

限制滿足問題



本章中我們會看到不把狀態僅僅當作小黑盒子時，會如何引導設計出更強有力的新搜尋方法，以及對問題的結構和複雜性有更深入的理解。

第三章和第四章探索了透過搜尋狀態空間求解問題的想法。這些狀態可以透過特定值域的啟發式演算法加以評價，並且測試它們是否為目標狀態。然而，從搜尋演算法的觀點看，每個狀態都原子的，或不可分的——是一個無內在結構的黑盒子。

本章描述一種更有效率的方式來求解各式各樣的問題。我們使用每個狀態的**因式表示法**：一個變數集，集合中的每個變數都有一個值。當每個變數都有一個值能滿足所有加諸於該變數的限制，該問題就算被解出。以這樣的方式被描述的問題稱之為**限制滿足問題**，或 CSP。

CSP 搜尋演算法利用狀態結構的優勢並使用一般性的而非問題特定的啟發式來使複雜問題的解答變得可能。主要的構想是經由辨識出那些違反限制的變數/值之組合來一次大量削減搜尋空間。

6.1 限制滿足問題的定義

一個限制滿足問題由 X 、 D 、 C 三個成份所組成：

X 是一個變數 $\{X_1, \dots, X_n\}$ 的集合。

D 是一個值域 $\{D_1, \dots, D_n\}$ 的集合，每個變數各有一個。

C 是一個限制的集合，指明值可允許的組合方式。

每個值域 D_i 是由變數 X_i 可允許的值 $\{v_1, \dots, v_k\}$ 之集合所組成。每個限制 C_i 由 $\langle \text{scope}, \text{rel} \rangle$ 對所組成，其中 scope 是參與限制的變數之元組，而 rel 是定義變數能賦予哪些值的關係。關係可以用一張明列出所有滿足該限制之值的元組的名單來代表，或是以一個抽象關係來代表，該關係支援兩個操作：測試元組是否為該關係的成員，以及列舉該關係的成員。舉例來說，如果 X_1 與 X_2 都有值域 $\{A, B\}$ ，那麼兩個變數必須有不同值的限制可以被寫為 $\langle (X_1, X_2), [(A, B), (B, A)] \rangle$ 或為 $\langle (X_1, X_2), X_1 \neq X_2 \rangle$ 。

欲求解一個 CSP，我們必須定義一個狀態空間與一個解的符號。CSP 中的每個狀態定義為一些或是所有變數的**賦值**， $\{X_i = v_i, X_j = v_j, \dots\}$ 。一個不違反任何限制條件的賦值，稱作相容的或者合法的賦值。**完全賦值**是讓每個變數有賦值，而 CSP 問題的**解**是一個一致的完全賦值。**不完全賦值**指的是僅僅設定值予某些變數。

6.1.1 範例問題：地圖著色

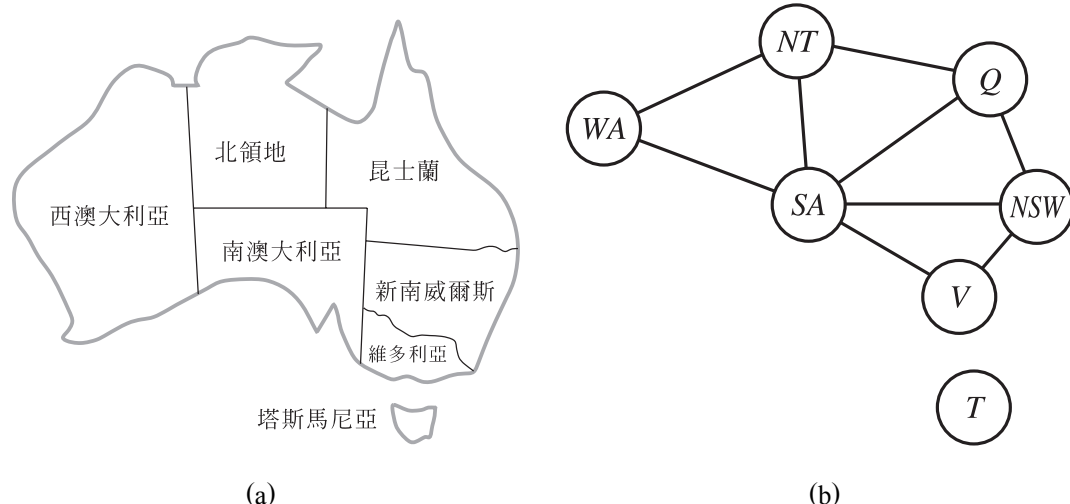


圖 6.1 (a) 澳大利亞主要的州和地方行政區。對此地圖的著色可以視為一個限制滿足問題。目標是對每個區域分配顏色，使得相鄰的區域不同色。(b) 表示地圖著色問題的限制圖

假設，厭倦了羅馬尼亞，我們看看展示澳洲各州以及其領土的地圖(圖 6.1a)。我們被交付的任務是為每個地域塗上顏色，可以是紅色，綠色，或藍色，著色的條件是兩個相鄰的區域不能塗上同樣的顏色。要將這個題目改寫成一個 CSP 問題，我們將各區域定義成變數

$$X = \{WA, NT, Q, NSW, V, SA, T\}$$

每個變數的值域是集合 $D_i = \{red, green, blue\}$ (即紅綠藍三色)。限制是要求隔鄰的區域要有不同的顏色。因為有九個地方是彼此交界的，所以有九個限制：

$$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

這裡使用縮寫； $SA \neq WA$ 是 $\{(SA, WA), (WA, SA)\}$ 的簡寫，其中 $SA \neq WA$ 能夠被完整地逐一地列出如

$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$$

這個問題有很多可能的解，如：

$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red\}$$

把 CSP 視覺化地表示為限制圖是有用的，如圖 6.1(b)所示。圖的節點對應於此問題的變數，一個連接將兩個參與某個限制的變數連在一起。

為什麼要將一個問題公式化為一個 CSP 問題？一個原因是 CSP 對許多類型的問題能得到很自然的表示；如果你已經有一個 CSP 求解系統，使用它來求解一個問題較諸使用另一個搜尋技術來設計個定製的答案來得容易。除此之外，CSP 求解器的速度比狀態空間搜尋器來的快，因為 CSP 求解器能快速地消除大塊的搜尋空間。舉例來說，一旦在澳洲問題中我們已經選擇 $\{SA = blue\}$ ，我們能得到的結論是五個相鄰之變數無一能夠為藍色。如果沒有利用到限制傳播的好處，則搜尋程序必須考慮這五個相鄰變數的 $3^5 = 243$ 種賦值；有了限制的傳播我們就不必將藍色列入考慮，所以我們只需要檢查 $2^5 = 32$ 個賦值，減少了 87%。

於正常的狀態空間搜尋我們只能問：這個特定的狀態是目標嗎？不是？那麼這個呢？有了 CSP，一旦我們發現某個部分賦值不是解答的時候，我們能夠立即將之拋棄進而調整該不完全賦值。進而，我們能看出為什麼該賦值不是解答——我們看看哪一個變數違背了限制條件——因此我們能將注意力集中在事關重要的變數。因此，許多正常狀態空間搜尋無法處理的問題，在予以公式化成 CSP 後，能夠被迅速地解出。

6.1.2 範例問題：加工車間排程

有整天作業排程問題的工廠，承受了各式的限制。實際上，許多這些問題都已透過 CSP 技術被解出。考慮一輛車子的組裝排程問題。整個作業是由數個任務所組成，而我們能夠將每個任務塑造如同一個變數，其中每個變數的值是任務開始的時間，以分鐘數來表示。限制條件能夠要求說某個任務必須在另一個任務之前出現——舉例來說，一個車輪必須先於輪轂罩裝上之前安裝好——且只有如此許多的任務才能夠同時開展。限制也能夠指明某個任務需耗時多久才能完成。

我們考慮一小部份的車輛組裝任務，共有 15 個任務：安裝車軸(前面和後面)，裝上所有四個輪子(左/右/前/後)，擰緊每個車輪的螺母，裝上轂蓋，最後組裝檢查。我們能夠以 15 個變數來表示這些任務：

$$X = \{AxleF, AxleB, WheelRF, WheelLF, WheelRB, WheelLB, NutsRF, NutsLF, NutsRB, NutsLB, CapRF, CapLF, CapRB, CapLB, Inspect\}$$

每個變數的值是任務開始的時間。接著我們表示出各個任務間的**前提限制**。每當一個任務 T_1 必須在任務 T_2 之前出現，而任務 T_1 需耗用時間 d_1 來完成，我們加入一個如下形式的數學性限制

$$T_1 + d_1 \leq T_2$$

在我們的例子中，在安裝輪子之前，輪軸必須先到位，放置車軸需要 10 分鐘，所以我們寫成

$$\begin{aligned} AxleF + 10 &\leq Wheel_{RF}; AxleF + 10 \leq Wheel_{LF}; \\ AxleB + 10 &\leq Wheel_{RB}; AxleB + 10 \leq Wheel_{LB} \end{aligned}$$

然後我們說，對每個輪子，我們必須使輪子固定好(需時 1 分鐘)，然後擰緊螺母(2 分鐘)，最後附上輪轂罩(1 分鐘，但是還沒有被表示出來)：

$$\begin{aligned} Wheel_{RF} + 1 &\leq Nuts_{RF}; Nuts_{RF} + 2 \leq Cap_{RF}; \\ Wheel_{LF} + 1 &\leq Nuts_{LF}; Nuts_{LF} + 2 \leq Cap_{LF}; \\ Wheel_{RB} + 1 &\leq Nuts_{RB}; Nuts_{RB} + 2 \leq Cap_{RB}; \\ Wheel_{LB} + 1 &\leq Nuts_{LB}; Nuts_{LB} + 2 \leq Cap_{LB} \end{aligned}$$

假設我們有四個工人安裝輪子，但是它們必須共用一個能協助輪軸定位的工具。我們需要一個**選言限制**(disjunction constraint)來說明 $AxleF$ 與 $AxleB$ 在時間上必須不能重疊；不是其中一個先做，就是另一個先做：

$$(Axle_F + 10 \leq Axle_B) \text{ 或 } (Axle_B + 10 \leq Axle_F)$$

這看起來像是一個更複雜的限制，結合了數學與邏輯。但是它仍然被簡化成 $Axle_F$ 與 $Axle_B$ 能適用之成對值的集合。

我們也必須聲明检查工作會最後完成並需時 3 分鐘。除了 *Inspect*，對每一個變數我們加了這個形式 $X + d_X \leq \text{Inspect}$ 的限制。最後，假設有一個需求是整個組裝必須在 30 分鐘內完成。我們能夠藉由限制所有變數的值域來達成：

$$D_i = \{1, 2, 3, \dots, 27\}$$

這個特別的問題很容易解出，但是已經被應用於如此類的加工車間排程問題的 CSP 多具有上千個變數。於某些情況，有些複雜的限制很難以 CSP 形式化的方法來具體說明，此時會援用更高層次的規劃技術，如第 11 章所述。

6.1.3 CSP 形式化的變種

最簡單的一種 CSP 其涉及的變數具有**離散且有限**的值域。地圖著色的問題以及有時間限制的排程這兩者都屬於這一種。第三章中描述的八皇后問題也可以視為有限值域的 CSP，其中變數 Q_1, \dots, Q_8 是每個皇后在列 1, ..., 8 中的位置，每個變數的值域是 $\{1, 2, 3, 4, 5, 6, 7, 8\}$ 。

一個離散值域可能是**無限**的，如整數或是字串的集合(如果我們不對作業排程問題加上截止期限，那麼每個變數的開始時間就會有無限多個)。用無限值域，不再可能透過列舉所有可能取值的組合來描述限制條件了。取而代之的做法是，必須借用**限制語言**來了解限制如 $T_1 + d_1 \leq T_2$ ，不用一一列出可允許之 (T_1, T_2) 值對的集合。對於整數變數的**線性限制**——也就是，諸如剛剛給出的限制，其中每個變數都只以線性形式出現——存在特殊的求解演算法(我們在這裡不討論)。可以證明沒有演算法能夠求解一般的整數變數的**非線性限制**問題。

連續值域的限制滿足問題在現實世界是十分常見的，並在作業研究值域中有廣受研究。例如在哈伯太空望遠鏡上的實驗排程安排要求非常精確的觀測時間選擇；每次觀測的開始、結束時間和機動時間都是連續值變數，它們必須遵守許多天文的、優先權的和電力的限制。最著名的連續值域 CSP 是**線性規劃**問題，其中限制必須是構成一個凸多邊形的一組線性不等式。線性規劃問題可以在與變數數有關的多項式時間內求解。有不同類型的限制和目標函數的問題也已廣為研究——二次規劃，二階二次曲線規劃，等等。

除了考察出現在 CSP 問題中的變數的種類，考察限制的類型也是有用的。最簡單的類型是**一元限制**，它只限制單個變數的取值。舉例來說，於地圖著色問題可能有南方澳洲人不會容忍綠色的情況；我們能將之表示為一元限制 $\langle SA, SA \neq \text{green} \rangle$ 。

一個**二元限制**與兩個變數有關。例如， $SA \neq NSW$ 就是一個二元限制。二元 CSP 只包含二元限制；它可以表示為限制圖，如圖 6.1(b)。

我們也能描述更高層次的限制，如聲明 Y 的值介於 X 與 Z 之間，具有三元限制 $\text{Between}(X, Y, Z)$ 。

一個可以有任意個變數數目的限制被稱之為一個**全局限制**(這名稱很傳統但是容易混淆因為它不要求於一個問題中用到所有的變數)。最常見的全局限制之一是 *Alldiff*，說的是與限制有關的變數必須有不同的值。於數獨(Sudoku)問題(見第 6.2.6 節)，於一行或列的所有變數必須滿足 *Alldiff* 限制。另一個例子是**密碼算術遊戲**(參見圖 6.2b)。於一個密碼算術遊戲的各個字母代表不同的數字。在圖 6.2(a)所示的情況下，這將表示為一個六變數限制 *Alldiff*(F, T, U, W, R, O)。遊戲中四欄的加法限制可以寫成如下數的 n 元限制：

$$O + O = R + 10 \cdot C_{10}$$

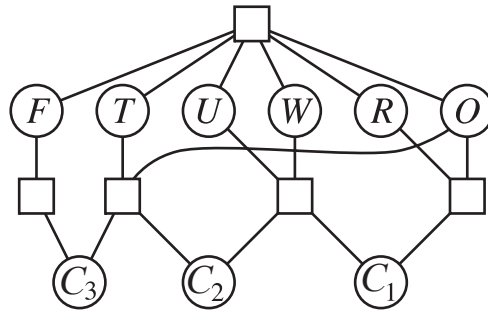
$$C_{10} + W + W = U + 10 \cdot C_{100}$$

$$C_{100} + T + T = O + 10 \cdot C_{1000}$$

$$C_{1000} = F$$

其中 C_{10} 、 C_{100} 、 C_{1000} 是代表十位數，百位數，或千位數之數字的輔助變數。高階限制可以用**限制超圖**表示，諸如圖 6.2(b)所示。超圖是由一般的節點(圖中的圓圈)與超節點(方格)組成，它代表表 n 元限制。

$$\begin{array}{r} T \ W \ O \\ + \ T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$



(a)

(b)

圖 6.2 (a) 一個密碼算術遊戲。每個字母表示一個不同的數字；目標是找到能使加法式子成立的代替字母的數字，附加限制是最前面的數字不能是 0。(b) 密碼算術問題限制超圖，顯示 *Alldiff* 限制(頂端的方格)，以及欄的加法限制(中間的四個方格)。變數 C_1 、 C_2 、 C_3 代表三個欄的進位數字。

另一方面，如習題 6.5 所要求你證明者，如果引進了足夠的輔助變數，每個有限值域的限制可以被簡化為一個二元限制的集合，所以我們可以轉換任何 CSP 為一個僅具有二元限制的問題；這使得這個演算法更簡單。另一個轉換 n 元 CSP 為一個二元問題的方法是**雙圖形轉換**(dual graph transformation)：製作一張新的圖，於這張圖中，為原圖中的每個限制提供一個變數，並且對於共享變數之原圖中的每對限制提供一個二元限制。舉例來說，如果原圖中有個變數 $\{X, Y, Z\}$ 與限制 $\langle(X, Y, Z), C_1\rangle$ 與 $\langle(X, Y), C_2\rangle$ 那麼雙圖形會有變數 $\{C_1, C_2\}$ 且具有個二元限制 $\langle(X, Y), R_1\rangle$ ，其中 (X, Y) 是共享變數且 R_1 是一個的新關係，它定義了共享變數之間的限制，由原來的 C_1 與 C_2 所指明。

不過有兩個原因為什麼我們偏好全局限制如 *Alldiff* 而非二元限制的集合。第一，使用 *Alldiff* 來描述問題會更容易且不容易出錯。第二，針對無法用於更原始限制集合之全局限制來設計專用的推論演算法有是可能的。我們於第 6.2.5 節描述這些推論演算法。

到目前為止我們已經描述過的限制都是絕對限制，任何違反規則的都排除在解之外。然而許多現實世界的 CSP 包含**偏好限制**，指出哪些解是更偏好的。舉例來說，在大學課程排程的問題上有個絕對限制是沒有任何教授能同一時間教授兩門課程。但是我們也可以允許偏好限制(*preference constraints*)：R 教授可能比較喜歡在早上授課，然而 N 教授比較喜歡在下午授課。X 教授在下午 2 點授課的時間表雖然是一個解（除非 X 教授正好是系主任），但不是最佳解。偏好限制通常被計入個體變數賦值的成本——例如，分配給 X 教授下午時段在總體目標函數中需要消耗 2 點，而上午時段只需要消耗 1 點。用這種形式化，有偏好限制的 CSP 問題可以用基於路徑的或局部的最佳搜尋方法求解。我們稱這樣的問題是**限制最佳化問題**，或 COP。線性規劃問題做這種最佳化。

6.2 限制的傳播：於 CSP 推理

於正常的狀態空間搜尋，一個演算法只能做一件事情：搜尋。於 CSP 有一個選擇：演算法能搜尋（從數個可能性選取一個新的變數賦值）或是做一個特定類型稱之為**限制傳播**(*constraint propagation*)的**推理**：使用限制以降低某個變數合法值之數目，從而降低另一個變數的合法值之數目等等。限制的傳播可能與搜尋緊密相連，或在搜尋開始之前，它可以被當做一個預先處理步驟。有時候這樣的預先處理能解出全部問題，所以根本不需要搜尋。

關鍵的概念是**局部相容性**。我們對待每個變數如同圖中的一個節點（見圖 6.1b）以及每個二元限制如同一個邊，那麼強化圖形各部份之局部相容性的過程所造成的不相容值會在整張圖中被消除。局部相容性有不同的類型，我們將逐一介紹。

6.2.1 節點相容性

一個單一個變數（相應於 CSP 網路的節點）屬於**節點相容**(*node-consistent*)，如果於該變數的值域中所有的值都滿足該變數的一元限制。舉例來說，在澳洲地圖著色問題的變種題目（圖 6.1）其中南澳人不喜歡綠色，變數 SA 以值域 $\{red, green, blue\}$ 開始，我們可以藉由消去綠色使之為節點相容，留下的是縮減了的值域 $\{red, blue\}$ 的 SA 。我們說一個網路是節點相容的，如果於該網路中的每一個變數是節點相容的。

藉由執行節點相容性來消去一個 CSP 中的限制總是可能的。轉換所有的 n 元限制為二元限制也是可能的（見習題 6.5）。因為這樣的原因，定義僅能運用於二元限制的 CSP 求解器就變得很常見；我們在本章其後的內容都做如此的假設，除了特別註明之外。

6.2.2 邊相容性

於一個 CSP 的一個變數是**邊相容**(*arc-consistent*)，如果變數的值域中的每一個值都滿足變數的二元限制。更正式地說， X_i 對另一個變數 X_j 是邊相容的，如果於目前的值域 D_i 中的每一個值，於值域 D_j 有一些值滿足加諸於邊 (X_i, X_j) 的二元限制。一個網路屬於邊相容性，如果每一個變數與其他的變數都是邊相容的。舉例來說，考慮限制 $Y = X^2$ 其中 X 與 Y 值域是數字的集合。我們能夠直接地寫出這個限制為

$$\langle (X, Y), \{(0, 0), (1, 1), (2, 4), (3, 9)\} \rangle$$

要使得 X 對 Y 是邊相容的，我們縮減 X 的值域成 $\{0, 1, 2, 3\}$ 。如果我們也是讓 Y 對 X 是邊相容的，那麼 Y 的值域變成 $\{0, 1, 4, 9\}$ 且整個 CSP 是邊相容的。

另一方面，邊相容性為澳洲地圖著色問題幫不上忙。考慮下述加諸於 (SA, WA) 的不等式限制：

$$\{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$$

不論你為 SA (或是 WA) 選取哪一個值，對其他的變數總有一個真確的值。所以施加邊相容性並不會影響任一個變數的值域。

最流行的邊相容性演算法已知是 AC-3 (見圖 6.3)。要使每一個變數為邊相容的，AC-3 演算法保留一個邊的佇列供作考慮之用。(實際上，考慮的前後順序並不重要，所以在資料結構實際上是一個集合，但是習慣稱它是一個佇列)。剛開始時，該佇列儲存了 CSP 中所有的邊。AC-3 然後從佇列中隨意地挑出一個邊 (X_i, X_j) 並且讓 X_i 對 X_j 為邊相容的。如果這讓 D_i 沒什麼改變，該演算法就移到下一個邊。但是如果更動了 D_i (使得該值域縮小些)，那麼我們將所有 (X_k, X_i) 邊加進佇列之中，其中 X_k 是一個 X_i 的鄰居。我們必須如此做，因為 D_i 的改變可能會讓 D_k 的值域進一步減縮，即使我們前面已經考慮過 X_k 了。如果 D_i 被修改得一無所有，那麼我們知道整個 CSP 沒有相容的解答，而 AC-3 會立即地傳回失敗。否則，我們一直檢查，嘗試把數值從變數的值域移出，直到佇列中沒有任何邊。在那個時候，我們手上有個等價於原始 CSP 的 CSP——它們兩個都有相同的解——但是在大多數情況下邊相容的 CSP 的搜尋會快速些，因為它的變數有更小的值域。

```

function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components  $(X, D, C)$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
    if REVISE(csp,  $X_i, X_j$ ) then
      if size of  $D_i = 0$  then return false
      for each  $X_k$  in  $X_i.\text{NEIGHBORS} - \{X_j\}$  do
        add  $(X_k, X_i)$  to queue
  return true



---


function REVISE(csp,  $X_i, X_j$ ) returns true iff we revise the domain of  $X_i$ 
  revised  $\leftarrow$  false
  for each  $x$  in  $D_i$  do
    if no value  $y$  in  $D_j$  allows  $(x, y)$  to satisfy the constraint between  $X_i$  and  $X_j$  then
      delete  $x$  from  $D_i$ 
      revised  $\leftarrow$  true
  return revised

```

圖 6.3 邊相容演算法 AC-3。在使用 AC-3 之後，每一個邊是邊相容的，或是某些變數有個空的值域，顯示出該 CSP 無法被解出。該演算法的發明者(Mackworth, 1977)使用「AC-3」這個名字，是因為這是他論文中的第三個版本

AC-3 的複雜度可以分析如下：假設一個有 n 個變數的 CSP，每個值域的大小最多是 d ，且具有 c 個二元限制(邊)。每個邊 (X_i, X_j) 可以被插入佇列最多 d 次，因為 X_i 最多有 d 個值可以刪除。檢查一個邊的相容性可以在 $O(d^2)$ 量級的時間內完成，所以我們得到最壞情況總時間是 $O(cd^3)$ [1]。

延伸邊相容性的符號來處理 n 元而非只是二元限制是可能的；這稱之為一般化邊相容性或有時候稱之為超邊相容性，視作者而定。稱一個變數 X_i 對一個 n 元限制是**一般邊相容**，如果於 X_i 的值域中的每一個值 v ，存在一個隸屬於該限制之成員的元組值，它的值全取自相應之變數的值域，並且讓它的 X_i 的分量等於 v 。舉例來說，如果所有的變數擁有值域 $\{0, 1, 2, 3\}$ ，那麼要使變數 X 與限制 $X < Y < Z$ 相容，我們必須從 X 的值域除去 2 和 3，因為當 X 等於 2 或 3 的時候，該限制是無法被滿足的。

6.2.3 路徑相容

邊相容性在朝向縮減變數的值域時，可能會走很長的路，有時候找到解答(藉由縮減每個值域的大小達到 1)而有時候發現 CSP 無法被解出(每個值域的大小縮減為 0)。但是對於其他網路，邊相容性無法製造足夠的推理。考慮澳洲地圖著色問題，不過只允許使用兩種顏色，紅色和藍色。邊相容性什麼也不用做，因為每一個變數已經是邊相容的：每個邊可能一端是紅而另一端是藍的(或者反過來也行)。但是很清楚地該問題沒有解答：因為西澳大利亞州，北領地和南澳都彼此接壤，就這三個區域我們就至少需要三種顏色。

邊相容性使用邊(二元限制)收緊值域(一元限制)。要想在地圖著色的問題上有所進展，我們需要更強的相容概念。**路徑相容**藉由檢視三倍的變數所推論出的間接性限制來收緊二元限制。

一個雙變數集合 $\{X_i, X_j\}$ 對第三個變數 X_m 是路徑相容的，如果對每一個賦值 $\{X_i = a, X_j = b\}$ 與加諸於 $\{X_i, X_j\}$ 之限制，有一個賦值 X_m 滿足加諸於 $\{X_i, X_m\}$ 與 $\{X_m, X_j\}$ 之限制。這稱之為路徑相容，因為你可以想像成正望著一條從 X_i 到 X_j 的路徑而 X_m 位在路中間。

讓我們看看路徑相容如何用兩種顏色塗澳洲地圖。我們會使得集合 $\{WA, SA\}$ 路徑相容於 NT 。我們列舉對集合相容之賦值開始。這樣的話，僅有兩個： $\{WA = red, SA = blue\}$ 與 $\{WA = blue, SA = red\}$ 。我們可以看到 NT 的這兩個賦值既不能是紅色的也不能是藍色的(它會與 WA 或 SA 的顏色發生衝突)。因為沒有適用於 NT 的選擇，我們除去這兩個賦值，而我們得到的結果是沒有合適的值可賦予 $\{WA, SA\}$ 。因此，我們知道對這問題不可能有答案。PC-2 演算法(Mackworth, 1977)達成路徑相容的方式與 AC-3 達成邊相容的方式非常相似。因為它是如此的相似，我們在這裡就不多做說明。

6.2.4 K-相容

用 **k 相容** 的概念可以定義更強的傳播形式。如果對於任何 $k-1$ 個變數的相容賦值，第 k 個變數總能被賦予一個和前 $k-1$ 個變數相容的值，那麼這個 CSP 問題就是 k 相容的。1-相容說的是，已知空集合，我們可以讓任何一個變數的集合是相容的：這我們稱之為節點相容。2 相容和邊相容是一樣的。對於二元限制網路，3-相容與路徑相容是相同的。

如果一個圖是 k 相容的，也是 $k-1$ 相容的、 $k-2$ 相容的、...，直到 1 相容，那麼這個圖是強 k 相容的。現在假設我們的 CSP 問題有 n 個節點而且令它是強 n 相容的(即 $k=n$ 時的強 k 相容)。我們然後能求解該問題如下：首先，我們對變數 X_1 選擇一個相容值。我們保證能夠給 X_2 也選擇一個相容值，因為它是 2 相容的，對 X_3 也如此，因為它是 3 相容的，等等。對每個變數 X_i ，我們只需要在值域內的 d 個值中找到與 X_1, \dots, X_{i-1} 相容的值。我們被保證在 $O(n^2d)$ 時間內找到解。當然，世上沒有免費的午餐：任何建立 n 相容的演算法在最壞情況下必須花費 n 的指數時間。更糟的是， n -相容也需要 n 的指數量級的空間。記憶體的問題比時間的問題更嚴重。實際上，決定合適的相容性檢驗層次的主要是經驗科學。可以說使用者通常會用 2 相容來計算而比較少用 3 相容。

6.2.5 全局限制

請記得**全局限制**說的是一個擁有任意數目之變數(但不必然是所有變數)的限制。全局限制經常出現在實際問題中，且用專用演算法處理的效率比用至今描述的通用方法要更高。例如，*Alldiff* 限制要求涉及的全部變數都必須取不同的值(如密碼算術問題)。*Alldiff* 限制問題的矛盾檢測的一個簡單形式包括如下工作：如果限制中涉及 m 個變數，且它們一共有 n 個可能的不同取值，並且 $m > n$ ，則這個限制不可能被滿足。

這引出了下面的簡單演算法：首先，刪除限制中只有單值值域的變數，然後將這些變數的取值從其餘變數的值域中刪去。只要還有單值變數，就重複這個過程。如果得到一個空的值域或者剩下的變數數比剩下的數值個數大，那麼就產生矛盾。

這個方法可檢測圖 6.1 中的賦值 $\{WA = red, NSW = red\}$ 的矛盾。注意變數 SA ， NT 和 Q 是透過 *Alldiff* 限制有效連接起來的，因為它們之間任何兩個都要著不同的顏色。在這個不完全賦值上應用 AC-3 演算法，每個變數的值域就縮小為 $\{green, blue\}$ 。就是說我們有三個變數但只有兩種顏色可選，所以違反了 *Alldiff* 限制。因此一個高階限制的簡單相容過程有時候比把邊相容用於二元限制的等價集合效率更高。也有更為複雜而能傳播更多限制但是執行上計算更昂貴的推理演算法可用於 *Alldiff* (見 van Hoes 及 Katriel, 2006)。

另一個重要的高階限制是**資源限制**，有時稱為**大多數**(atmost)限制。例如，在排程問題中，令 P_1, \dots, P_4 表示分配給四項任務的人員個數。總共分配不超過 10 人的限制記為 $atmost(10, P_1, P_2, P_3, P_4)$ 。透過檢驗當前值域中的最小值之和就能檢測出矛盾；例如，如果每個變數的值域為 $\{3, 4, 5, 6\}$ ，就不能滿足 *atmost* 的限制。我們也可以透過刪除變數值域中與其他變數的值域中的最小值不相容的最大值來保持相容性。因此，在我們的例子中如果每個變數的值域是 $\{2, 3, 4, 5, 6\}$ ，那麼 5 和 6 可以從每個變數的值域中刪去。

對於大型的整數值的資源限制問題——諸如涉及用數以百輛計交通工具來運送數以千人這樣的後勤問題——用整數集合來表示每個變數的值域，然後透過相容性檢驗方法逐步削減集合，通常是不可能的。換個方式，值域用上界和下界來表示，並透過**邊界傳播**來管理。例如，在一個航線排班問題中，假設有兩次航班 F_1 和 F_2 ，分別有 165 和 385 個座位。每次航班可承載的乘客數的初始值域為

6-10 人工智慧－現代方法 3/E

$$D_1 = [0, 165] \text{ 及 } D_2 = [0, 385]$$

現在假設我們又有一個附加限制，這兩次航班所載的總乘客數必須是 420： $F_1 + F_2 = 420$ 。傳播邊界限制，我們可以把值域縮減成

$$D_1 = [35, 165] \text{ 及 } D_2 = [255, 385]$$

如果對於每個變數 X 和它的取值的上下界，每個變數 Y 都存在某個取值滿足 X 和 Y 之間的限制，我們稱該 CSP 問題是邊界相容的。這種邊界傳播廣泛應用於實際的限制問題。

6.2.6 數獨例子

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A | | | 3 | | 2 | | 6 | | |
| B | 9 | | | 3 | | 5 | | | 1 |
| C | | | 1 | 8 | | 6 | 4 | | |
| D | | | 8 | 1 | | 2 | 9 | | |
| E | 7 | | | | | | | | 8 |
| F | | | 6 | 7 | | 8 | 2 | | |
| G | | | 2 | 6 | | 9 | 5 | | |
| H | 8 | | | 2 | | 3 | | | 9 |
| I | | | 5 | | 1 | | 3 | | |

(a)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A | 4 | 8 | 3 | 9 | 2 | 1 | 6 | 5 | 7 |
| B | 9 | 6 | 7 | 3 | 4 | 5 | 8 | 2 | 1 |
| C | 2 | 5 | 1 | 8 | 7 | 6 | 4 | 9 | 3 |
| D | 5 | 4 | 8 | 1 | 3 | 2 | 9 | 7 | 6 |
| E | 7 | 2 | 9 | 5 | 6 | 4 | 1 | 3 | 8 |
| F | 1 | 3 | 6 | 7 | 9 | 8 | 2 | 4 | 5 |
| G | 3 | 7 | 2 | 6 | 8 | 9 | 5 | 1 | 4 |
| H | 8 | 1 | 4 | 2 | 5 | 3 | 7 | 6 | 9 |
| I | 6 | 9 | 5 | 4 | 1 | 7 | 3 | 8 | 2 |

(b)

圖 6.4 (a) 一個數獨謎題；(b) 它的解答

限制滿足問題已透過受歡迎的數獨謎題引介給數百萬人了，儘管他們或許並不自知。數獨板是由 81 個方格所組成，其中某些方格剛開始時已經填入了 1 到 9 等數字。謎題是在剩下的方格中填入數字使得沒有任何數字會在同一行、同一欄，或是一個 3×3 的方框範圍內出現兩次(見圖 6.4)。一行，一欄，或是方框稱之為一個單位。

印在報紙與謎題書的數獨謎題有恰好有一解的特性。儘管有些謎題以手算的方式來求解會很棘手，可能耗時達十數分鐘，但即使最難的數獨問題 CSP 求解器也能在不到 0.1 秒的時間內解出。

一個數獨謎題被視為一個具有 81 個變數的 CSP，每個格子是一個變數。最上一行(從左至右)我們使用變數名稱 A_1 至 A_9 ，直到最後一行使用 I_1 至 I_9 。空的格子有值域 $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ 而預先填好數字的格子單一值所組成的值域。除此之外，有 27 不同的 *Alldiff* 限制：一個針對每個行，欄，與 9 格框。

$$\text{Alldiff}(A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9)$$

$$\text{Alldiff}(B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9)$$

...

$$Alldiff(A_1, B_1, C_1, D_1, E_1, F_1, G_1, H_1, I_1)$$

$$Alldiff(A_2, B_2, C_2, D_2, E_2, F_2, G_2, H_2, I_2)$$

...

$$Alldiff(A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3)$$

$$Alldiff(A_4, A_5, A_6, B_4, B_5, B_6, C_4, C_5, C_6)$$

讓我們看看邊相容性能夠如何引領我們。假設 *Alldiff* 限制已經被擴增為二元限制(如 $A_1 \neq A_2$)使得我們能直接運用 AC-3 演算法。考慮圖 6.4(a)的變數 E6——在中間框中介於 2 與 8 之間的空格。從方框的限制條件，我們不僅僅能夠從 E6 值域移除 2 與 8 也包括 1 與 7。從它的欄限制，我們能消除 5, 6, 2, 8, 9, 3。這讓 E6 留下值域 {4}；換言之，我們知道 E6 的答案。現在考慮變數 I6——那個被方框 1, 3 及 3 圍繞的底部中間的框的方格。運用邊相容性於該欄，我們消去了 5, 6, 2, 4(因為我們現在知道 E6 必須是 4), 8, 9 與 3。藉由 I5 的邊相容性我們除去 1，於值域 I6 我們僅剩下值 7。現在於欄 6 有 8 個已知值，所以邊相容性能夠推論出 A6 必須是 1。沿著這些線索持續推理，最後，AC-3 能求解整個謎題——所有的變數將它們的值域縮減為單一值，如圖 6.4(b)所顯示者。

當然，如果每一個謎題能夠以機械性的運用 AC-3 來解出數獨的話，它很快就會失去吸引力，確實 AC-3 僅僅能對最簡單的數獨謎題發揮效用。略微難的謎題能夠被 PC-2 解出，但是在一個更高昂的計算成本之下：考慮一個有 255,960 不同的路徑限制的數獨謎題。想要解出最難的謎題並且讓解題進展有效率，我們需要更聰明些。

確實，數獨謎題對於人類求解器的吸引力在於運用更複雜的推理策略時必須是資源充足的。數獨迷給了它們富有色彩的名字，如「三聯顯數格」(naked triples)該策略作用的方式如下：在任何單元中(行，欄或框)，找出三個格子，這些格子中都有一個包含相同的三個數字或是一個那些數字之子集合的值域。舉例來說，三個值域也許是 {1, 8}，{3, 8} 與 {1, 3, 8}。從那兒我們不知道哪一個格子中包含 1，3，或 8，但是我們真的知道那三個數字必然在那三個格子中。因此我們能夠自該單元中每隔一個格子之值域中拿走 1，3 與 8。

令人感興趣的是注意到我們能走多遠而不用說那是特別針對數獨而為者。我們當然必須說有 81 個變數，它們的值域是數字 1 至 9，且有 27 個 *Alldiff* 限制。除此之外，所有的策略——邊相容，路徑相容等等——將一般性地運用至所有的 CSP，而非只是數獨問題。即使三聯顯數格其實是個用於強化 *Alldiff* 限制之相容性的策略，本來就與數獨無關。這是 CSP 正規型式的威力所在：對於每個新問題值域，我們僅須以限制來定義問題；然後一般的限制求解機制就可以接手處理。

6.3 CSP 問題的回溯搜尋

數獨問題設計上是用來於限制下做推理來解題。但是許多其他 CSP 無法單單由推理來解出；總會有這麼個時候我們必須搜尋一個解答。在這一節我們看看回溯搜尋演算法如何運用於不完全賦值；在下一節我們看看局部搜尋演算法如何蘊運用於完全賦值。

我們能夠運用標準的深度限制搜尋(第 3 章)。狀態會是不完全賦值，且於賦值中某個動作會加進一個 $var = value$ 。但是對於一個具有 n 個值域大小為 d 之變數的 CSP，我們很快地注意到一件可怕的事情：頂層的分支參數就是 nd ，因為 d 個值的任何一個都可以賦給 n 個變數的任何一個。在下一層，分支參數是 $(n-1)d$ ，依此類推 n 層。我們產生了一棵有 $n! \cdot d^n$ 個葉子的搜尋樹，儘管只有 d^n 個可能的完全賦值！

我們表面上似乎很合理但是天真的形式，忽略了常見於所有的 CSP 的關鍵特性：**可交換性**。如果任何給定行動集的施用順序對結果沒有影響，那麼這個問題就是可交換的。CSP 是可交換的，因為對變數作賦值的時候，不論賦值順序為何，我們都會到達相同的不完全賦值。因此，於每個搜尋樹的節點我們僅須考慮一個變數。例如，在對澳大利亞地圖進行著色的搜尋樹的根節點，我們可能要在 $SA = red$ ， $SA = green$ 和 $SA = blue$ 之間選擇，但我們永遠不會在 $SA = red$ 和 $WA = blue$ 之間做選擇。有了這個限制，葉節點的個數如我們所希望的減少到了 d^n 個。

回溯搜尋可用於深度優先搜尋，一次為一個變數選擇值，當沒有合法的值可以再賦給該變數時就回溯。該演算法如圖 6.5。它重複地選擇一個尚未被賦予值的變數，然後依次嘗試那個變數之值域中的所有值，試圖找出一個解答。如果發覺有不相容的情事，那麼 BACKTRACK 傳回失敗的訊息，使得前一個程式呼叫去試另一個值。圖 6.6 顯示了澳大利亞問題的部分搜尋樹，其中我們按照 WA ， NT ， Q ，.....的順序來給變數賦值。因為 CSP 問題的表示是標準化的，不需要給 BACKTRACKING-搜尋提供特定值域的初始狀態、後繼函數或目標測試。

```

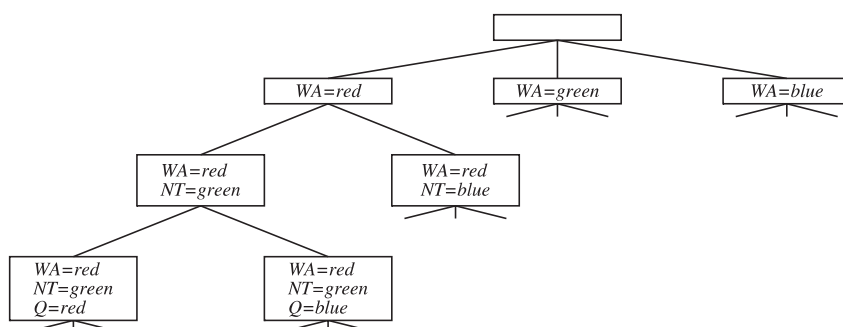
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK( $\{ \}$ , csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add  $\{ var = value \}$  to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, value)
      if inferences  $\neq$  failure then
        add inferences to assignment
        result  $\leftarrow$  BACKTRACK(assignment, csp)
        if result  $\neq$  failure then
          return result
      remove  $\{ var = value \}$  and inferences from assignment
  return failure
  
```

圖 6.5 限制滿足問題的一個簡單回溯演算法。該演算法以第三章的遞迴深度優先搜尋為模型。將函數 SELECT-UNASSIGNED-VARIABLE 和 ORDER-DOMAIN-VALUES 作變化，我們可實作課文中討論的通用啟發式演算法。函數 INFERENCE 可視需要而選擇性加上邊-、路徑-、或 k-相容性。如果所選擇的值造成失敗(由 INFERENCE 或 BACKTRACK 所通知)，則賦值(包括由 INFERENCE 所做的)會從目前的賦值中被移除，而且會嘗試一個新的值

圖 6.6

圖 6.1 中的地圖著色
問題的部分搜尋樹



請留意 BACKTRACKING-搜尋只保留一個狀態的單一表示並改變該表示而非創造新的一個，如 3.4.3 所述。

在第 3 章中，我們透過提供源自問題知識的值域特定的啟發函數，來彌補無接收資訊的搜尋演算法的不佳性能。結果顯示，我們可以不用特定值域的知識就有效地解決 CSP 問題。相反的，我們能增加一些複雜度到圖 6.5 中未指定的函數，使用它們來凸顯下述問題：

1. 下一步該對哪個變數作賦值(SELECT-UNASSIGNED-VARIABLE)，又要按什麼順序來嘗試它的值(ORDER-DOMAIN-VALUES)？
2. 於(INFERENCE)搜尋中每一個步驟該執行什麼樣的推理？
3. 當搜尋抵達某一個違反限制的賦值，該搜尋是否能避免重蹈失敗的覆轍？

接下來的幾個小節將逐一回答這些問題。

6.3.1 變數和賦值順序

回溯演算法有這麼一行：

```
var ← SELECT-UNASSIGNED-VARIABLE(csp)
```

SELECT-UNASSIGNED-VARIABLE 用的最簡單的策略是依照順序選取下一個還沒賦予值的變數， $\{X_1, X_2, \dots\}$ 。這種靜態的變數排序很少能造成高效率的搜尋。例如，在賦值 $WA = red$ 和 $NT = green$ 之後， SA 只剩下一個可能的賦值，因此下一個賦值 $SA = blue$ 要比給 Q 賦值有意義。實際上，在給 SA 賦值之後， Q ， NSW 和 V 的選擇都是強制性的了。這種直觀的想法——選擇「合法」取值最少的變數——稱為**最少剩餘值(MRV)**啟發式演算法。也稱為「最受限制變數」或「失敗優先」啟發式演算法，後者的叫法是因為它選擇了最可能很快導致失敗的變數，從而對搜尋樹剪枝。如果某些變數 X 沒有合法的值可供選用，MRV 啟發式會選擇 X 並立刻偵知失敗了——避免了無頭蒼蠅似的搜尋其他變數。MRV 啟發式通常執行的效能較隨機或靜態排序來的好，有時候達到 1,000 或更高的倍數，儘管其結果會視問題之不同而有甚大的差距。

MRV 啟發式演算法在澳大利亞問題中對選擇第一個著色區域根本沒有幫助，因為初始的時候每個區域都有三種合法的顏色。在這種情況下，**鄰接度啟發式(degree heuristic)**演算法便顯得很好用。它試圖藉由選擇牽涉到其他尚未賦值變數之限制數目為最大的變數來降低未來選擇的分支因數。在圖 6.1 中， SA 是有最高的鄰接度的變數，它的鄰接度為 5；其他變數的鄰接度為 2 或者 3，除了 T

的鄰接度為 0。實際上，一旦選擇了 SA ，應用鄰接度啟發式演算法求解問題可以不走錯任何一步——你可以在每個選擇點上選擇任何相容的顏色，仍然可以回溯就找到解。最少剩餘值啟發式演算法通常是一個更強的指導，而鄰接度啟發式演算法則對打破僵局有用。

一旦一個變數被選定，演算法就要決定檢驗它的取值的次序。為此，**最少限制值**啟發式演算法在某些情況下是有效率的。在限制圖中排除鄰居變數之可選取值為最少的值會被優先選擇。例如，假設在圖 6.1 中，我們已經製作出不完全賦值， $WA = red$ 和 $NT = green$ ，我們下一步要為 Q 選擇值。這裡藍色不是一個好的選擇，因為它消除了 Q 的鄰居 SA 的最後一個可選用的合法值。最少限制值啟發式演算法因此更願意選擇紅色而不是藍色。一般來說，啟發式演算法應該試圖給剩下的變數賦值留下最大的靈活性。當然，如果我們試圖找到某個問題的所有解，而不只是第一個解，那麼這個排序就無所謂了，因為無論如何我們要考慮每個值。當問題沒有解的時候也是一樣。

為什麼變數的選取方式該是失敗優先，而值的選取方式卻是失敗墊後？原來，對各式各樣的問題，一個選取剩餘值個數最少之變數的變數順序有助於早些修剪大部份的搜尋樹來最小化搜尋樹之節點數目。對於值的順序，技巧在於是我們只需要一個解答；因此先尋找最可能的值是合理的做法。如果我們想要列舉出所有的解答而非只是找出一個，那麼值的順序就無關緊要了。

6.3.2 交錯搜尋與推理

到目前為止我們已經看到 AC-3 與其他演算法如何在我們開始搜尋之前先推理來縮減變數的值域。但是在搜尋期間做推理甚至可能更具威力：每當我們為該變數選擇一個值，我們就有一個在該變數附近做推理新值域縮減的嶄新機會。

最簡單的推理形式被稱之為**前向檢驗**。無論何時只要變數 X 被賦予值了之後，前向檢查程序會為之建立邊相容性：對於每個透過限制而連接到 X 的尚未賦值變數 Y ，從 Y 的值域刪除任何與為 X 所選出之值不相容的值。因為前向檢驗僅僅做邊相容性推理，如果我們已經以預先處理的步驟完成了邊相容性，那麼沒有理由做前向檢驗。

圖 6.7 展示以前向檢驗做澳洲 CSP 問題的回溯搜尋。這個例子中需要注意兩個重點。第一，注意在賦值 $WA = red$ 和 $Q = green$ 之後， NT 和 SA 的值域都減小到了單個值；我們透過 WA 和 Q 傳播的資訊刪除了這些變數上的一些分支。第二點需要注意的是，當賦值 $V = blue$ 之後， SA 的值域就變成空的。因此，前向檢驗已經檢測到不完全賦值 $\{WA = red, Q = green, V = blue\}$ 與問題的限制條件是矛盾的，因此程式會立刻回溯。

如果我們結合 **MRV** 啟發式與前向檢驗，對許多的搜尋問題而言將會更有效率。考慮賦值 $\{WA = red\}$ 後的圖 6.7。直覺地，似乎那個賦值限制了它的鄰居， NT 與 SA ，所以我們應該接著處理那些變數，然後所有的其他的變數會各就其位。那正就是使用 **MRV** 所會發生的情況： NT 和 SA 有兩個值，所以兩者之一會被先選取，然後另一個，然後依序是 Q ， NSW 和 V 。最後的 T 仍然有三個值，而且任何它們三個中的一個都能發揮作用。實際上，我們可以把前向檢驗視為一個逐步增加地計算 **MRV** 啟發式演算法完成任務所需資訊的高效方法。

雖然前向檢驗能檢驗出許多矛盾，它還是不能檢驗出所有的矛盾。問題是它使得目前變數是邊相容的，但是不向前看而使得所有其他變數變得邊相容的。例如，考慮圖 6.7 中的第三行。它顯示出當 WA 是 *red*、 Q 是 *green* 的時候， NT 和 SA 都被迫是 *blue*。前向檢驗向前看得不夠遠而沒能注意到這是個不相容： NT 及 SA 為相鄰，所以不能有相同值。

該演算法稱之為 **MAC**(Maintaining Arc Consistency)偵測到此不相容。當一個變數 X_i 被賦予一個值，**INFERENCE** 程序呼叫 **AC-3**，但是與其將所有 CSP 的邊都放進一個佇列，我們僅從所有鄰近於 X_i 中尚未被賦予值的邊(X_j, X_i)開始。從那裡，**AC-3** 以尋常的方式來做限制的傳播，而且如果有任何變數的值域被縮減成空集合，對 **AC-3** 的呼叫就算失敗而我們立刻知道要回溯。我們能看得出 **MAC** 較前向檢驗更有力，因為前向檢驗剛開始時對 **MAC** 佇列中的邊做的事與 **MAC** 所做者是相同的；但是與 **MAC** 不同的是，當變數的值域出現變動時，前向檢驗並不遞迴地傳播限制。

| | WA | NT | Q | NSW | V | SA | T |
|-----------------|-------|-------|-------|-------|-------|-------|-------|
| 初始值域 | R G B | R G B | R G B | R G B | R G B | R G B | R G B |
| After $WA=red$ | (R) | G B | R G B | R G B | R G B | G B | R G B |
| After $Q=green$ | (R) | B | (G) | R B | R G B | B | R G B |
| After $V=blue$ | (R) | B | (G) | R | (B) | | R G B |

圖 6.7 使用前向檢驗方法的地圖著色搜尋的進行方式。首先賦值 $WA = red$ ；然後前向檢驗從其相鄰變數 NT 和 SA 的值域中刪除 *red*。賦值 $Q = green$ 之後，*green* 從 NT 、 SA 、 NSW 的值域中被刪除。賦值 $V = blue$ 之後，*blue* 從 NSW 和 SA 的值域中被刪除，這時 SA 已沒有合法值。

6.3.3 更聰明的回溯：向後看

圖 6.5 中的 **BACKTRACKING-SEARCH** 演算法當某個分支上的搜尋失敗時，會採取一個簡單的方針：倒退回前一個變數並且嘗試另一個值。這稱為**時序回溯**，因為重新存取的是時間最近的決策點。在這個小節，我們考慮 更好的可能性。

考慮我們在圖 6.1 的問題中按照固定的變數順序 Q, NSW, V, T, SA, WA, NT ，應用簡單回溯演算法。假設我們已經產生了不完全賦值 $\{Q = red, NSW = green, V = blue, T = red\}$ 。當我們嘗試下一個變數 SA 時，我們發現任何值都無法滿足限制。我們倒退回 T ，試著賦予 Tasmania 新的顏色！顯然這種做法很愚蠢——對 T 重著色不能解決 South Australia 的問題。

一個更聰明的回溯方法是回溯到一個或可修復此問題的變數——一個該讓 SA 可能值之一變為不可能負起責任的變數。為做到這件事，我們將持續追蹤一組與某些 SA 值相互矛盾之值。集合(在 $\{Q = red, NSW = green, V = blue\}$ 的情況)，被稱之為 SA 之**衝突集**。**後向跳躍**方法回溯到衝突集中時間最近的變數；在這種情況下，後向跳躍將跳過 T 而嘗試 V 的新值。這個方法很容易地經由修改 **BACKTRACK** 實作，使它在檢查一個合法值的同時累積衝突集。如果沒有找到合法值，那麼它利用失敗標記返回衝突集中時間最近的變數。

眼尖的讀者會注意到前向檢驗演算法可以不需要額外的工作就能提供衝突集：當基於 X 的賦值的前向檢驗從變數 Y 的值域中刪除一個值時，應該把 X 加入 Y 的衝突集裡。如果 Y 值域的最後一個值被刪除，那麼 Y 衝突集中的賦值被加入 X 的衝突集。這樣，當我們到達 Y 的時候就知道如果回溯應該回到哪個變數。

細心的讀者會注意到某些古怪之處：後向跳躍只出現在值域中的每個值都和當前的賦值有衝突的情況下；但是前向檢驗能檢測到這個事件並且一旦到達這樣的節點就阻止搜尋！實際上，可以證明每個被後向跳躍剪枝的分支在前向檢驗演算法中也被剪枝。因此，簡單的後向跳躍在前向檢驗搜尋中，或者說在諸如 MAC 這樣使用更強的相容性檢驗的搜尋中是多餘的。

除了上一段文字中的觀察結果，隱藏在後向跳躍背後的想法仍然是值得稱許的：基於失敗的原因而回溯。後向跳躍當一個變數的值域為空的時候會注意到失敗，但是在很多情況下，一個分支在這發生很久以前就已經註定要失敗了。再次考慮不完全賦值 $\{WA = red, NSW = red\}$ (從我們前面的討論中知道，它是矛盾的)。假設我們下一個嘗試 $T = red$ ，然後給 NT 、 Q 、 V 、 SA 賦值。我們知道對這最後四個變數沒有可以採行的賦值，因此最終我們用完了 NT 的所有可能取值。現在的問題是向哪裡回溯？後向跳躍是行不通的，因為 NT 確實有和前面賦值的變數相容的值—— NT 沒有由前面能導致失敗的變數所完整組成的衝突集。然而我們知道， NT 、 Q 、 V 和 SA 這四個變數之所以會失敗，是肇因於前面的一組變數，那些與這四個變數直接發生衝突的變數。這引出了關於諸如 NT 這樣一個變數的衝突集的更深概念：是那組前面變數致使 NT ，連同任何其後的變數，沒有相容解。在這種情況下，那組變數是 WA 和 NSW ，所以演算法會越過 Tasmania 回溯到 NSW 。按這種方式定義的衝突集的後向跳躍演算法稱為**衝突導向的後向跳躍**。

我們現在必須解譯這些新的衝突集是怎樣計算的。實際上方法很簡單。一個搜尋分支的「終端」失敗總是因為一個變數的值域變為空；該變數有一個標準的衝突集。在我們的例子中， SA 失敗了，它的衝突集(比如)是 $\{WA, NT, Q\}$ 。我們後向跳躍到 Q ，而 Q 將 SA 的衝突集(當然減去 Q 本身)也就是 $\{NT, NSW\}$ 吸收到自己的直接衝突集裡；新的衝突集是 $\{WA, NT, NSW\}$ 。就是說，在給定了前面對 $\{WA, NT, NSW\}$ 的賦值之後，從 Q 向前是無解的。因此我們回溯到 NT ，集合中最近的一個。 NT 把 $\{WA, NT, NSW\} - \{NT\}$ 吸收到自己的直接衝突集 $\{WA\}$ 裡，得到 $\{WA, NSW\}$ (如上一段所說的)。現在演算法如我們所希望的那樣後向跳躍到 NSW 。總結來說，令 X_j 是當前變數，再令 $conf(X_j)$ 為其的衝突集。如果 X_j 的每個可能取值都失敗了，後向跳躍到 $conf(X_j)$ 中最近的一個變數 X_i ，並置

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_j\}$$

當我們面臨矛盾的情況，後向跳躍能告訴我們該回溯多遠，所以我們不用浪費時間去更改無助於修復問題的變數。但是我們也希望避免再次地陷入相同的問題。當搜尋面臨衝突的境地時，我們知道衝突集的某些子集合對該衝突是有責任的。**限制學習**是從造成問題之衝突集中尋找一個最小的變數集合的構想。這個變數集合，加上它們的相應的值，被稱之為一個**不良**(no-good)。我們然後記錄下不良，方式是在 CSP 中加入限制或是另外保留不良值的快取資料。

舉例來說，考慮於圖 6.6 底下一行的狀態 $\{WA = red, NT = green, Q = blue\}$ 。前向檢查可以告訴我們這個狀態是一個不良，因為沒有合法的值可賦予 SA 。在這個特別的情況，記錄不良並不能幫上什麼忙，因為一旦我們從搜尋樹上剪斷這段分枝，我們將不會再次遇到這樣的組合。但是假設於圖 6.6 中的搜尋樹實際上是最先賦值予 V 與 T 的大型搜尋樹的一部份。那麼將 $\{WA = red, NT = green, Q = blue\}$ 記錄為一個不良就變得值得了，因為對每個 V 與 T 可能賦值的集合我們會再一次的碰到同樣的問題。

不良可以有效地被前向檢查或是後向跳躍所使用。限制學習是現今 CSP 求解器有效率解決複雜問題的最重要的技術之一。

6.4 CSP 問題的局部搜尋

局部搜尋演算法(參見 4.1 節)對解許多 CSP 問題都很有效的。它們使用完整的狀態形式：初始狀態給每個變數都賦予一個值，且搜尋會一次改變一個變數的取值。舉例來說，於八皇后問題(見圖 4.3)，初始狀態可能是 8 皇后於 8 行的隨機配置，且每一步會移動一個皇后到它行內的一個新位置。通常，剛開始的猜測會違反好幾個限制。區域搜尋的重點是如何剔除被違反的限制^[2]。

在為變數選擇一個新值的時候，最明顯的啟發式演算法是選擇會造成與其他變數的衝突最小的值——**最小衝突(min-conflicts)**啟發式演算法。圖 6.8 顯示了該演算法，而圖 6.9 圖示了這個演算法應用於八皇后問題時的情況。

最小衝突在許多 CSP 問題上的表現出乎意外地好。令人驚異的是，在 n 皇后問題上，如果你不計算皇后的初始放置，最小衝突演算法的執行時間大體上獨立於問題的大小。它甚至能在平均 50 步之內解決百萬皇后問題(在進行了初始賦值之後)。這個不尋常的現象激發了 20 世紀 90 年代對於局部搜尋以及容易和難的問題之間的區別的大量研究(第 7 章)。粗略地說， n 個皇后問題對於區域搜尋來說是很容易的，因為解答稠密地分佈在狀態空間各處。最小衝突對於困難的問題也表現的不錯。例如，它用於安排哈伯太空望遠鏡的觀察日程時間表，安排一周的觀察日程所花費的時間從三周(!)減少到了 10 分鐘。

所有第 4.1 節的區域搜尋技術都是應用於 CSP 的候選技術，且其中一些已經被證實非常有效。於最小衝突啟發式下 CSP 的地貌通常有好幾個綿延起伏的高原。可能有數以百萬計的變數賦值其距離解答之處只有一個衝突之遙。高原搜尋(Plateau search)——可以以相同的得分讓山肩移動到另一個狀態——有助於區域搜尋找出一條脫離高原的路。在高原上的這般游走可以用**禁忌搜尋(tabu search)**來引導：保留一份最近拜訪過之狀態的小型清單並禁止演算法回到那些狀態。模擬的退火(annealing)方法也可以被用來脫離高原區。

另一個技術，稱之為**限制加權(constraint weighting)**，有助於集中心力於搜尋重要的限制。每個限制給予一個限制加權， w_i ，剛開始時都等於 1。在每一個搜尋步驟，演算法選擇一對變數/值來更改以便讓所有違反限制的總加權值為最低。然後藉著漸漸增加每個限制的權重來調整違反目前賦值之限制的權值。這有兩個優點：它加入了高原的地形，以確保從目前狀態獲得改善是可能的，同時它也隨著時間的推移，為那些經過證明很難解出的限制加些權值。

局部搜尋的另一個優勢是當問題改變時，它可以用於線上情況。這在排程問題中尤其重要。一周的航班日程表可能涉及數以千次航班和數萬人次的分配，但是一個機場的惡劣天氣，可能就會打亂原來的日程安排。我們希望以最小的更改來修改日程。這可以用從當日程開始的局部搜尋演算法很容易地完成。一個使用新限制集的回溯搜尋通常需要更多的時間，並且找到的解有可能要對當日程進行很多更改。

```

function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
           max_steps, the number of steps allowed before giving up

  current  $\leftarrow$  an initial complete assignment for csp
  for i = 1 to max_steps do
    if current is a solution for csp then return current
    var  $\leftarrow$  a randomly chosen conflicted variable from csp.VARIABLES
    value  $\leftarrow$  the value v for var that minimizes CONFLICTS(var, v, current, csp)
    set var = value in current
  return failure

```

圖 6.8 一個用局部搜尋解決 CSP 問題的 MIN-CONFLICTS 演算法。初始狀態的選擇可採隨機方式，或藉由貪婪賦值過程依次為每個變數選擇一個最小衝突的值而完成。函數 CONFLICTS 統計一個特定值破壞限制的數量，在已知當前賦值的剩餘值的情況下

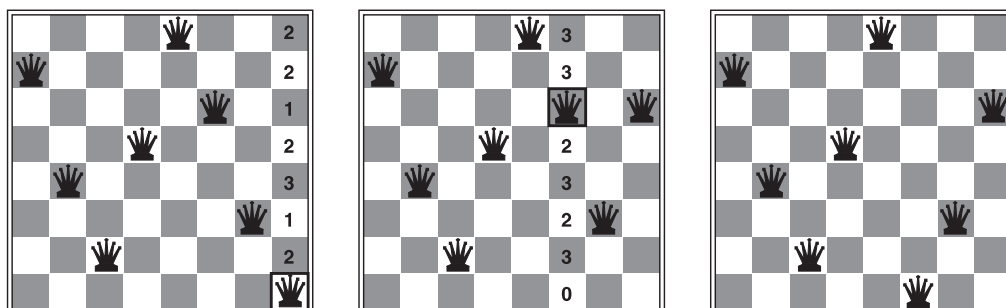


圖 6.9 用最小衝突演算法解決八皇后問題的一個兩步解。每一步選擇一個皇后，在它所在列中重新分配位置。衝突的個數(在這個問題中是能攻擊到的皇后的個數)在每個方格裡列出來。演算法將皇后移到最小衝突的方格裡，最小衝突值有多個方格時則隨機地選取其中之一

6.5 問題的結構

在這一節中，我們考慮以什麼方式利用問題的結構，如限制圖所表示的，能快速地找到解。大多數這裡所提的方法，除了 CSP 類型的問題之外，也適用於其他的問題，如機率性推理。畢竟我們處理實際世界問題時，有希望的唯一辦法就是將它分解為很多子問題。再看一次澳洲地圖的限制圖(圖 6.1b，重複於圖 6.12a)，凸顯出一種事實：Tasmania 和大陸不相連^[9]。直觀上，顯然對 Tasmania 著色和對大陸著色是獨立的子問題——任何對大陸區域著色的解和任何對 Tasmania 著色的解合併起來，都得到整個地圖的一個解。獨立性可以簡單地透過在限制圖中尋找連通域來確定。每個連通域

對應於一個子問題 CSP_i 。如果賦值 S_i 是 CSP_i 的一個解，那麼 $\cup_i S_i$ 是 $\cup_i CSP_i$ 的一個解。為什麼這是很重要的？考慮以下問題：假設每個 CSP_i 有總共 n 個變數中的 c 個變數， c 是一個常數。那麼就有 n/c 個子問題，解決每個子問題最多花費 d^c 步工作，其中 d 為域的大小。因此總的工作量是 $O(d^c n/c)$ ，是 n 的線性函數；而不進行分解的話，總的工作量是 $O(d^n)$ ，是 n 的指數函數。我們給一個更具體的例子：將一個 $n = 80$ 的布林 CSP 問題分解成 4 個 $c = 20$ 的子問題，會使最壞情況下的時間複雜度，從宇宙壽命那麼長的時間減少到一秒以內。

完全獨立的子問題很誘人，但是很少見。幸運地，某些其他圖形架構也很容易解出。舉例來說，當任兩個變數僅僅由一條路徑所連通的時候，一個限制圖形是一棵樹。我們將證明任何一個樹狀結構的 CSP 問題可以在變數個數的線性時間內求解^[4]。關鍵之處是一個新的相容概念，稱之為**有向的邊相容性**或 DAC。一個 CSP 在變數順序 X_1, X_2, \dots, X_n 下被定義為是邊相容的，若且唯若各個 X_i 於 $j > i$ 下與每一個 X_j 是邊相容的。

要求解一個樹狀結構的 CSP，首先挑選任一個變數當做樹的根節點，並選取一個變數的順序使得每個變數出現在樹的母節點之後。這樣的排序被稱之為**拓樸排序**。圖 6.10(a)展示一棵範例樹，且 (b)顯示一個可能的排序。任何具有 n 個節點的樹有 $n - 1$ 個邊，因此我們能夠使得這個圖形於 $O(n)$ 個步驟內使之變為具有有向邊相容性，對於兩個變數而言每個步驟必須比較的可能的值域值多達 d 個，總時間為 $O(nd^2)$ 。一旦我們有了有向邊相容的圖，我們只要逐一查看變數表並選取任何的剩餘值。因為每個從母節點到子節點的連結是邊相容的，我們知道為任何我們選來當作母節點的值，將會有一個正確值留用來選取為子節點。這意指我們不必回溯；我們能線性地移動該變數。完整的演算法如圖 6.11 所示。

現在我們已經有了一個對樹狀結構的高效演算法，我們可以考慮更一般的限制圖是否能簡化成樹的形式。有兩種基本的方式：一種基於刪除節點，另一種基於合併節點。

第一種方法涉及先對其中的某些變數賦值，使剩下的變數能夠形成一棵樹。考慮澳大利亞的限制圖，如圖 6.12(a)所示。如果我們能刪除 South Australia，這個圖就會變影像(b)中的一棵樹。幸運的是，我們可以做到這個(只是在圖中刪除，而不是真的從大陸上刪除)，透過給變數 SA 固定的值並且從其他變數的值域中刪除任何與為 SA 選中的值不相容的值。

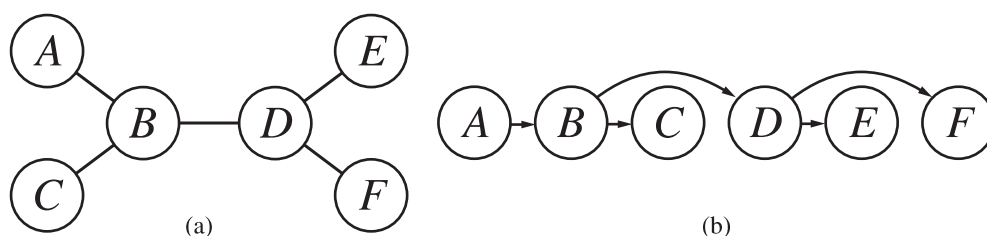


圖 6.10 (a) 樹狀結構 CSP 的限制圖。(b) 與以 A 為根節點的樹相容的變數的線性排序。這稱為該變數的拓樸排序

```

function TREE-CSP-SOLVER(csp) returns a solution, or failure
  inputs: csp, a CSP with components  $X, D, C$ 

   $n \leftarrow$  number of variables in  $X$ 
  assignment  $\leftarrow$  an empty assignment
  root  $\leftarrow$  any variable in  $X$ 
   $X \leftarrow \text{TOPOLOGICALSORT}(X, \text{root})$ 
  for  $j = n$  down to 2 do
    MAKE-ARC-CONSISTENT(PARENT( $X_j$ ),  $X_j$ )
    if it cannot be made consistent then return failure
  for  $i = 1$  to  $n$  do
    assignment[ $X_i$ ]  $\leftarrow$  any consistent value from  $D_i$ 
    if there is no consistent value then return failure
  return assignment

```

圖 6.11 TREE-CSP-SOLVER 演算法用於求解樹狀架構 CSP。如果該 CSP 有一個解答，我們會於線性時間內找到它；如果沒有，我們會找出矛盾

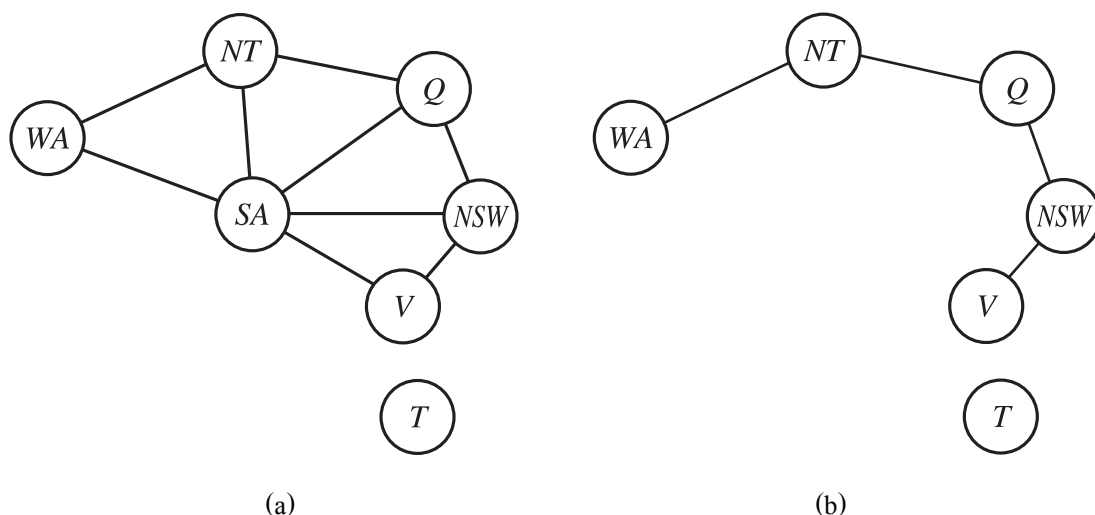


圖 6.12 (a) 圖 6.1 的原始限制圖。(b) 刪除 SA 之後的限制圖

現在，在刪除了 SA 和它的限制之後，CSP 問題的每個解都將與為 SA 選擇的值是相容的。(這對二元 CSP 問題是可行的；在高層次的限制問題中情況會更複雜)。因此，我們可以用前面給出的演算法求解剩餘的樹，並由此得到整個問題的解。當然，在一般情況下(與地圖著色相反)為 SA 選擇的值可能是錯誤的，因此我們將需要嘗試所有的值。一般演算法如下：

1. 從 CSP 變數中選擇一個子集 S ，使得限制圖在刪除 S 之後成為一棵樹。 S 稱為環割集。
2. 對於每個賦予 S 中滿足所有施加於 S 限制之變數的值
 - (a) 自剩餘數值域移走任何不相容於賦予 S 之值的值，且
 - (b) 如果其餘的 CSP 有一個解，將它隨同 S 之賦值傳回。

如果環割集的大小為 c ，那麼總執行時間為 $O(d^c \cdot (n-c)d^2)$ ：我們必須試試每個 S 中變數的 d^c 個值的組合，且對每個組合我們必須解一個樹的大小為 $n-c$ 的問題。如果限制圖「近似於一棵樹」，那麼 c 將會很小，並且比直接回溯將有很大的節省。然而在最壞情況下， c 可能大到 $(n-2)$ 。找到最小的環割集是一個 NP-hard，但已有若干高效率的近似演算法。總體演算法叫做**割集調整**；我們將在第十四章再次見到，在那裡用來進行機率推理。

第二個方法是建立在構造限制圖的**樹分解**的基礎上的，它把限制圖分解為相關聯的子問題集。每個子問題將會獨立地求解，再把得到的結果合併起來。像多數分治演算法一樣，如果沒有一個子問題特別大，它的效果就會很好。圖 6.13 給出了地圖著色問題的樹分解，形成 5 個子問題。一個樹分解必須滿足以下 3 個條件：

- 每個原始問題中的變數至少在一個子問題中出現。
- 如果兩個變數在原問題中由一個限制相連，那麼它們至少同時出現在同一個子問題中(連同它們的限制)。
- 如果一個變數出現在樹中的兩個子問題中，它必須出現在連接這些子問題的路徑上的所有子問題裡。

前兩個條件保證了所有的變數和限制都在分解中表示出來。第三個條件看起來更技術性，但是它只是反映了任何給定的變數在每個子問題中必須取值相同的限制；子問題之間的連接強制了這個限制。例如，在圖 6.13 中 SA 出現在連接起來的所有 4 個子問題中。你可以從圖 6.12 驗證這種分解是有意義的。

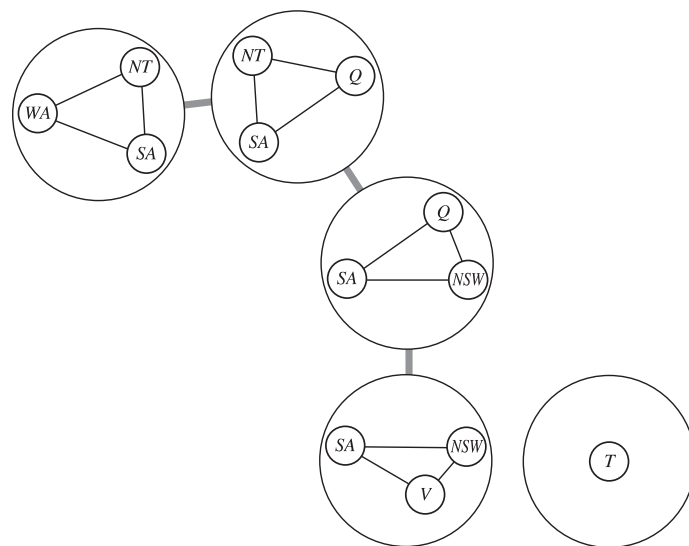


圖 6.13 圖 6.12(a)中限制圖的樹分解

我們獨立地求解每個子問題；如果其中任何一個無解，那麼原始問題就無解。如果我們能求解所有的子問題，接下來我們就要嘗試構造一個全局解。首先，我們把每個子問題視為一個「大變數」，它的值域是這個子問題的所有解的集合。例如，圖 6.13 中最左邊的子問題是一個有 3 個變數的地圖著色問題，因此有 6 個解——其中一個是 $\{WA = red, SA = blue, NT = green\}$ 。然後我們可以用前面給出的對樹的有效演算法來求解連接這些子問題的限制問題。對於子問題之間的限制僅僅需要保持它們的共用變數的相容性。例如，對第一個子問題給定的解 $\{WA = red, SA = blue, NT = green\}$ ，下一個子問題的相容解只能是 $\{SA = blue, NT = green, Q = red\}$ 。

一個給定的限制圖允許有多種樹分解；在選擇分解的時候，目標是分解出來的子問題越小越好。圖的樹分解的樹寬比最大的子問題的大小少 1；圖本身的樹寬定義為它所有樹分解的最小樹寬。如果一個圖的樹寬為 w ，並且給定相對應的樹分解，那麼這個問題的時間複雜度是 $O(nd^{w+1})$ 。因此，限制圖樹寬有界的 CSP 問題是多項式時間內可解的。不幸的是，找到最小樹寬的樹分解是一個 NP 難題，不過實際上有一些很可行的啟發式方法。

到現在為止，我們已經查看過限制圖的架構。於變數值也可能有重要的架構。考慮 n 個顏色的地圖著色問題。對每一個相容的解答，實際上有一個經由排列顏色名稱所構成的 $n!$ 解答集合。舉例來說，在澳洲地圖我們知道 WA ， NT ，與 SA 必須全都有不同的顏色，但是有 $3! = 6$ 方法賦予三個顏色於這三個區域。這稱之為為值的對稱。我們想要藉由打破對稱來縮減搜尋空間達 $n!$ 倍。我們藉由引進打破對稱的限制來達成目的。就我們的例子而言，我們可能會施加一個任意排序限制， $NT < SA < WA$ ，該限制要求這三個值須依字母順序來排列。這個限制確保 $n!$ 個解答中只有一個是可能的： $\{NT = blue, SA = green, WA = red\}$ 。

就地圖著色而言，找出除去對稱的限制很容易，且一般而言可能在多項式量級的時間內找出能除去所有而只有一個是對稱之解答的限制，但是於搜尋期間，要除去暫時值之集合的所有對稱則屬於一種 NP-hard 的問題。實務上，在許多問題上，打破值的對稱性業經證明是很重要的且有效的。

6.6 總結

- **限制滿足問題(CSP)**以變數/值對的集合來表示狀態並以施加於該變數之限制的集合來表示解答的條件。許多重要的現實世界問題都可以描述為 CSP 問題。
- 有些推理技術使用限制來推論哪些變數/值對是相容的且哪些是不相容的。這些包括節點，邊，路徑，與 k -相容。
- **回溯搜尋**，深度優先搜尋的一種形式，經常用於求解 CSP 問題。推理可以與搜尋交錯運用。
- **最少剩餘值和度啟發式**演算法是在回溯搜尋中決定下一步選擇哪個變數的兩個獨立於值域的方法。對於一個已給定的變數，**最少限制值(least-constraining-value)**啟發式有助於決定該優先試哪一個值。回溯發生在對一個變數找不到合法賦值的時候。**衝突導向的後向跳躍**，直接回溯到問題的源頭。
- 使用**最小衝突**啟發式演算法的局部搜尋，在求解限制滿足問題方面，已經很成功。

- 求解 CSP 問題的複雜度大部份上取決於限制圖的結構。樹狀結構的問題可以在線性時間內求解。**割集調整**可以把一般的 CSP 問題簡化為樹狀結構，並且在能找到比較小的割集的情況下，十分有效。**樹分解**技術把 CSP 問題轉變為子問題的樹，並且當限制圖的**樹寬**很小的時候是很有有效的。

◎ 參考文獻與歷史的註釋 BIBLIOGRAPHICAL AND HISTORICAL NOTES

限制滿足問題有關的最早工作大部分是處理數值限制。整數域內方程式的限制最早在 7 世紀由印度數學家 Brahmagupta 研究過；在希臘數學家戴奧弗多斯(Diophantus, 約 200-284)之後這類問題常稱為**戴奧弗多斯方程式**(Diophantine equations)，他實際是在正有理數域內考慮了這個問題。透過消元法解線性方程的系統方法由高斯(Gauss, 1829)研究過；對線性不等式限制的求解方法可追溯到傅立葉(Fourier, 1827)。

有限值域的限制滿足問題也有很長的歷史。例如，**圖著色問題**(地圖著色是其中的一種特殊情況)是數學史上的一個古老問題。四色問題假說(每一平面圖都可用四種或更少的顏色著色)是 1852 年由狄摩根(de Morgan)的一個學生 Francis Guthrie 首次提出。它抗拒解答——儘管數個已刊載的文章，是做相反的聲言——直到 Appel 與 Haken(1977)提供了證明[請參閱 *Four Colors Suffice*(Wilson, 2004)一書]。純粹理論者對於其中一部份證明須仰賴電腦來證明感到失望，所以 Georges Gonthier(2008)，使用 COQ 理論證明器，推導出一個正式的證明展示 Appel 與 Haken 的證明是正確的。

特定類型的限制滿足問題的出現，貫穿於電腦科學史。其中一個最有影響力的早期例子是 SKETCHPAD 系統(Sutherland, 1963)，它解決了圖的幾何限制問題並且是現代繪圖程式和 CAD 工具的先驅。把 CSP 當作一般性的問題要歸功於 Ugo Montanari(1974)。將高階 CSP 問題透過輔助變數簡化為純粹的二元 CSP 問題(參見習題 6.5)，初源自 19 世紀的邏輯學家 Charles Sanders Peirce。Dechter(1990b)把它納入 CSP 文獻，並由 Bacchus 和 van Beek(1998)作了詳細描述。對解有偏好的 CSP 問題在最佳化的文獻中有廣泛的研究；參見 Bistarelli 等人(1997)對允許偏好的 CSP 架構的一般化。桶消除演算法(Dechter, 1999)也可以用於問題最佳化。

限制傳播方法的普及歸功於 Waltz(1975)在電腦視覺方面的多面體線性標注問題上的成功。Waltz 證明了在許多問題中，限制的傳播能完全消除回溯的需要。Montanari(1974)引入了限制網路的概念和透過路徑相容性進行的限制傳播。Alan Mackworth(1977)提出了 AC-3 演算法，實作了增強的邊相容以及把回溯與某種程度的強制相容結合起來的一般想法。更有效的邊相容演算法 AC-4 是由 Mohr 和 Henderson(1986)發展出來的。在 Mackworth 的論文出現之後不久，研究者們開始實驗研究，強制相容的代價與由搜尋減少帶來的好處，兩者之間的折衷。Haralick 和 Elliot(1980)證實了 McGregor(1979)描述的最小前向檢驗演算法，而 Gaschnig(1979)提出了在每個變數賦值之後的完全邊相容檢驗演算法——後來被 Sabin 和 Freuder(1994)稱為 MAC 的演算法。後者的論文提供了令人信服的證據，說明在更難的 CSP 問題上完全邊相容演算法是成功的。Freuder(1978, 1982)研究了 k 相容的概念，以及它與求解 CSP 問題的複雜度之間的關係。Apt(1999)描述了可以用來分析相容性傳播演算法的一般性演算法架構，而 Bessière(2006)提出當前的探究成果。

處理高階或全局的限制問題的特殊方法，最早是在**限制邏輯程式設計**的脈絡中發展出來的。Marriott 和 Stuckey(1998)提供了該值域的出色的綜述。*Alldiff* 限制由 Regin(1994)，Stergiou 與 Walsh(1999)，以及 van Hoes(2001)所研究。Van Hentenryck 等人(1998)把邊界限制問題與限制邏輯程式設計結合起來。有關全局限制方面的探討由 van Hoes 與 Katriel(2006)所提供。

數獨已變成最為熟知的 CSP 且被 Simonis(2005)做如是的介紹。Agerbeck 與 Hansen(2008)描述一些策略並於一個 $n^2 \times n^2$ 板展示數獨屬於一種 NP-hard 問題。Reeson 等人(2007)展示一個建立在 CSP 技術的互動求解器。

回溯搜尋的構想可以追溯到 Golomb 與 Baumert(1965)，且它的應用於限制滿足問題則歸功於 Bitner 與 Reingold(1975)，儘管它們的基本演算法可回溯到 19 世紀。Bitner 和 Reingold 也引入了 MRV 啟發式演算法，他們稱之為最強限制變數啟發式演算法。Brelaz(1979)用度啟發式演算法，作為 MRV 啟發式演算法的一個打破僵局的工具。這樣的演算法雖然簡單，但卻是任意圖 k 著色問題的最佳方法。Haralick 和 Elliot(1980)提出了最少限制值啟發式演算法。

最基本的後向跳躍方法是由 John Gaschnig(1977, 1979)提出來的。Kondrak 和 van Beek(1997)證明了這個演算法本質上包含在前向檢驗演算法中。衝突指導的後向跳躍演算法是 Prosser(1993)發明的。最一般也是最有效形式的聰明後向跳躍演算法其實很早就由 Stallman 和 Sussman(1977)提出來了。他們的**依賴性指導的回溯技術**，引導了**真值維護系統**的發展(Doyle, 1979)，我們將在第 12.6.2 節中討論。Kleer(1989)分析了這兩個值域之間的聯繫。

Stallman 和 Sussman 的工作還引入了**限制學習**的想法，透過搜尋得到的部分結果可以保存下來，並在後面的搜尋中重新使用。該構想由 Dechter(1990a)予以正式化。**後向標記**(Gaschnig, 1979)是一個特別簡單的方法，其中相容和不相容的成對賦值被保存下來，並且用來避免重複檢驗限制。後向標記可以和衝突指導的後向跳躍演算法結合使用；Kondrak 和 van Beek(1977)提出了一個包含這兩種演算法的混合演算法。**動態回溯方法**(Ginsberg, 1993)保留了後面變數子集的成功的不完全賦值，當回溯略過早期選擇時，不會破壞後來的成功。

數個隨機化回溯方法的實驗性研究由 Gomes 等人(2000)，以及 Gomes 與 Selman(2001)所完成。Van Beek(2006)探討了回溯。

局部搜尋演算法在限制滿足問題中普及是由於 Kirkpatrick 等人(1983)在模擬退火方面的研究(參見第四章)，它在排程問題中得到了廣泛應用。最小衝突啟發式演算法首先由 Gu(1989)提出來以及由 Minton 等人(1992)獨立發展出來。Sosic 和 Gu(1994)說明了如何用之在 1 分鐘之內求解 3 百萬皇后問題。在 n 皇后問題上應用最小衝突啟發的局部搜尋演算法上令人震驚的成功，使人們重新考慮「容易」和「難」的問題的本質和通性。Peter Cheeseman 等人(1991)探索了隨機產生的 CSP 問題的難度，並發現幾乎所有這樣的問題不是極其簡單，就是就無解。只有當問題產生器的參數在一個特定的很窄的區域裡，產生的問題才有大約一半可解，而我們能找到「難」題的實例。我們將在第七章進一步討論這個現象。Konolige(1994)展示在具有某種程度的區域架構之問題的區域搜尋是不如回溯搜尋的；這引領出結合了區域搜尋與推理的成果，如 Pinkas 與 Dechter(1995)者。Hoos 與 Tsang(2006)探討區域搜尋技術。

CSP 問題相關的結構和複雜性研究，起源於 Freuder(1985)，他顯示在邊相容樹上無回溯的搜尋是可行的。用無環超圖的擴展得到了一個類似的結果(Beeri 等人，1983)。Bayardo 與 Miranker(1994)展示了針對樹狀架構之 CSP 的演算法，它在沒有任何預處理的程序下，執行時間會是線性的。

由於這些論文的發表，在求解 CSP 問題的複雜性和該問題限制圖結構之間的聯繫上，有了許多更廣泛的研究進展。圖論家 Robertson 和 Seymour(1986)引入了樹寬的概念。Dechter 和 Pearl(1987, 1989)在 Freuder 的研究基礎上，把同樣的概念應用於限制滿足問題(他們稱之為**歸納寬度**)，同時展出第 6.5 節中描述的樹分解方法。利用這些知識和資料庫理論的結果，Gottlob 等人(1999a, 1999b)提出了超樹寬的概念，它建立在把 CSP 問題刻畫為一個超圖的基礎上。除證明了任何超樹寬為 w 的 CSP 問題可以在 $O(n^{w+1} \log n)$ 時間內求解以外，他們還證明了在超樹寬有界而其他度量無界的情況下，超樹寬包含了所有先前定義的「寬度」度量。

對於將回看(look-back)方法運用於回溯的興趣被 Bayardo 與 Schrag(1997)的成果所重新點燃，他的 RELSAT 演算法結合了限制學習與後向跳躍，並且展示了時間上優於許多其他演算法。這引領出可同時應用於 CSP 與機率性推理的 AND/OR 搜尋演算法(Dechter 與 Mateescu, 2007)。Brown 等人(1988)引進打破 CSP 中對稱的構想，而 Gent 等人(2006)提供了最近的探討成果。

分散式限制滿足領域探討的是，當有一群代理人，每個人控制一部分限制變數，如何求解 CSP。從 2000 年起關於這個問題已經舉行了年度講習，以及好文章(Collin 等人，1999；Pearce 等人，2008；Shoham 與 Leyton-Brown，2009)。

CSP 演算法的多屬於實驗性的科學：很少有理論的結果能證明在所有的問題上，某個演算法會優於另一個演算法；相反的，我們必須實驗看看於典型問題的實例上哪一種演算法表現的比較好。如同 Hooker(1995)指出的，我們必須小心的辨別各個爭出鋒頭的測試之間有何不同——如同出現在演算法以執行時間來互爭高下——以及科學性的測試，它的目標是辨認出的演算法特性，此特性可用於決定在某類型問題上它的效率。

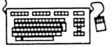
由 Apt(2003)與 Dechter(2003)新近出版的教科書，以及 Rossi 等人(2006)的彙編都是限制處理的極佳資源。關於 CSP 技術有很多好的介紹，包括 Kumar(1992)，Dechter 和 Frost(2002)，Bartak(2001)的；還有由 Dechter(1992)和 Mackworth(1992)寫的百科全書式的文章。Pearson 和 Jeavons(1997)綜述了 CSP 中可操作的問題，涵蓋了結構分解方法和依賴於值域或限制本身屬性的方法。Kondrak 和 van Beek(1997)給出了回溯搜尋演算法的一個分析性綜述，而 Bacchus 和 van Run(1995)給出了一個更加經驗化的概述。限制程式的設計涵蓋於 Apt(2003)與 Fruhwirth 與 Abdennadher(2003)所寫的書。在 Freuder 和 Mackworth(1994)編輯的論文集中，描述了 CSP 問題的幾個有趣的應用。關於限制滿足問題的論文通常出現在《人工智慧》(*Artificial Intelligence*)和專業期刊《限制》(*Constraints*)上。這一值域主要的會議是「限制規劃理論與實踐國際會議」(International Conference on Principles and Practice of Constraint Programming)，通常簡稱為 CP。


❖ 習題 EXERCISES

- 6.1 圖 6.1 所示的地圖著色問題一共有多少個解？如果可允許四種顏色，則會有多少解答？兩種顏色呢？
- 6.2 考慮將 k 個騎士放到一個 $n \times n$ 棋盤使得任兩個騎士不會互相攻擊的問題，其中 k 已知且 $k \leq n^2$ 。
- 6.3 考慮一個構造(不是求解)縱橫字謎的問題^[5]：將詞填入矩形的格子中。作為問題的一個部分，矩形格子指定了哪些方格是空的，哪些是有陰影的。假設給定了一個單詞清單(也就是一個詞典)，任務是用單詞清單的任何子集來填充空格。用以下兩種方法精確地形式化這個問題：
- 形式化為一般的搜尋問題。選取一個適合的搜尋演算法併指明一個啟發式函數。每次將一個字母填入空格會是比较好的做法或是每次一個字？
 - 形式化為一個限制滿足問題。問題的變數應該是字母還是單詞？
- 你認為這兩種形式化的方法哪種更好？為何？
- 6.4 對以下兩個問題給出精確的限制滿足問題的形式化：
- 直線性的樓面規劃：在一個大的矩形裡找到不重疊位置供許多小矩形放置。
 - 授課日程安排：給定了固定數量的教授和教室，一個可提供課程的清單，以及可能安排課程的時間段清單。每個教授有他(或她)能教的課程列表。
 - 漢密頓周遊：給予一個由馬路連接所形成的城市網路，選擇不會重複拜訪某個國家中的所有城市的順序。
- 6.5 分別用回溯演算法、前向檢驗演算法、MRV 和最少限制值啟發式演算法手工求解圖 6.2 中的密碼算術問題。
- 6.6 說明如何透過使用一個輔助變數把一個諸如「 $A + B = C$ 」這樣的三元限制問題轉變成三個二元限制。可以假設值域是有限的。(提示：考慮一個取自那些值與其它值成對的新變數，並考慮限制如「 X 是 Y 對的第一個元素。」)接著，展示三個以上變數的限制如何可以相似的方法來處理。最後，說明如何透過改變變數的值域來消除一元限制。這就完整展示了任何 CSP 問題都可以轉變為只使用二元限制的 CSP。
- 6.7 考慮下述的邏輯問題：有 5 所不同顏色的房子，住著 5 個來自不同國家的人，每個人都喜歡一種不同牌子的香煙、不同牌子的飲料和不同的寵物。給定下列已知條件，請回答問題「斑馬住在哪兒？以及哪所房子裡的人喜歡喝水？」
- 英國人住在紅色的房子裡。
 - 西班牙人養的是狗。
 - 挪威人住在最左邊的第一所房子裡。
 - 綠色屋子就在象牙屋的右邊。
 - 喜歡抽 Chesterfields 牌香煙的人住在養狐狸的人的旁邊。
 - Kit Kat 在黃色房子中被吃掉。
 - 挪威人住在藍色房子旁邊。
 - 吃 Smarties 的人擁有蝸牛。

- 吃 Snickers 的人喝柳橙汁。
- 烏克蘭人喜歡喝茶。
- 日本人吃 Milky Way。
- Kit Kat 緊挨著麥養馬匹房子的房子中被吃掉。
- 住綠色屋子的人喜歡喝咖啡。
- 住在中間房子裡的人喜歡喝牛奶。

討論把這個問題表示成 CSP 問題的不同表示法。你認為哪種比較好，為什麼？

- 6.8 考慮具有 8 個節點的圖形 $A_1, A_2, A_3, A_4, H, T, F_1, F_2$ 。對所有的 i ， A_i 連接至 A_{i+1} ，每個 A_i 連接至 H ， H 連接至 T ，而 T 連接至每個 F_i 。使用下述策略以手算的方式找出這張圖的三個顏色：以衝突導向的後向跳躍做回溯，變數的順序是 $A_1, H, A_4, F_1, A_2, F_2, A_3, T$ ，與值的順序是 R, G, B 。
- 6.9 解釋為什麼在 CSP 搜尋中，一個好的啟發式演算法選擇變數的時候應該選擇限制最多的變數，而選擇值的時候應該選擇造成限制最少的。
- 6.10  產生地圖著色問題的隨機實例如下：將 n 個點分散到單位正方形上；隨機地選擇點 X ，將 X 與距離最近的點 Y 用直線段連接起來，之前 Y 必須沒有和 X 相連，它們的連線也不能跨越已有的連線；重複前面的過程直到不再有可能的連接。點代表地圖上的區域且直線連接鄰近的區域。現在對 $k = 3$ 與 $k = 4$ 試著找出每張地圖的 k 個顏色，使用最小衝突，回溯，具前向檢驗之回溯，具 MAC 之回溯。針對每個演算法，建立一張平均執行時間表格，數值從 n 到你所能掌握的最大值。評論你的結果。
- 6.11 用 AC-3 演算法說明邊相容對圖 6.1 中所示問題能夠檢測出不完全賦值 $\{WA = red, V = blue\}$ 的不相容。
- 6.12 在樹狀結構 CSP 問題的最壞情況下執行 AC-3 演算法的複雜度是多少？
- 6.13 AC-3 演算法在從 X_i 的值域中刪除任何值時，都把每條邊 (X_k, X_i) 放回到佇列裡，即使 X_k 中的每個值都和 X_i 的一些剩餘值相容。假設對每條邊 (X_k, X_i) ，我們記錄 X_i 與 X_k 中的每個值都相容的剩餘值的個數。解釋如何有效地更新這些數字，並因此使邊相容演算法時間複雜度為 $O(n^2 d^2)$ 。
- 6.14 TREE-CSP-SOLVER(圖 6.10)從葉節點開始讓邊相容並反方向朝根節點進行。它為什麼會這樣做？如果它朝相反的方向走，會發生什麼後果？
- 6.15 我們引進數獨並視之為一個準備透過搜尋一部份的賦值來予以解出的 CSP，因為這是一般人求解數獨問題的方式。當然，也有可能以區域搜尋全部賦值的方式來求解這些問題。一個使用最小衝突啟發式的區域求解器在數獨問題上會表現的多好？
- 6.16 以你自己的話來定義限制，可交換性，邊相容，後向跳躍，最小衝突，與環割集。

- 6.17  假設已知一含有不超過 k 個節點的環割集的圖。描述一個尋找最小環割集的簡單演算法，它的執行時間在 n 個變數的 CSP 問題中不超過 $O(n^k)$ 。在參考文獻中查閱在割集大小的多項式時間內能完成尋找近似最小環割集的方法。現有的演算法能使環割集方法實用嗎？

本章註腳

- [1] AC-4 演算法(Mohr 與 Henderson, 1986)最壞情況下的執行時間是 $O(cd^2)$ 但是在平均情況下可能會略慢於 AC-3。(參見習題 6.12)。
- [2] 局部搜索很容易擴展限制最佳化問題(COP)。如果那樣的話，爬山法和模擬退火的所有技術都可以用來最優化目標函數。
- [3] 一個細心的繪圖者或 Tasmanian 的熱愛者可能會反對把 Tasmania 塗上和離它最近的大陸鄰域相同的顏色，以避免別人認為 Tasmanian 可能是那個州的一部分。
- [4] 令人難過的是,世界上有非常少的區域具有樹狀架構化的地圖，儘管 Sulawesi 已是非常接近了。
- [5] Ginsberg 等人(1990)討論過幾種構造縱橫字謎的方法。Littman 等人(1999)討論了更難的問題，如何求解縱橫字謎問題。