1000
           costF[pointi][pointj] = 1000;
        }
     for (int i = 1; i <= count; i++) {
      for (int j = 1; j <= count; j++) {
        costG[i][j] = 1000;
        if(costF[i][j]<1000){                      //有路的才將路徑成本存再另一個陣列 costG
           costG[i][j]=costF[i][j];
         }
       }
       }

     //計算路徑成本   每個狀態都要能回到 1 表示路可走
  for(int g=0;g <=(count+20)/10;g++){
   for (int i = 1; i <= count; i++) {
      for (int j = 1; j <= count; j++) {
        if (costG[i][j] < 1000) {
           for (int k = 1; k <= count; k++) {
              if (costG[j][k] < 1000) {
                 sumF = costG[i][j] + costG[j][k];
                 if (costG[i][k] > sumF){
                    costG[i][k] = sumF;
```

```
                    }
                }
            }
        }
    }
}

int routeback=0;
for(int i=2;i<=count;i++){
    for(int j=1;j<=count;j++){
    if(costG[i][1]==1000&&i!=j&&euler[i][j]>0){
        routeback=1;
    }
    }
}
//若有狀態不能回到初始狀態   routeback=1
    if(routeback==0){
        path=(path+"->"+String.valueOf(pointj));
        computeArea.append("\nstate"+pointi+" -> t"+transition[pointi][pointj]+" -> state"+pointj);
            if(euler[pointi][pointj]==0){
                costF[pointi][pointj] = 1000;
            }
            pointi=pointj;
            System.out.print("\npath"+path);
    }
    else if(routeback==1){
        euler[pointi][pointj]++;
        costF[pointi][pointj]=1;
        not=pointj;
         passpath++;
    }
  }
}
```

6. GUI 介面

```
package GUI;
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import java.util.Set;


//規範測試路徑變數設定
public class GUI extends JFrame {

//   宣告陣列變數的大小
private int amount = 300;
   //規範測試路徑變數設定
private GridBagLayout layout;                          //版面管理員
private GridBagConstraints constraints;                //版面限制
private JPanel panel1, panel2; //tabbed1's panel
private JMenuBar bar; //menu bar
private JTextArea outputArea, outputArea1, outputArea3;                    //outputArea in JPanel One
private int count=0;                                   //計算有幾個 state
private int cost[][] = new int[amount][amount];        //成本變數儲存
private int inputarc[]=new int[amount];                //進去此狀態次數
private int outputarc[]=new int [amount];              //由狀態出去次數
private int transition[][]=new int[amount][amount];    //轉移點變數儲存
private String map[][]=new String[amount][amount];     //記錄需要加邊的路徑
private int euler1[][]=new int[amount][amount];        //有幾個路徑  eulerplus[start][reach]
//private String lifecycle[][]=new String[300][300];   //有幾個路徑  eulerplus[start][reach]
private StringBuffer buffer , buffer1;                 //讀取資料檔案字串

//生產循環變數設定
private int availablestate;                              //可用狀態數
private int machinetime[]=new int[amount];               //儲存機台運作時間
private String manufacturCycle[] = new String[amount];   //儲存生產循環
private int cyclepoint = 0;                               //第幾次循環有生產週期現象

   public GUI() {
      //Frame Title
      super("自動規範測試路徑");

      // set up File menu and its menu items
      JMenu fileMenu = new JMenu("檔案");
   //    fileMenu.setMnemonic('F');

      // set up Open File
      JMenuItem openFile = new JMenuItem("開啟檔案");
     // openFile.setMnemonic('o');
      fileMenu.add(openFile);
      openFile.addActionListener(
         new ActionListener() {    //anonymous inner class
            // Open gra File
            public void actionPerformed(ActionEvent event)
            {
               openFile();         //執行開啟檔案
               costTest();         //執行路徑成本計算
            }
         } // anonyomous inner class
```

```
      );//end actionlistener

    JMenuItem clearFile = new JMenuItem("清除檔案");
   // clearFile.setMnemonic('C');
    fileMenu.add(clearFile);
    clearFile.addActionListener(
     new ActionListener() {    //anonymous inner class
        // Open gra File
        public void actionPerformed(ActionEvent event)
        {
           exitFile();         //執行清除所有記錄
        }
     } // anonyomous inner class
     );//end actionlistener



    //set up About menuItem
    JMenu aboutItem = new JMenu("關於...");
//    aboutItem.setMnemonic('A');

    //set up Information
    JMenuItem informationItem = new JMenuItem("系統資訊");
    informationItem.setMnemonic('I');
    aboutItem.add(informationItem);
    informationItem.addActionListener(
       new ActionListener() {    //anonymous inner class
          //display message dialog when user selects About...
          public void actionPerformed(ActionEvent event)
          {
             JOptionPane.showMessageDialog(null,
                                           "此軟體為\n 自動規範測試系統",
                                           "About",JOptionPane.PLAIN_MESSAGE);
          }
       } // anonyomous inner class
       );//end actionlistener

    //set up Exit menu item
    JMenuItem exitItem = new JMenuItem("Exit");
    exitItem.setMnemonic('x');
    fileMenu.add(exitItem);
    exitItem.addActionListener(
       new ActionListener(){ //anonymous inner class
         //teminate application when user click exitItem
         public void actionPerformed(ActionEvent event)
         {
            System.exit(0);
         }
       } //end anonymous inner class
       ); //end actionlistener

       //create JTabbedPane
       JTabbedPane tabbedPane = new JTabbedPane();

       //set up panel1 and add it to JTabbedPane
      // JLabel label1 = new JLabel("panel1 one");
       panel1 = new JPanel();
       tabbedPane.addTab("規範測試",null, panel1,"First Panel");

       //set up panel2 and add it to JTabbedPane
       JLabel label2 = new JLabel("panel2 two");
       panel2 = new JPanel();
```

```
panel2.add(label2);
tabbedPane.addTab("生產循環",null, panel2,"Second Panel");

//*規範測試 Panel1*
//Panel One Layout Setting
layout = new GridBagLayout();
panel1.setLayout(layout);

//intantiate gridbag constriants
constraints = new GridBagConstraints();

//Button in Panel One to produce shoutest path
JButton deadlockzone =new JButton("鎖死分析");
JButton cpa =new JButton("可達圖對稱化");
JButton fleurybutton =new JButton("一筆劃路徑");
JPanel nothing = new JPanel();

//deadlock button action setting
deadlockzone.addActionListener(
new ActionListener() {    //anonymous inner class
   // Open gra File
   public void actionPerformed(ActionEvent event)
   {
      //加開啟檔案的程式
      deadlockTest();        //鎖死分析
   }
} // anonyomous inner class
);//end actionlistener

//cpa button action setting
cpa.addActionListener(
new ActionListener() {    //anonymous inner class
   // Open gra File
   public void actionPerformed(ActionEvent event)
   {
      //加開啟檔案的程式
      cpaTest();        //鎖死分析
   }
} // anonyomous inner class
);//end actionlistener

//fleurybutton button action setting
fleurybutton.addActionListener(
new ActionListener() {    //anonymous inner class
   // Open gra File
   public void actionPerformed(ActionEvent event)
   {
      //加開啟檔案的程式
      fleury();        //鎖死分析
   }
} // anonyomous inner class
);//end actionlistener


//panel add Scroll
outputArea = new JTextArea();
outputArea1 = new JTextArea();
outputArea.setText("檔案開啟:");
outputArea1.setText("結論:");
//scroll setting output zone
ScrollPane scrollPane = new ScrollPane();
```

```
  ScrollPane scrollPane1 = new ScrollPane();
  scrollPane.add(outputArea);
  scrollPane1.add(outputArea1);
// scrollPane1.add(outpu)

  // grid Layout
  constraints.weightx = 1000;
  constraints.weighty = 1000;
  constraints.fill = GridBagConstraints.BOTH;
  addComponent(scrollPane, 0, 0, 1, 4);
  addComponent(scrollPane1, 0, 1, 1, 4);
//    addComponent(outputArea, 0, 0, 1, 4);

  constraints.weightx = 0;
  constraints.weighty = 0;
  addComponent(deadlockzone, 0, 2, 1, 1);
  addComponent(cpa, 1, 2, 1, 1);
  addComponent(fleurybutton, 2, 2, 1, 1);
  addComponent(nothing, 3, 2, 1, 1);
  //*規範測試 Window end*

  //*生產循環 Window*
    //Panel One Layout Setting
  panel2.setLayout(layout);

  //Button in Panel One to produce shoutest path
  JButton cyclebutton =new JButton("循環分析");
  JButton opentimebutton =new JButton("讀入操作時間");
  JPanel nothing1 = new JPanel();

  //panel2 add Scroll
  outputArea3 = new JTextArea();
  outputArea3.setText("生產循環分析");

  //scroll setting output zone
  ScrollPane scrollPane3 = new ScrollPane();
  scrollPane3.add(outputArea3);

  // grid Layout
  constraints.weightx = 1000;
  constraints.weighty = 1000;
  constraints.fill = GridBagConstraints.BOTH;
  addComponent1(scrollPane3, 0, 0, 1, 4);

  constraints.weightx = 0;
  constraints.weighty = 0;
  addComponent1(cyclebutton, 0, 1, 1, 1);
// addComponent1(opentimebutton, 1, 1, 1, 1);
  addComponent1(nothing1, 2, 1, 1, 3);

  cyclebutton.addActionListener(
  new ActionListener() {    //anonymous inner class
     // Open gra File
     public void actionPerformed(ActionEvent event)
     {
        //生產循環的程式
           cycle();        //鎖死分析
       //     producecycle();
     }
  } // anonyomous inner class
  );//end actionlistener
```

```
        //*生產循環 Window end*


        //*Main window* create Menu bar and attach it to MenuTest Window
        bar = new JMenuBar();
        setJMenuBar( bar );
        bar.add(fileMenu);
        bar.add(aboutItem);

        //*Main window* contentPane, add JTabbedPane to container
        getContentPane().add(tabbedPane);
        getContentPane().setBackground(Color.LIGHT_GRAY);
        setSize(500,400);
        setVisible(true);
    }
    private void addComponent (Component component, int row, int column, int width, int height)
    {
      //set gridX and gridY
      constraints.gridx = column;
      constraints.gridy = row;

      //set gridwidth and gridheight
      constraints.gridwidth = width;
      constraints.gridheight = height;

      //set constraints and add component
      layout.setConstraints(component , constraints);
      panel1.add( component );
    }

    private void addComponent1 (Component component, int row, int column, int width, int height)
{
    //set gridX and gridY
    constraints.gridx = column;
    constraints.gridy = row;

    //set gridwidth and gridheight
    constraints.gridwidth = width;
    constraints.gridheight = height;

    //set constraints and add component
    layout.setConstraints(component , constraints);
    panel2.add( component );
}
```

7. 生產循環分析計算

```
private void cycle() {
  int k=1, a=1 , b=1,c;
  int arraysize = 5000;
  String cyclestring;                          //暫存初始狀態轉換資訊為字串
  int firstword = 0 ;                          //起始狀態
  String cycle[] = new String[arraysize];      //展開樹陣列
  String cycleshow[] = new String[arraysize];  //回到原始狀態的展開樹字串存至此
  int countlist[] = new int[arraysize];        //狀態轉移次數
  int countlist2[] = new int[arraysize];       //回到原始狀態的狀態轉移次數
  int cyclenumber[] = new int [arraysize];     //具有相同轉移次數的狀態數

  //display how much available states
  //System.out.println("\navailablestate" + availablestate);
  //initial Value (could add to Deadlock())
    for(int    i=1 ; i <= count; i++){
       for(int j=1 ; j <= count ; j++){
          if(euler1[i][j] > 0 ){
             cyclestring = (i+"]["+ j);
             countlist[k] = 2;
             cycle[k++] = cyclestring;
          }
       }
    }
     boolean exitcycle=false;        //   檢驗有無 Cycle
    //Spanning Tree
    for (int i = 1; i < k ; i++){
       for (int j = 1; j <= count; j++){
          if(euler1[Integer.parseInt(cycle[i].substring(cycle[i].lastIndexOf("[")+1))][j]>0                    &&
j!=Integer.parseInt(cycle[i].substring(0, cycle[i].indexOf("]")))){
             countlist[k] = countlist[i]+1;
            // set[k].add(String.valueOf(j));
             cycle[k++] = cycle[i]+"]["+j;
             c = k-1;
             System.out.println("\ncycle[" +c+ "]=" + cycle[c]);
             System.out.println("\ncountlist[" +c+ "]=" + countlist[c]);
          //   System.out.println("\ncountlist[" +i+ "]=" + countlist[i]);
          }
          else                if(euler1[Integer.parseInt(cycle[i].substring(cycle[i].lastIndexOf("[")+1))][j]>0          &&
j==Integer.parseInt(cycle[i].substring(0, cycle[i].indexOf("]")))){

             countlist2[b] = countlist[i]+1;
             if(Integer.parseInt(cycle[i].substring(0, cycle[i].indexOf("]")))!=firstword){
               cyclenumber[countlist2[b]] ++;

               System.out.println("\ncountnumber[" +countlist2[b]+ "]=" + cyclenumber[countlist2[b]]);
             }
             cycleshow[b] = cycle[i]+"]["+j;
             if(cyclenumber[countlist2[b]] == availablestate){
                exitcycle = true;
                cyclepoint = countlist2[b];
                j=count+1;
                i=k+1;
             }
             b++;
             a = b-1;
             firstword =Integer.parseInt(cycleshow[a].substring(0, cycleshow[a].indexOf("]")));
          }
          if (countlist2[a] > availablestate){
          break;
```

```
      }
    }
  }

  //若具有相同的生產循環 存入另一個檔案
  String startnumber, last;     //紀錄讀取的點
  Set bag = new HashSet();       //儲存狀態轉移的狀態組合
  Set bag2 = new HashSet();     //儲存狀態轉移的狀態組合 2
  int p=0;   //變數累加
  String manucycle[]=new String[k];        //將有等效的生產循環存在 manucycle
  if(exitcycle==true){
     outputArea3.append("\n 存有生產循環\n");
     for(int i = 1 ; i < k ; i++){
        if(countlist2[i]==cyclepoint){
           manucycle[p++]=cycleshow[i];
        }
     }
  }
  else{
     outputArea3.append("\n 沒有存在相同的生產循環");
  }

  //檢驗相同的組合併刪除
for(int i = 0 ; i < manucycle.length ; i++){
     if(manucycle[i]!=null){
        System.out.println("\nmanucycle" + manucycle[i]);
        last = manucycle[i];
          for(int j = 1 ; j < cyclepoint ;j++){
             startnumber = last.substring(0,last.indexOf("]"));
             bag.add(startnumber);
                  last = last.substring(last.indexOf("[")+1);
              }
             System.out.println("\nbag" + bag.toString());
          for(int g = i+1 ; g < manucycle.length ; g++){
             if(manucycle[g]!=null){
                last = manucycle[g];
                for(int j = 1 ; j < cyclepoint ;j++){
                   startnumber = last.substring(0,last.indexOf("]"));
                   bag2.add(startnumber);
                                last = last.substring(last.indexOf("[")+1);
                           }
             }
             if(bag.equals(bag2)){
                           manucycle[g]=null;
             }
             bag2.clear();
          }

          //相同性
          bag.clear();
     }
}
int y=0;
//印出所有循環
for(int i = 0 ; i < manucycle.length ; i++){
   if (manucycle[i]!=null) {
      manufacturCycle[y++] = manucycle[i];
      outputArea3.append("\n"+y+":[" + manucycle[i] + "]");
   }
}
}
```