第三章 子代例子之篩檢

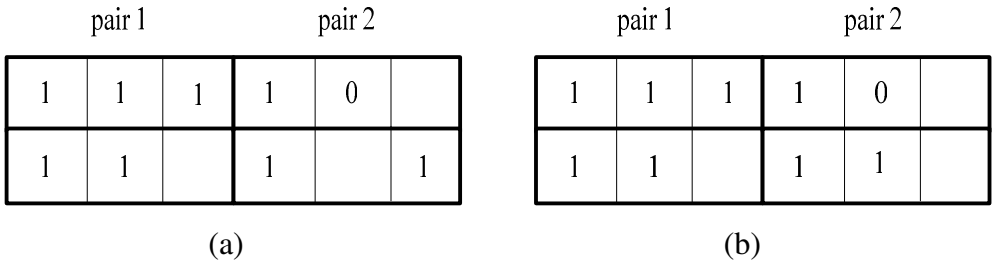
第一節 子代例子合法解篩選

套用「基因重整繁衍法」產生出子代候選例子後，還必須對這些例子作「是否具合法性」之篩檢，使之成為有效的 PCP 例子。為了要做到合法性的判斷，首先須要在繁衍子代時，由於 string 是由提示基因與空白基因組合而成，故判斷子代的每一組 pair 的上下 string 長度中的空白基因個數是否屬於合法，例如：size=2，width=3，正常來說要有兩組 pair，且每組 pair 的上下 string 的空白基因個數應該要介於 0 到(width-1)之間，假若繁衍出的 pair 中的 string 之空白基因的個數等於 width 長度時，則必須被篩檢出來，底下以圖十三說明之：

pair 1			pair 2		
1	1	1	1	0	
1	1				

圖十三：不合法例子

pair 2 中的 downstring 的空白基因個數不符合規定，其個數必須小於 3。或是當有底下圖十四(a)的情況發生時，亦不具合法性：



(a) (b)

圖十四：合法性的排序重整前後

當圖十四(a)中的 PCP[2,3]例子之 pair 2 的 downstring 的提示基因並無連續時，會造成 Solver 判讀為 PCP[3,3]，因為 Solver 是以空白當做是 pair 的區隔，故須要將圖十四(a)的例子先轉換為圖十四(b)的例子，也就是 string 是由數個提示基因接連後再加上零個或數個空白基因所組合。為了要篩檢出子代的 size 與 width 是否具有合法性，在本研究中必須做「合法性的排序重整」。也就是將每一組 pair 的上下 string 中，若存在有提示基因不連續時，則必須重新合併使其連續，將空白基因排到提示基因的右邊，如圖十四(b)為重整後的結果。

接著便還要再多判斷每一個重新合併後的子代例子，其第一組 pair 的 upstring 是否滿足符合皆為提示基因的條件，以及其它 pair 中的 string 空白基因個數是否滿足該特定 size 與 width 集合所應該具備的條件。

第二節 子代例子重覆性篩檢

當基因繁衍完之後會產生許多候選子代，這些候選子代經由合法性篩選完後，可能會有重覆的例子存在，則需將其篩檢過濾，否則送到 Solver 時，便會出現重覆解題情況發生，浪費許多時間。

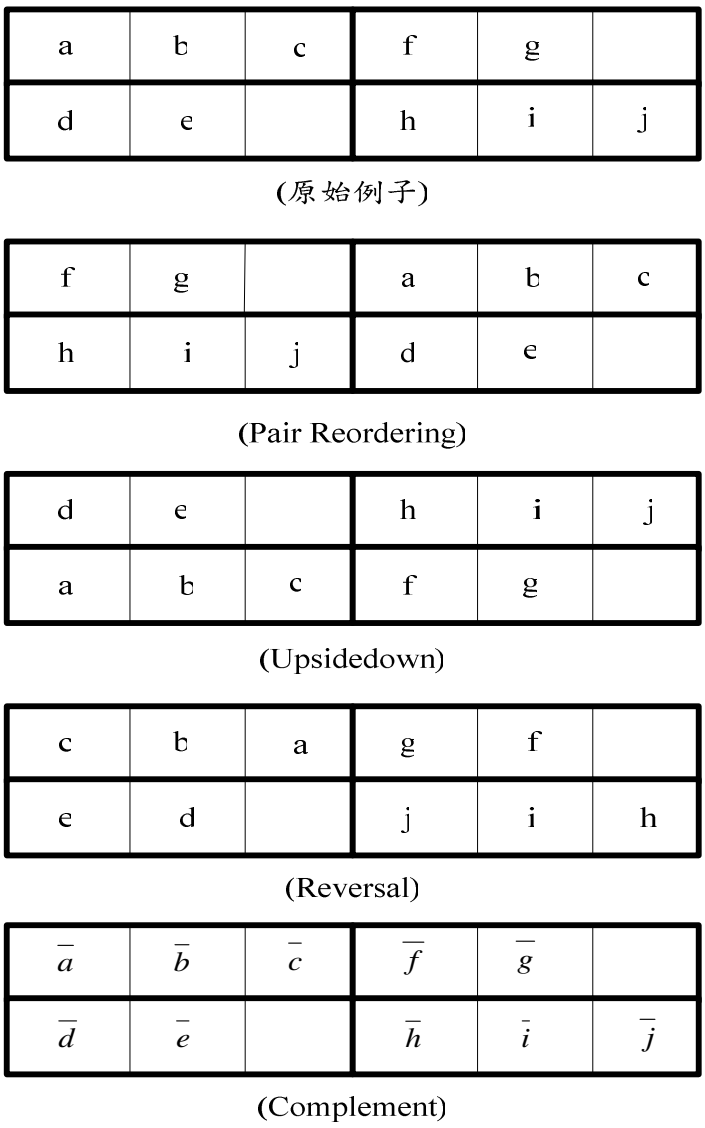
這些候選子代例子其提示基因與空白基因完全相同者稱之為重覆盤面，而不

屬於上述完全相同例子，本論文引用 Ling Zhao 的說法，將其稱之為同型(isomorphisms)。由於 isomorphisms 組合數過於龐大，在本論文中為了簡化處理重覆例子的時間，將其視為不同例子，不另外設計篩檢演算法排除。

還有底下是 isomorphisms among PCP instances 中四種變形：

1. Pair Reordering：藉由重新排列同一例子中的 pair。
2. Upsidedown：將每一組 pair 的上下 string 同時互換。
3. Reversal：將每一組 pair 的上下 string 同時左右顛倒。
4. Complement：將每一組 pair 的每一 string 中，以 0 取代 1，以 1 取代 0。

底下以圖十五來說明上述四種變形之後的情況。



圖十五：isomorphisms among PCP instances

由於這四種變形差異性太大，若要逐一去比對，則會花費太多時間，故本論文為了減少這四種變形出現，在產生子代時會從第一個 pair 的 string 長度中的第二個格位數上開始繁衍，則可以避免產生第四類 Complement 變形，而即使產生如圖十五中的第一類與第二類變形時，又因為不符合我們所設定的規則即 pair 1 的 string 必須全為提示基因，不可為空白基因的前提之下，則可大大降低第一類與第二類變形發生的機率。第三類變形須要左右對稱情況下才會產生。因此，基於以上的作法，PCP 例子會隨著 size 與 width 逐漸增大情況之下，產生的可能性

微乎其微，在本研究中將不針對其它變形的排列組合納入重覆性篩檢的考量。

一般性的重覆檢查，就是將繁衍出的子代經由合法性篩檢通過的例子讀入到程式後，將其轉換為 Hash table 的資料結構來存放，接著利用 Hash table 的特性，將每個例子依 Key 值作為分類，相同 Key 值所對應到的所有例子都存放在 Value 之中(如圖十七所示)，也就是將新產生的例子與先前繁衍產生的例子做比對，如果有重複出現的例子，則從子代中刪除。經過一般性重複性的篩檢，會使得該特定 size 與 width 所繁衍的子代不具有重複的情況，以便協助後續演算法中快速的比對及運作。

經過上面兩步驟後，產出符合該特定 size 與 width 的合法候選例子之後，還必須對這些例子進行「是否具有解」的篩檢，使之成為有效的 PCP instance。

為了要做到有解的判斷，我們首先在網站上取得「Solver」的程式原始檔，利用 Solver 提供可以針對輸入檔(.txt)做一解題的特性，改寫為新增一功能可針對基因繁衍的例子來進行解題，因為 Solver 原先設計可以針對輸入檔做解題的特性是假設這些輸入檔的每一個例子都通過過濾器，但由於我們產生的例子並無經過過濾器，故可能有些無解的例子包含在其中，故改寫為針對基因繁衍的例子，需先經過該 Solver 所內建的過濾函數，過濾掉絕對不可能有解的例子之後，才正式開始去解可能會有解的例子。而在執行解題完成後，則將輸出檔分為有解、無解與待解(在某一定層數找不到答案後，便存放在此文字檔中)三種。

此外還針對本論文所提出困難度指標，即搜尋到有解的最低階層為止的解題時間除以該階層所有的解答數目，與前人當初所定義的困難度為解題的長度來判定有所差異。而 Solver 程式在設計之初，基於設定困難度指標為解題的長度，而使用 iterative deepening DFS(Depth-First Search)演算法，若要改為本論文所提出的

困難度指標，則需將 Solver 改為採用 BFS(Breadth-First Search)的演算法，然而 BFS 必須用較多的空間來儲存節點：當搜尋深度為 n，且旁枝因素(Branch factor)—即每個節點平均的子節點個數為 d 時，BFS 最多需儲存 d^n 個節點，故在較少階層時可以較快搜尋，但在針對 PCP 例子時則無法適用，其原因在於拜訪的子節點數目過於龐大，而 BFS 無法儲放超過一定階層拜訪的節點數。基於以上因素，本論文改寫 Solver 程式使用另一種近似困難度指標的計算方式，即將解題時間除以有解的個數，並沿用具有在拜訪完後便將先前拜訪的節點丟棄特性的 iterative deepening DFS 演算法。當然一個 PCP 例子的各個解可能位於不同層，但一來是其解答個數通常不多之外，也和 Solver 所採用的 iterative deepening DFS 有關，因為此程式設定搜尋到最深階層為 320 層，每次從第 0 階層開始搜尋，在前 20 層若是找不到有解的答案時，便會疊代加深 20 層，找到時便停止，沒有找到便再疊代加深 20 層，直到設定的門檻值為 320 層為止，因此若有一組以上的解答時，必定是差距在 20 層之內。經過以上兩步驟的改寫，在本論文中將此程式命名為「Hard index solver」，利用 Hard index solver 作為篩檢基因繁衍的子代例子，並增加困難度指標，使得本論文所開發的 PCP 困難例子產生器具有快速檢驗其合法有解性質的實質意義。

將基因繁衍法所產生的子代候選例子輸入到 Hard index solver 篩檢後，只可能產生以下三種狀況的其中一種：具有困難度指標的子代例子、無解例子、待解子代例子。