如何讓電腦計算網路流量
—— 組合最佳化的介紹

賀培鑫

1. 前言

老王問：「到電影院最近的路是那條?」公司老闆問:「我要分配那些工作給那個人做最適當?」造船廠工程師問：「我的船要多久才能造好呢?」航空公司經理問：「如何能用最少架的飛機來飛所有預定的班次呢?」以上這些問題, 都是組合最佳化(Combinatorial Optimzation) 的應用範圍。大致來說, 組合最佳化就是研究如何在事物有限但衆多的不同組合當中, 很快的找到最佳組合的一門學問。而這個找到最佳組合的方法, 稱之爲演算法 (Algorithm)。組合最佳化是離散數學中很重要的一支, 一方面是因爲它的應用很廣, 另一方面由於電腦的發展, 人們可以把演算法寫成程式, 讓電腦來替我們計算結果, 如此大大提高了複雜演算法的實用價值。

那我們要如何才能設計出快速的演算法呢? 基本上, 我們要以智取, 不以蠻力取勝。我們必須先研討問題的組合結構, 利用其組合特性, 才有可能成功。所以組合最佳化和找六合彩的號碼組合, 是一點關係也沒有的, 這是因爲隨機產生的號碼, 不具組合特性的緣故。本文將藉著對一類問題的探討, 介紹一下組合最佳化的問題以及所用的工具。在此討論的三個問題稱之爲最大水流問題(maximum flow problem), 最小支出水流問題(minimum cost flow problem), 以及最大動態水流問題(maximum dynamic flow problem); 以上三者通稱爲網路流量問題(network flow problems)。本文將儘量避免使用過多的數學符號, 以增加本文的可讀性。

2. 最大水流問題

第一個問題是最大水流問題, 這個問題的應用並不一定限於計算自來水管網路的最大輸送量。台北市捷運系統完成之後, 在同一時間內, 能送多少人由木柵到火車站呢? 現在台北到大陸的電話, 是透過香港, 日本, 及美國的電話線路轉接的, 同時能容許多少人由台北打電話到上海呢? 思考電腦 (Thinking Machine Company) 的連結電腦 (Connection Machine 5) 有1024個處理器, 這一部份的處理器, 同時能送多少訊息

給另一部份的處理器呢? 以上的三個問題, 事實上都是最大水流問題的特例。這一類的問題, 都包含一個網路 (network), 網路上運送的可以是任何東西, 但我們都以水流來想, 以有利於問題的思考。

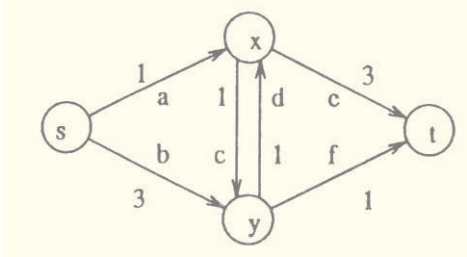

圖 1

我們用一個圖形 (graph), 來表示"一個網路" (圖 1) 它包含了一些"點" (圖 1 的中的圓圈) 和一些由一點到另一點的"有向邊"(directed edge)(圖 1 中的箭頭), 它代表著水能由一點流到另一點。圖形中還有一個"源點" s(source), 和一個"終點" t(sink)。在有向邊上的數字代表此邊的"負載量", 問題即是在同一時間內, 我們最多能由源點 s 送多少的水到終點 t 呢? 聰明的讀者一定已經發現, 爲什麼前面的三個問題都是最大水流問題的特例了罷。

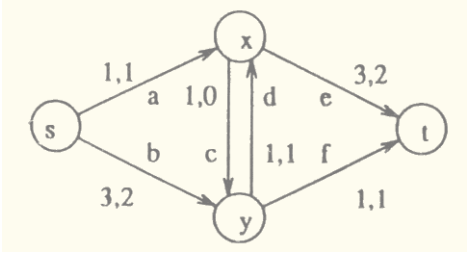

圖 2

圖 2 顯示的是個網路上水流的例子, 有向邊上的第二個數字代表的是此時流過此邊的流量。一個合法的水流必需滿足二個條件: (1) 除了源點和終點外, 流進一個點的流量總和與流出的流量總和相等, 和 (2) 每條邊的流量不得大於負載量或小於零。一個水流的流量, 即爲所有有流入終點 t 的水流量的總和, 因此圖中的水流流量爲 3, 那麼圖二中的水流是不是一個最大水流呢?

組合最佳化的工作者常常試著從不同的角度來看問題。一個同時由問題的正反兩面探討的解法, 有時稱之爲對偶法(primal-dual method)。我們正是要用此法來解決最大水量問題。試想是什麼限制了最大流量? 如果我們把由 s 到 x 及 s 到 y 的有向邊中間切一刀來把源點 s 和終點 t 分開 (如圖 3),

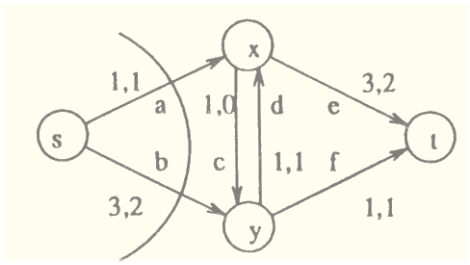

圖 3

我們可以發現, 所有由 s 到 t 的水, 必須經過這個切口, 因此由 s 到 t 的最大水流量絕不可能超過 4, 這是因爲由 s 出去的二根有向邊的負載量之和是 4 的緣故。我們稱 s 出去的邊 a 和 b 爲網路的一個截面 (cut), 截面的負載量爲其中有向邊的負載量之和。截面並非只有一個; 若我們去掉一些有向邊後, 使得 s 再也不能送水到 t 去, 則這些邊所成

的集合便是一個截面。值得一提的是, 因爲流至 t 的水必須通過每一截面, 所以我們可以想見, 最大的水流量, 是不可能超過最小的截面負載量的。由此可見, 若我們可以找到一個水流其流量相等於任何一個截面負載量, 便可推知此一水流必爲一最大水流。由於有向邊 a, d 和 f 構成一個負載量爲 3 的截面, 因此圖2 中的水流已是一最大水流。知道在什麼情況下我們可確定已找到一最大水流後, 我們該如何找一個方法來算最大水流呢? 這需要研究一番才行。

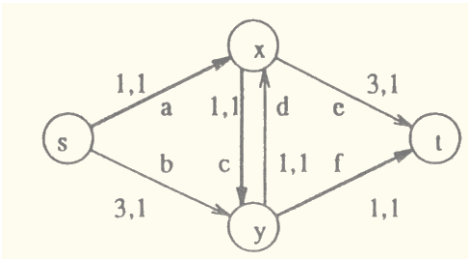

圖 4

請看圖4, 其中的邊 a, c, f 形成一個由 s 到 t 的通路, 稱之爲一「有向路徑」(directed path)。我們最多只能放一單位的水經此有向路徑送至終點 t, 因爲一有向路徑不能輸送超過其中任一有向邊的負載量的水。同樣的, 邊 b, d, 及 e 也構成一條有向路徑, 其負載量也是是 1。如果我們把二條有向路徑上的水流合併, 形成的水流量爲 2, 這並不是最大水流量, 但此時我們已經無法再找到一條新的有向路徑可以把水由 s 送到 t 了。因此我們發現, 用分別找一條條有向路徑的方法來求最大水流, 是不會成功的。但再分析一下圖四, 我們知道應該做的是由 s 多送一單位的水到 y, 把由 x 到 y 的水“推回”x, 改由 x 送到終點 t。結果流入 y 的水總和不變, (由 s 到 y 的水量增加, 但由 x 到 y 的水減少), 且流出點 y 的水量總和也不變 (我們沒做任何改變)。我們這項大禹治水的結果, 產生了一個最大水流。但是我們該如何教會電腦這個“治水之法”呢? 解答是由程式根據目前的網路水流造出一剩餘圖 (residual graph) 來標明所有可以往前和往回推水的地方, 以方便電腦來發現最大水流。圖 5 中的圖形即爲圖 4 網路的剩餘圖。

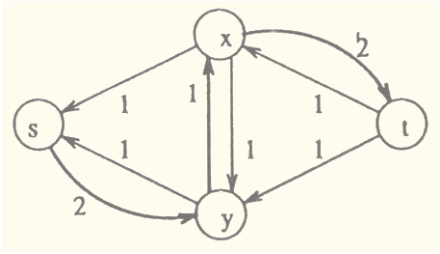

圖 5

剩餘圖和原網路有相同的點, 但可能有較多的邊。若一原網路中的有向邊上的水量小於負載量, 在剩餘圖中就有一個對應的「送出邊」(forword edge), 其負載量爲原有向邊負載量與水量之差, 即還剩下可供輸送的水量。若一原網路的有向邊已有水量 g 流於其中, 在剩餘圖中就有一個對應的反方向的邊, 其負載量爲 g。這種邊我們稱之爲「推回邊」(backword edge), 它標明了可以推回去的水量。圖5 中的由 s 到 y 的邊爲一送出邊, 其負載量爲 2, 因爲在圖4 的網路中邊 b 的負載量爲 3 但僅有水量 1, 其差爲 2。圖 5 中的由 y 到 x 的邊爲一推回邊, 其負載量

爲 1, 因爲在圖 4 中邊 c 的水量爲 1。注意邊 c 並不對應於圖 5 中任一送出邊, 因爲邊 c 的水量已經飽和, 我們不能再由它送出任何水。仔細一看, 在圖 5 剩餘圖中僅有的一條由 s 到 t 的向路徑正好對應於前面提到的治水法。所以由剩餘圖中的一有向路徑, 我們可以知道應如何修正原有的水流, 以增加流量, 所以我們稱在剩餘圖中的一有向路徑爲一增加路徑(augmenting path)。明顯的, 修正的方法如下: 對於增加路徑中送出邊的水量, 我們在原網路對應之有向邊中多送出此一水量。對於增加路徑中推回邊中的水量, 我們在原網路對應之有向邊少送此水量。修正之後, 在原網路中增加的水量, 恰等於增加路徑的負載量。至於圖 2 網路水流的剩餘圖則在圖 6, 其中不包含任一增加路徑, 因圖 2 中的水流已是最大水流。

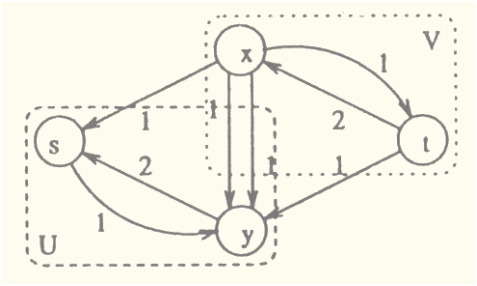

圖 6

3. 最大水流問題的演算法

由以上的發現, 我們可以設計一個演算法來計算最大水量。首先從現存的網路中的水量 (剛開始是零), 我們造一個剩餘圖出來。在剩餘圖中我們一步步的找一條增加路徑。找法如下: 我們把源點 s 可以用一根有向邊走到的所有點作一個記號 s; 再把所有可以從這些新走到的點能用一根有向邊走到點作一個記號, 來說明是從那個點走來的; 如此繼續作記號下去, 直到終點 t 被作了記號或是我們沒辦法再給新的點作記號爲止。當我們發現 t 被做了記號, 我們可以由 t 上的記號追查到前一個點是誰, 並一直查到源點 s 爲止, 如此便找出了一個增加路徑。根據這個增加路徑, 我們修正了原有的水流後, 再回到開始處去造新的剩餘圖, 繼續改進我們的水流。當我們發現我們沒法再給新的剩餘圖中的點作記號, 我們就停止演算法的執行並且把目前網路中的水流做爲最大水流問題的答案。

我們怎能確定用前述方法可找到最大水流呢? 如我們討論過的, 我們只要證明網路中有一截面其負載量相等於求得的水量即可。現在就讓我們把這個截面給揪出來。在程式停止前, s 和所有帶有記號的點形成一點集合 U, 當然, t 不在 U 內, 否則我們便已發現了一個增加路徑。讓集合 V 爲包括 t 在內的所有沒有記號的點。則我們可以斷定, (1) 在剩餘圖中沒有由 U 到 V 的邊, 也就是說, 在原網路中, 所有由 U 到 V 的有向邊, 均已飽和, 且 (2) 所有由 V 到 U 的有向邊, 均沒有任何水流。因爲所有由 U 到 V 的水都沒有再流回在 U 中的點, 所以所有由 U 到 V 的水都流到 t 裡去了, 因此我們找出的水流流量相等於所有由 U 到 V 的有向邊上的水流量之和。由於所有這種有向邊均已飽和, 這個流量又等於所有由 U 到 V 的有向邊的負載量之和。因爲所有由 U 到 V 的有向邊形成一截面, 且截面負載量相等於這些邊的負載量

之和, 因此這截面的負載量等於我們所求得之水流流量。這證明了我們的方法永遠找得到最大水流, 同時也證明了以下的二個定理: (1) 最大水流量等於最小截面負載量, 和 (2) 網路中的水流是最大水流若且唯若剩餘圖中不存在一增加路徑。以圖6 爲例, U 包含 s 和 y, V 包含 x 和 t, 對應於原網路的截面包含邊 a, d 及 f, 其負載量爲 3。

那這個方法究竟快不快呢? 我們用什麼標準來評判一個方法的快慢呢? 假設所有的負載量均爲非負整數, 則很明顯的, 每一條增加路徑至少有負載量 1。若最大流量爲 f^*, 則我們最多修正水流 f^* 次。每次修正前都要造剩餘圖, 找增加路徑, 和修正原水流。造剩餘圖要考慮每個網路的點和邊。若在有 n 個點 m 個邊的情況下, 造剩餘圖所花的時間會正比於 $n+m$, 我們說 $O(n+m)$ 爲造剩餘圖的時間複雜度(Time complexity)。同理, 找增加路徑及修正原水流所花的時間亦正比於 $n+m$, 故時間複雜度也都是 $O(n+m)$, 而全部修正時間亦因此正比於 $(n+m)$, 故時間複雜度就是 $O(n+m)$, 結果整個方法的時間複雜度爲 $O(f^*(n+m))$。Tarjan [T] 用類似的方法配合上高超的資料結構技巧, 設計出一 $O(nm\log n)$ 的方法, 這要比我們的方法好很多, 因爲它的複雜度與 f^* 的值無關。我們的方法的複雜度因與 f^* 的值成正比, 是一種"指數時間"的演算法, 比不上 Tarjan 的"多項式時間"演算法, 其詳細原因不在本文討論範圍之內。

4. 最小支出水流問題

現在讓我們來看看第二個問題: 最小支出水流問題。這個問題可以經由修改我們求最大水流的方法來解決。讓我們來考慮一個要收費的網路, 如圖7 和圖8。每根有向邊上有三個數字, 最左邊的數字代表負載量, 第二個數字是每單位水流通過此有向邊需付出的「買路財」, 第三個數字是目前的水量。所以圖7 的水流要花費17, 圖8 的水流要花費18, 且二圖中的水流均爲最大水流。最小支出水流問題即是要找一個總價所低的最大水流。不知道電信局在租借外國線路時有沒有考慮過這個問題?

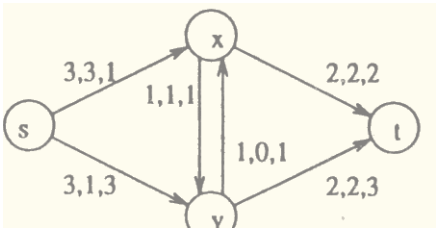

圖 7

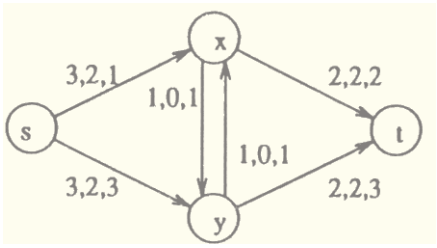

圖 8

最小支出水流問題比最大水流問題要難。對一個負載量及花費均爲非負整數的網路, 我們可以修改最大水流的演算法來解決此問題。我們只要在找增加路徑時, 永遠找花費最低

的一條即可, 此處一路徑的花費爲其上有向邊的單位花費之和。Dijkstra 有一個在圖形上找源點和終點間最短路徑的演算法, 若配合上資料結構 Fibonacci heap [K], 其時間複雜度爲 $O(m + n\log n)$。若我們把距離改爲花費, 則最短路徑即爲最廉路徑, 所以我們可以用 Dijkstra 的演算法在剩餘圖上找最廉增加路徑來修正現有水流。如此一來, 若最大流量爲 f^*, 我們的演算法的時間複雜度爲 $O(f^*(m + n\log n))$。

5. 最大動態水流問題

最後讓我們來看看最大動態水流問題。假如有台北市議員問捷運局長, 在早上 7:30 到 8:30 之間, 我們的捷運系統能送多少人由木柵到火車站呢? 此時局長應立刻回答: 這是一個最大動態水流問題, 並拿出一筆記本型電腦, 在動了動鍵盤上的小球後, 螢幕上立即顯示出答案。如果事情眞的如此, 筆者猜局長一定可以升官了。的確, 最大動態水流問題是考慮在一段時間內能流進終點 t 的最大水流問題。

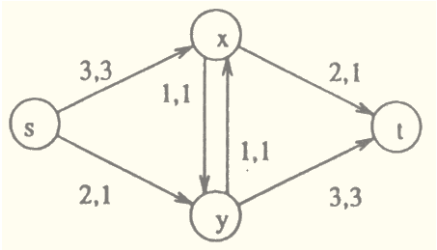

圖 9

圖9 [FF] 是一個簡單的例子, 有向邊的第一個數字爲負載量, 第二個數字是運送水由有向邊這一端到另一端所需的時間, 假定此地時間的單位爲秒。所以, 由 s 送水到 x 要花三秒鐘。我們想問, 在 5 秒內, 最多能有多少水由 s 流入 t 呢? 圖 10.1 到 10.5 代表了第一秒至第五秒的網路流量狀況, 這個動態水流能在 5 秒內送 8 單位的水由 s 到 t。在第一秒時, s 可以送 2 單位的水到 y, 但 s 送到 x 的水還在三分之一的路上, 因爲要花三秒鐘才能抵達 x。在第二秒時, s 又送了 2 單位的水到 y, 先前送到 y 的水, 已兵分二路, 分別抵達 x 和向 t 前進。在第三秒時, y 已不需再送水出去, 因爲此時送水已晚了, 無法在時限內抵達 t。此時已有一單位的水流入 t。在第四及第五秒時, 各有3個單位和 4 個單位的水流入 t, 故流入 t 的水量總和爲 8 個單位。

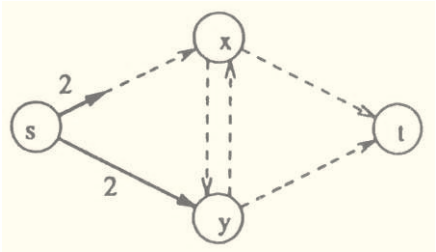

圖10.1

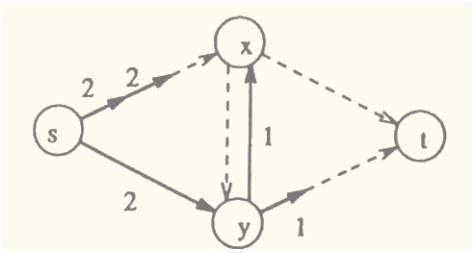

圖10.2

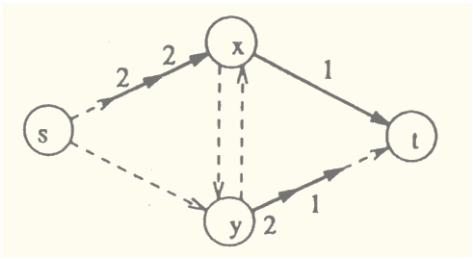

圖10.3

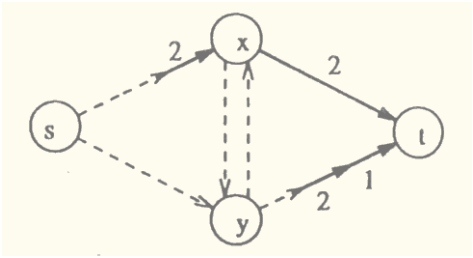

圖10.4

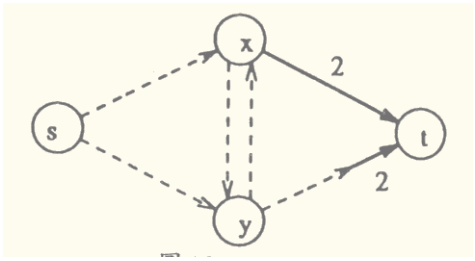

圖10.5

解決這個問題最容易的方法爲轉換此最大動態水流問題成爲一最大水流問題。這個方法的缺點是其時間複雜度較大。我們根據圖 9 的動態網路, 可以造出一不含時間限制的網路 (圖 11)。

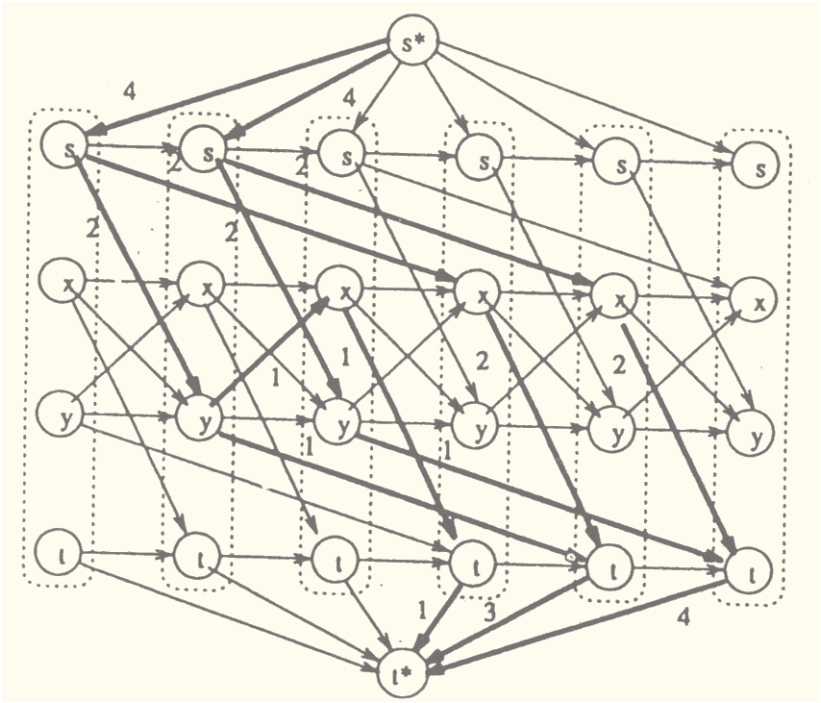

圖 11

這個新的網路的最大水量相等於原動態網路的最大動態水流量。這個轉換而來的網路, 稱之爲擴張網路(expanded network), 它對每一時間單位, 均有一份原動態網路的拷貝。故若原網路有 n 個點, 但時間限制爲 k, 則其擴張網路有 $kn+2$ 個點。假設在原網路中由點 u 到點 v 要花 m 單位時間, 則在擴張網路中第 i 個拷貝中的點 u, 有一條有向邊到第 $i+m$ 個拷貝中的點 v。請看圖11, 第 0 秒的點 s 有一根有向邊到第 3 秒的點 x, 因爲由點 s 到點 x 在原網路中要花三秒。同理, 第 1 秒的點 y 有一根有向邊到第 4 秒的點 t, 因爲由點 y 到點 t 在原網路中要花三秒。在擴張網路中, 我們也加入新的源點 s^* 及終點 t^*。圖11 上的粗線標明了此網路上的一個最大水流, 聰明的讀者在稍加分析之後, 一定可以理解爲何擴張網路中的最大水流對應於網路中的最大動態水流。若時間限制爲 k, 在擴張網路中有 k 個原網路的拷貝, 故這個轉換方法的時間複雜度變成 $O(kf^*(m+n))$, 這不是一個很快的方法。較好的解決辦法是利用最小支出水流的演算法來做 [A,FF], 此法不在本文中討論。

網路流量問題已經被研究了近50年, 但現在仍是一個相當活躍的領域。本文介紹了三個不同的網路流量問題的應用及簡單的演算法。但願能激起讀者對組合最佳化這門學問的興趣。

參考書目:

[A] Aronson J. E., A survey of dynamic network flows, Annals of Operational Research 20 (1989), 1-66.

[FF] Ford L. R. and Fulkerson D. R., Flows in Networks, Princeton Uniersity Press, 1962.

[K] Kozer D. Design and Analysis of Algorithms, Springer-Verlag, 1992.

[PS] Papadimitriou C. H. and Steiglitz K., Combinatorial Optimization: Algo- rithms and Complexity, Prentice-Hall, 1982.

[T] Tarjan R. E., Data Structures and Network Algorithms, SIAM, 1983.

—本文作者爲美國康乃爾大學計算機科學系博士候選人—